Deep representation learning for time series averaging

Tsegamlak Terefe Debella

Deep Representation Learning for Time Series Averaging

Tsegamlak Terefe Debella

A Dissertation

Submitted to Université de Haute Alsace
Institut de Recherche en Informatique, Mathématiques, Automatique et Signal.

In Partial Fulfillment of the Requirement for
a Doctor of Philosophy in Computer Science

Jury

Prof. Laetitia Jourdan
Université de Lille
Examinatrice

Dr. Charlotte Pelletier
Université Bretagne Sud
Examinatrice

Prof. Nicolas Meger
Université Savoie Mont Blanc
Rapporteur

Prof. Engelbert Mephu Nguifo
Université Clermont Auvergne
Rapporteur

Supervisors

Prof. Germain Forestier
Université de Haute Alsace
co-directeur

Dr. Jonathan Weber
Université de Haute Alsace
co-directeur

Dr. Maxime Devanne
Université de Haute Alsace
co-encadrant

Dr.-Ing. Dereje Hailemariam
Addis Ababa University
co-director

Date of Submission: May 6, 2022

Acknowledgments

This Ph.D. dissertation is a result of a collaboration among different organizations and individuals. With this in mind, I would like to first thank the Embassy of France for Ethiopia and African Union and the former Ethiopian Ministry of Science and Higher Education (MOSHE) for funding the Ethio-France Ph.D. program. Moreover, I would also like to thank Madam Ouloufa Dorani, Madam Sophia Nicée Samba, and Dr. Esayas Gebreyouhannes for handling the administrative issues associated with the program. Without their continuous support, this dissertation would have not come to a realization. Besides the funding organizations, I would also like to thank Université de Haute Alsace, Université de Strasbourg, and Addis Ababa University, Addis Ababa Institute of Technology for creating and sustaining the collaboration through difficult times.

In addition, I would also like to thank my supervisors Prof. Germain Forestier, Dr. Jonathan Weber, Dr. Maxime Devanne, and Dr.-Ing. Dereje Hailemariam for their patience, understanding, and guidance. Moreover, I would also like to thank Miss Bethelehem Seifu Shawel and Prof. Sofie Pollin for collaborating with us in a joint work that aims to show a practical aspect of one of our proposal. Furthermore, I would also like to thank the creators of the UCR archive for openly providing the datasets that were used for experimental evaluations, Moreover, I would also like to thank Université de Strasbourg for providing the cluster of GPUs (Mésocentre) that we used to conduct extensive experiments. Last but not least, I would like to thank my family for their wise counsel, sympathetic ears, and for keeping everything running while I was traveling for my study. I am in debt to the kindness and understanding they have shown me throughout my study.

Abstract

The estimation of an optimal time series average has been studied for over four decades. In practice, time series averages are often key inputs to most temporal data mining techniques. For instance, in one nearest centroid classification, time series averages serve as a template for the identification of class membership. Additionally, in most time series clustering techniques, averages define the center of gravity for cluster formation. To this end, in practice, the constraints placed on time series averages are not trivial. In this regard, time series averages are expected to preserve the most descriptive features (shapes) that are observed in the averaged set. They are expected to preserve the shapes while minimizing the discrepancy between themselves and members of the averaged set. However, in reality, meeting the demands of such constraints is not trivial due to temporal distortion (shifts) that could arise for various reasons. For instance, a difference in the behavior of observed entities, difference in the sampling rate of sensors, and a difference in size (shape) of objects from which temporal datasets are extracted are some examples of sources of temporal distortion. In practice, such sources of temporal shifts often misalign most descriptive shapes observed in an averaged set. To this end, in most cases, an arithmetic mean becomes a sub optimal estimate for practical considerations.

With this understanding, over the course of four decades, a range of time series averaging heuristics have been proposed. In general, all the proposed averaging heuristics suggest aligning members of an averaged set prior to estimating an average. However, even if the alignment minimizes the impact of temporal distortion, it often introduces additional challenges. For instance, all pioneering averaging heuristics that utilize Dynamic Time Warping (DTW) as an alignment technique have a computational complexity that is directly proportional to the number and dimension of the averaged series. With such observations in mind, in this dissertation, we avoid approaching time series averaging as a multiple alignment problem. On the contrary, we see time series averaging as a generative challenge. To this end, we first proposed to augment time series averages from the latent space of neural networks. In this regard, we first proposed to augment the averages from the latent space of variational and non-variational autoencoders. After accessing these proposals, we then modified the overall architecture and placed constraints that further refined the quality of extracted latent space features. In this aspect, we proposed multi-tasking autoencoders that performed multi-class classification and reconstruction. We mainly utilized the classifier to force the latent space features dense and separable which in turn mimicked the effects of multiple alignments. With this modification, we are able to provide time domain estimates that are far better than the arithmetic mean. However, we also noticed that the multi-tasking setup can further be refined by addressing limitations observed in the objective function. After addressing this limitation, we are able to provide a state-of-the-art latent space registration. Moreover, we are also able to provide time-domain estimates that are far better than the estimates of a time domain arithmetic mean and some of the DTW based averaging heuristics.

Contents

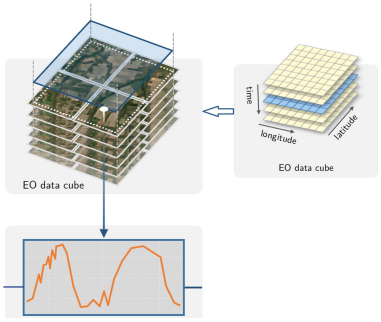
*

List of Figures

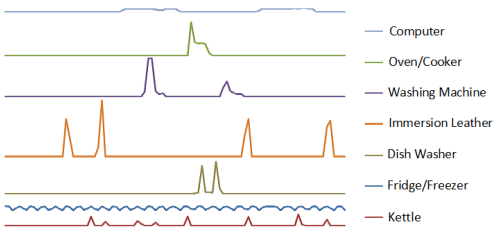
List of Tables

1 Introduction

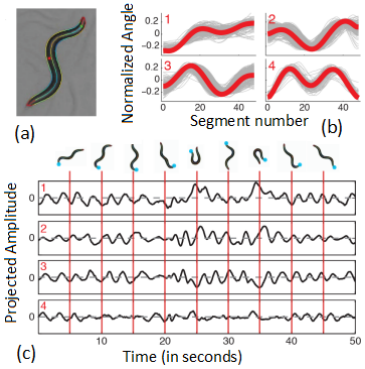
In today's data driven world, time series are considered as one of the most intensively investigated datasets [1]. The main driving force behind this reality is the possibility of defining the series from a seemingly unrelated topic [2]. For instance, Figure 1.1 shows how time series can be defined from a range of applications scenarios. In this regard, in Figure 1.1 (a), a time series is defined by taking pixel values of stacked satellite images [3]. Moreover, Figure 1.1 (b) shows time series defined from sensor that were taking the power consumption measurements of home appliances. Finally, in Figures 1.1 (c) & 1.1 (d), time series were respectively defined by either studying the shapes made by a moving earth worm or from distance measurements taken between a central reference point and points on the contours of a segmented image of chicken [4]. In overall, the examples given in Figure 1.1 span the application domains of: remote sensing (a), power consumption monitoring and classification (b), behavioral genetics (c) and image classification (d). In general, despite the domain from which the series gets extracted, a time series of the form $X = \{x_1, x_2, x_3, ..., x_M\}$ is defined from a set of



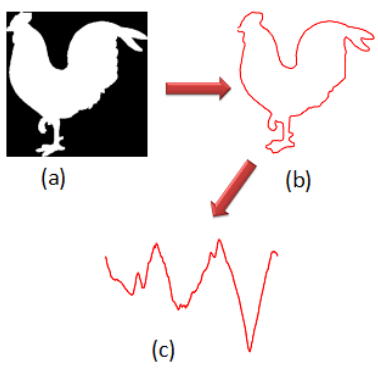
(a) Time series from stacks of satellite images.



(b) Power consumption of home appliances.



(c) Shapes made by the movement of an earth worm.

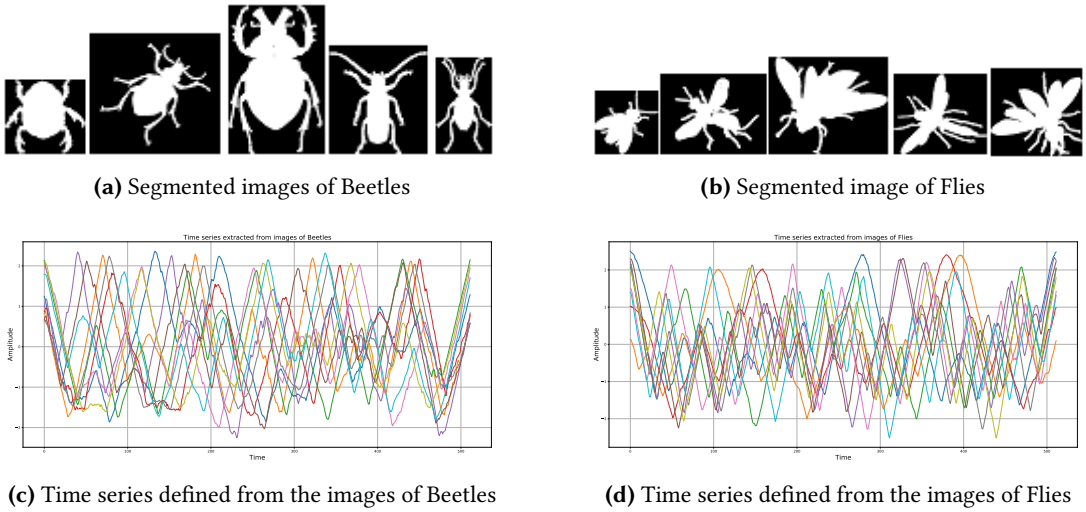


(d) Time series from segmented image of a chicken.

Figure 1.1: Time series extracted from different application scenarios [3], [4]

ordered observations where the ordering can be based on any of the common independent variables, i.e., space, time, frequency, etc [1], [5]. Moreover, depending on the dimensions of the individual observations (x_i), time series can also be further divided into two broad categories, i.e.,univariate

and multivariate. In this regard, in this dissertation, we consider a time series to be univariate if the individual observations (coordinates) are defined from real numbers ($x_i \in \mathbb{R}$). On the contrary, we consider a time series to be multivariate if $x_i \in \mathbb{R}^l : l > 1$. With these definitions, on one hand, we expect the ordering of the individual coordinates to define descriptive features (shapes). On the other hand, most temporal data mining techniques often relay on such descriptive shapes to meet their desired objectives [6]–[8]. To this end, temporal data mining techniques often emphasize on devising techniques that could capture such unique features to their advantage. In this regard, some rely on complicated non linear transformations [6], [7]. On the contrary, others rely on domain transformation [9], [10], approximation [11], warping [12], etc. However, despite the difference in the feature identification techniques, most of them often try to directly or indirectly address one common challenge, i.e., the impact of temporal distortions (time shifts) [9], [11], [13]–[17]. In practice, temporal shifts are evident in temporal datasets for various reasons. For instance, if we reconsider the example in Figure 1.1 (b), we do not expect the owners of individual home appliances to have similar daily routines. To this end, we do not expect power measurements of similar appliances to have peaks on identical time stamps. With this understanding in mind, researchers often propose different mitigation techniques. For instance, in distance based time series classification tasks, classifiers use time warping to define elastic distance measurement functions [13]. On the contrary, other alternatives try to overcome time shifts through approximation [11] and domain transformation [9]. In order to further elaborate on the implication of temporal misalignment and the need for mitigation techniques, we can consider univariate temporal datasets extracted from the segmented images of Beetles and Flies as an example [2]. We have given a sample of the images and their respective time series formats in Figure 1.2. In order to extract the time series, the colored images of Beetles and Flies were initially



(a) Segmented images of Beetles

(b) Segmented image of Flies

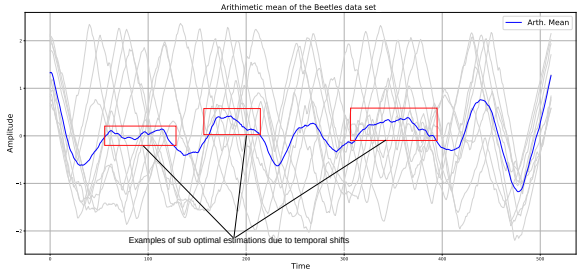
(c) Time series defined from the images of Beetles

(d) Time series defined from the images of Flies

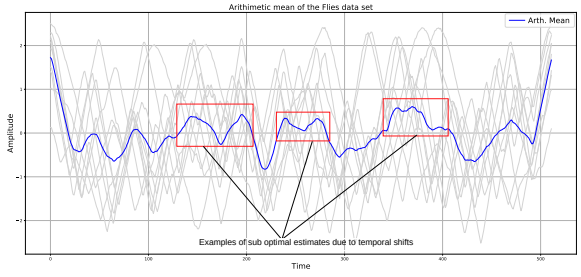
Figure 1.2: Time series defined from segmented images of Beetles and Flies [2], [4]. Even though the extracted time series representing each species show similar patterns, there is a significant temporal distortion due to image rotation and shape and size variations.

taken. Following this, the colored images were converted into a black and white format by segmenting the images through pixel value thresholding. Finally, the euclidean distance between a reference point within the white area and sample points on the contours of the segmented images were taken

at a fixed angular steps. In reality, the euclidean distances defined the amplitudes of the extracted temporal datasets. Moreover, the angular location of the sample contour points defined the order of the amplitude values [4]. In general, according to Figures 1.2 (c) & 1.2 (d), the extracted time series presented features that are unique to each species. In this aspect, the time series representing the Beetles show a relatively sharper and higher peak values. On the contrary, the time series corresponding to Flies have smaller and smoother peaks. Consequently, researchers proposed to utilize the extracted temporal datasets for the classification of the images of Beetles and Flies. However, despite the per-species (per-class) unique features, there are significant misalignments among descriptive shapes of the extracted time series. The misalignments are mainly caused by image rotations and intra species shape and size differences. In practice, such misalignments often become a major impairment to most common distance based classification techniques. For instance, if we consider nearest neighbors and nearest centroid classification techniques [18], [19] as an example, the misalignments could either increases the euclidean distance between members of the same class or the distance between members of a class and their respective per-class templates (centroids). Consequently, such temporal shift mainly contributes to most of the misclassification (classification errors) [12], [13]. In this aspect, if we consider Nearest Centroid Classification (NCC), per-class templates (centroids) are often defined by taking per-class averages. When this is the case, temporal misalignment often make an arithmetic mean to often be a sub optimal representative [13], [18]. This is better demonstrated in Figures 1.3 (a) & 1.3 (b) where we have plotted the per-class arithmetic averages of the Beetles and Flies time series. In the figure, we have marked segments of the arithmetic means that are significantly distorted with red boxes. These shape distortions are introduced due to the alignment of peaks and troughs through time shift. Consequently, for the given time series, a NCC based on arithmetic means results a 30% classification error [20].



(a) Arithimetic mean of time series corresponding to Beetles



(b) Arithimetic mean of time series corresponding to Flies

Figure 1.3: The arithmetic means of time series extracted from the segmented images of Beetles and Flies. The segments marked with red boxes indicate significant shape distortion evident due to misalignment of peaks and troughs caused by temporal distortion.

Practically, the use of such time series averages is not limited to the summarization of classes. In this aspect, time series averages are useful in formulating clusters or assess patterns in a sets of time series [18], [21]. For instance, in most time series clustering, cluster centroids (averages) are at the center of gravity for cluster formation [6], [21]. For instance, K-Means clustering and its variants utilize cluster centorids (averages) to identify cluster membership [22], [23]. Moreover, in a supervised setup, class averages have been utilized to identify class membership [13], [20]. In addition to this, in a more practical example, [24] showed the importance of time series averages in climatology. In this

aspect, the authors showed how the average of time series that are defined from the width of tree rings are important to the study of climate changes in dendroclimatology. Moreover, they also showed the importance of time series averages on estimating the coverage area of meteorological measurements, i.e., temperature, humidity, precipitation, etc. In another domain, [14] showed the importance of time series averages on the study of Evoked Potential (EP) that are taken from a human scalp. In practice, EP measurements that correspond to a human brain response to external stimuli were found to be noisy and weak. To this end, [14] propose to average EP measurements that corresponded to similar stimuli. In general, we can go on and present additional examples which could further show the importance of time series averages in temporal data mining tasks [15], [16], [25], [26]. However, if we pause at this point and look for a common ground, we can observe that proposal relying on time series averages collectively agree on the need for a quality estimates [14]–[16], [22], [23], [25]. Moreover, they also collectively agree on the negative impact of temporal shifts on the quality of estimated averages [13]–[16], [25], [27], [28].

With these observations, in this dissertation, we aim to address the impact of temporal distortion on quality of univariate time series averages. To meet this objective, we initially assess the limitations observed in previously proposed averaging heuristics. Following this assessment, we present a novel neural network based time series average estimation technique. We base our proposals on neural networks for two main reasons. First, neural networks provide optimization platforms that are capable of generalizing over a range of unseen data sets. Thus, they provide the possibility of avoiding costly re-runs through transfer learning [25]. Furthermore, neural networks provide a range of tunable hyper parameters that provide additional control on the way the averaging objective functions is optimized [29]. With this said, we will next formally introduce the univariate time series averaging problem.

1.1 Statement of the Problem

The computation of an optimal univariate time series average has intensively been studied for over four decades [14]–[16], [25], [28], [30]. The main driving factor behind these studies is the importance of the averages in cluster (class) based temporal data mining techniques [6], [13], [21], [23]. In such algorithms, time series averages are expected to preserve the most descriptive shapes that are observed within an averaged set. Moreover, they also expect the averages to preserve shapes while minimizing the discrepancy between themselves and members of an averaged set [13], [23]. Consequently, given an averaged set that has K members and a distance function d, time series averaging heuristics are expected to minimize the discrepancy between an estimated average $\mu \in \mathbb{R}^N$ and members of the averaged set $X_j \in \mathbb{R}^M$ (1.1), where $M \leq N$. In other words, they are expected to minimize (1.1).

$$F(\mu) = \frac{1}{K} \sum_{j=1}^{K} d(X_j, \mu) \tag{1.1}$$

In practice, it is often challenging to minimize (1.1) in time domain due to the presence of temporal distortion. To this end, averaging heuristics are often expected to perform some sort of temporal align-

ment as a pre-processing step. In this regard, pioneering techniques often relied on either Dynamic Time Warping (DTW) or diffeomorphism to meet this demand [14]–[16], [25], [28], [30]. However, despite the difference in the utilized alignment techniques, we can generalize the steps they take using two or three basic procedures. In this regard, first they consider an averaged set as a group of K univariate time series (vectors) such that $X_i \in \mathbb{R}^M$. Following this, the techniques try to minimize the discrepancy among members of averaged sets that could be evident due to temporal distortion either through time warping or diffeomorphism. In reality, the warping or morphing will transform the averaged series into a τ dimensional space where $\tau \geq M$. Moreover, since the alignment or morphing is expected to minimize temporal distortion, it is expected to increase the density (D) of the transformed set. In other words, given a set of K warped (morphed) version of the original series that are in $\mathbb{R}^\tau$ ($Y_i \in \mathbb{R}^\tau$), then the alignment (morphing) is expected to minimizes (1.2) [16], [23], [25].

$$D = \frac{1}{\tau}\sum_{l=1}^{\tau}\left(\frac{1}{K^2}\sum_{i=1}^{K}\sum_{j=1,j\neq i}^{K}(Y_i - Y_j)^2\right) \tag{1.2}$$

In theory, (1.2) can be minimized by simultaneously warping or morphing all members of the averaged set. However, in reality, this is difficult to realize for two main reasons. First, with the currently available warping or morphing techniques, multiple warping is computationally intractable [14]–[16], [25], [31]. Moreover, even if such techniques become available, there is no clear pre known ground truth for the multiple alignment. To this end, currently available averaging techniques rely on an indirect approach and often propose to minimize (1.2) by registering the averaged series to a template (land mark) which is often their τ space arithmetic mean. For this reason, the next common step taken in time series averaging is refining the registration of the averaged set to the selected land mark trough iterative warping (morphing). Thus, after the iterations, an average is estimated by taking the arithmetic mean of the warped or morphed series. To this end, in some cases, averaging techniques have to propose a way that will re-project the estimates to the time domain as a final step, i.e., if $\tau > M$ [16]. However, in all cases, the optimality of the estimated averages are only guaranteed in the registered space ($\mathbb{R}^\tau$). To this end, with currently available averaging techniques, there is an inherent assumption that estimated averages will be transformed to the space they were estimated from (their registered space) prior to any utilization [14]–[16], [25]. With this in mind, some pioneering averaging heuristics associate the temporal alignment step to a distance function (d) [32], [33]. In practice, techniques that mainly fall in this category are DTW based. Consequently, in such averaging techniques, the distance function d given in (1.1) gets modified to (1.3), i.e., the DTW distance (metric). In (1.3), δ_p is the squared $L2$ norm of the distance between a warped series ($Y_i \in \mathbb{R}^\tau$) and its warped space arithmetic mean ($\mu \in \mathbb{R}^\tau$), where the warping is along a DTW warping path p. In practice, when this is the case, (1.1) is commonly called the Fréchet function [16], [31].

$$d(Y_i, \mu) = \delta_p(Y_i, \mu) = ||Y_i - \mu||_{l2} \tag{1.3}$$

However, in reality, integrating an alignment techniques into the averaging objective function (1.1) often leads to major complications [14]–[16]. This is because, in most cases, alignment techniques have undesired mathematical properties that make the optimization of (1.1) relatively challenging. For

instance, (1.1) becomes non-smooth, non-convex, and a computationally intensive objective function when it is integrated with DTW distance [16], [31], [34]. To this end, in recent years, proposals have started to separate the distance function (d) from the underlying alignment technique. In such cases, the distance function d is often taken as the squared $L2$ norm of the difference between the estimated mean and transformed members of the averaged set, i.e., (1.4). In (1.4, $Y_i = \{y_1,\ y_2\ , \ldots,\ y_M\}$ and $\mu = \{\eta_1,\ \eta_2,\ \ldots,\ \eta_M\}$ are a transformed series and the arithmetic mean of the transformed averaged set. When this is the case, equation (1.1) is commonly called the Within Group Squared Sum (WGSS).

$$d(Y_i, \mu) = \ ||Y_i - \mu||_{l2} = \sum_{j=1}^{M} (y_j - \eta_j)^2 \tag{1.4}$$

Generally, in all cases, optimizing (1.1) is often not trivial and the level of the difficulty is often highly correlated with the underlying temporal alignment technique. In this regard, DTW based averaging techniques are often considered to be relatively challenging [34], [35]. This is mainly because, in time series averaging, we desire to minimize (1.2) through multiple alignment. However, the customization of DTW in such a manner is practically known to be Non-deterministic Polynomial (NP) hard [16], [34], [36]. Consequently, in practice, most DTW based averaging techniques rely on heuristic rather than exact solutions. However, in reality, proposed heuristics by themselves often induce additional complication. For instance, if one proposes to tackle the averaging problem using pair-wise DTW warping, each pair-wise warping will significantly increases the dimension of the final estimate [14], [15]. Furthermore, to make matters worse, the dimension of the final estimation is dependent on two external factors, i.e., the size of the averaged set and the dimension of the individual members [14], [15]. This could in turn further intensifies the computational and storage requirement of such averaging approaches. Additionally, even if we some how overcome this challenge, DTW based averaging heuristics are expected to optimize a non smooth and non convex objective function that are practically challenging to optimize [34], [37]. However, this by no means imply that their counterparts are problem free. In this regard, diffeomorphic approaches are often implementation wise complex as compared to their DTW counterparts. Moreover, due to their complexity they often place additional expectations from the underlying optimization setup. In some cases, the additional expectations could have a negative implication on the convergence of the optimization process [20]. With these observations in mind, in this dissertation, we aim to address the following key questions:

- Can we see the averaging problem from a different perspective and reformulate it as an augmentation or generative challenge?
- If the answer to the former question is yes, then we ask ourselves can we approach it using neural networks?

We find these two questions to be logical rather than random. This is because, if we carefully observe previous averaging proposals, we can correlate the overall averaging steps to steps taken in augmentation or generative problems [38]–[40]. In practice, generative models such as the Variational AutoEncoder (VAE) and Generative Adversarial Network (GAN) have shown that it is possible to generate synthetic datasets by modeling inputs using probabilistic models. In other words, they

aim to generate synthetic datasets that significantly resembles their inputs by taking samples from variables following a certain distribution. This is well in line with the first thing we expect from a time series average, i.e., an average should preserve shapes observed within an averaged set. Moreover, in time series averaging, we can assume the aligning step as the process of formulating a suitable augmentation space which correlates to the modeling of input series with probabilistic distributions in generative models. Furthermore, we can assume the arithmetic averaging of aligned series to be an augmentation or generative step which could be correlated to selecting a sample from a probabilistic model in generative models. However, unlike the generative or augmentation problem, in time series averaging we expect the augmentation space to be dense such that it aligns with the requirement of (1.2). Thus, the question now becomes, how can we learn such augmentation spaces? Moreover, what kind of neural architecture can meet this requirement? We ask the latter question since its answer will help us to introduce transfer learning into the averaging problem. This is because, neural network based optimization setups are known to generalize over a range of unseen datasets which in turn will help us to avoid costly re-runs. With these questions in mind, we will next present the general and specific objectives of this dissertation.

1.2 Objectives

1.2.1 General Objective

In this PhD dissertation, we mainly focus on computing an optimal univariate time series average using a flexible and novel neural network optimization setup. We consider an estimate to be optimal if it minimizes the aggregated distance between itself and members of the averaged set while preserving shapes observed in the averaged set.

1.2.2 Specific Objectives

To meet our general objective, we set the following specific objectives;

- Deeply understand, assess previous proposals and address associated limitations.
- Identify and propose an easily deployable neural network architecture that is suitable for time series average augmentation.
- Investigate and evaluate the implications of parameters affecting the average estimation, i.e., objective functions, the augmentation processes, hyper parameters.
- Identify an application scenario that demonstrates the implication of proposed approaches.

1.3 Scope

In this dissertation, we focus on the estimation of optimal univariate time series averages. Moreover, we constrain our study to a set of univariate time series that have fixed length. To the best of our knowledge, in practice, the averaging of multivariate and variable length time series are often

approached indirectly. For instance, [36] proposed to estimate averages in unconstrained manner by compressing estimates generated by a technique that is built for univariate and fixed length temporal datasets. In general, we believe the guideline for generating averages for the two categories is debatable and requires a separate investigation.

1.4 Organization

We have organized the dissertation into five additional chapters. In chapter two, we present a detailed review of previously proposed heuristics and concepts that are crucial to our proposals. Following this, in chapters three and four, we present the steps and the reasoning behind our proposed approaches. We will also use these chapters to present the experimental setups and evaluations of our proposals. In chapter five, we present a practical scenario that demonstrate the practical implication of our proposed approaches. Finally, we conclude our study and give a direction for possible future researches in chapter six.

2 Background and Related Works

In this chapter, we start our discussions by reviewing some literature associated with the Dynamic Time Warping (DTW) algorithm and its variants. In reality, the DTW is integrated into more than half of the pioneering averaging heuristics. Thus, we believe the reviews will assist the reader with understanding the steps taken in such averaging techniques and associated challenges. Following the discussions associated with DTW, we will present averaging heuristics based on DTW according to their order of appearance. Following this, we present concepts related to neural networks, layers of neural networks, key hyper-parameter setup techniques, and some well-known neural network architectures. The discussions in this subsection serve as a basis for our proposed approaches presented in chapter three. Moreover, they will also aid the reader in easily understanding terms in a pioneering neural network-based time series averaging heuristics presented at the end of this chapter.

2.1 The Dynamic Time Warping

The Dynamic Time Warping (DTW) was introduced as a temporal alignment technique for voice recognition systems [32]. In practice, such systems are often expected to recognize voice commands that are spoken by: people with a different accent, people that give different emphasis to similar words, and people that take different duration to utter similar words. With these difficulties in mind, [32] proposes to treat discrete samples of the voice commands as univariate time series. Moreover, the authors also aimed to time warp (stretch) the time series representations such that the discrepancies introduced by the way voice commands get spoken are minimized. To achieve this objective, DTW introduced two kinds of distance matrices, i.e., the local and global cost matrices. In overall, given a template time series $X = \{x_1, x_2, \ldots, x_N\}$ and an uttered voice command $Y = \{y_1, y_2, y_3, \ldots, y_M\}$, a DTW local cost evaluates the distance between each and every coordinate values of X and Y, i.e., $x_i \in X$ and $y_i \in Y$. To perform this computation, DTW first places any two aligned series along the columns and rows of an $M \times N$ ($N \times M$) matrix. It then fills each cell of the $M \times N$ matrix with the distance between the coordinate values of the aligned series. However, in practice, it was found that the type of the selected distance function could easily become a source of non-smoothness in DTW [32], [34], [35]. To this end, the authors proposed to use the squared euclidean distance due to its convex nature [34].

To further elaborate the overall computation process, we can consider the DTW warping of two misaligned univariate time series $X = \{1, 1, 5, 5, 5, 1, 1\}$ and $Y = \{1, 5, 5, 5, 5, 1, 1, 1\}$ as an example. The local cost values of the series are shown in Table (2.1), where we computed the values of each cell using a pair-wise squared euclidean distance. However, in reality, the local cost matrix by itself does not reveal an optimal warping path. To this end, DTW introduces the concept of the global cost matrix which is mainly used to identify group of cells that connect $(0,\ 0)$ to $(M,\ N)$ $((N,\ M))$. In practice,

Table 2.1: Local cost matrix of two DTW warped series

		Time Series Y							
		1	5	5	5	5	1	1	1
Time Series X	1	0	16	16	16	16	0	0	0
	1	0	16	16	16	16	0	0	0
	5	16	0	0	0	0	16	16	16
	5	16	0	0	0	0	16	16	16
	5	16	0	0	0	0	16	16	16
	1	0	16	16	16	16	0	0	0
	1	0	16	16	16	16	0	0	0

such a group of cells is called a warping path. Moreover, each entry of a warping path is known as DTW associated coordinates. Practically, given a global cost matrix, one could randomly identify a range of possible paths connecting $(0,\ 0)$ to $(M,\ N)$ $((N,\ M))$. However, the random grouping of global cost matrix cells often does not guarantee an optimal warping of the aligned series. To this end, the computation of a DTW global cost matrix entries and the identification of optimal warping paths are guided by constraints. For instance, after a DTW warping, we aim to preserve the precedence of coordinate values observed in the original series. In other words, we do not desire a DTW warping path that entangles coordinate values. With such consideration in mind, DTW places the following two key constraints that govern the allowable warping paths and the way the cells of a global cost matrix get computed:

- Given an $M \times N$ DTW global cost matrix, a warping path must start at $(0,\ 0)$ and end at $(M,\ N)$. This constraint preserves the start and end values of the original time series.
- If cell $(i,\ j)$ is identified as an entry of a warping path, then the next entry of a warping path can only be one of the following three possible entries: $(i+1,\ j)$, $(i,\ j+1)$, or $(i+1,\ j+1)$. In other words, if the entry of a warping path is $(i,\ j)$, then it could only have traversed trough: $(i-1,\ j)$, $(i,\ j-1)$, or $(i-1,\ j-1)$ for $0 \leq\ i\ \leq\ M$ and $0 \leq\ j\ \leq\ M$. These constraints avoid the entanglement of DTWwarped series and also preserve the precedence of the original coordinate values in the warped series.

Generally, among the two constraints, the second constraint has a strong tie with how the global cost values get computed. In reality, based on this constraint, we can segment a DTW global cost matrix into three regions from which the entries of a warping path get obtained. In this regard, the first region from which a warping path entry gets obtained are cells located on the first column of a global cost matrix. In this case, to include cell $(i,\ j)$ as an entry of a warping, a warping path must traverse through a cell located at $(i,\ j-1)$. Thus, the cost of including $(i,\ j)$ must also include the cost of its only predecessor $(i,\ j-1)$. In the second scenario, the entry of a warping path can originate from the first row of a global cost matrix. In this case, the cost of including a cell $(i,\ j)$ must account for the cost of its only allowable predecessor $(i-1,\ j)$. Finally, an entry of a warping path could be extracted from a location different from the first row or column of a global cost matrix. When this is the case, in order to include cell $(i,\ j)$ within a warping path, it must through one of the following cells: $(i-1,\ j)$, $(i,\ j-1)$, or $(i-1,\ j-1)$. Thus, for this case, the cost of $(i,\ j)$ must include the cost

of one of its three predecessors. In DTW, these observations are formalized using the mathematical expression given in (2.1) where $GC_{i,j},\ LC_{i,j}$ are respectively the global and local costs of cell $(i,\ j)$.

$$GC_{i,j} = \begin{cases} LC_{i,j} + Min\{GC_{i-1,j}, GC_{i,j-1}, GC_{i-1,j-1}\}, & \text{if \{i,j\}} \neq 0, \\ LC_{i,j} + Min\{GC_{i,j-1}\}, & \text{if i=0,} \\ LC_{i,j} + Min\{GC_{i-1,j}\}, & \text{if j=0} \end{cases} \tag{2.1}$$

With this equation in mind, we can now revisit our previous example and compute their corresponding global cost values as shown in Table (2.2). However, in practice, DTW does not compute the local and global cost matrices separately. On the contrary, DTW computes the local and global cost values on the fly. This is because, in reality, the second line of equation 2.1 is in line with the concepts of Dynamic Programming (DP) [32], [37]. To this end, in practice, DTW uses DP to recursively compute the global cost matrix and identify the optimal warping path without any visual aid. In this regard, given a global cost matrix $\triangle(X,\ Y) \in \mathbb{R}^{M,\ N}$, DTW can start the search for an optimal warping path from $(0,\ 0)$ which only has a local cost value. Following this, DTW can takes a step into one of the three allowed directions, i.e., $\{(i+1,\ j),\ (i,\ j+1),\ (i+1,\ j+1)\}$. However, since a step corresponds to an additional cost, in DTW's search for an optimal warping path, a step in a given allowable direction is taken if it incurs a minimal additional cost. Thus, by consecutively stepping on such cells, DTW can identify a warping path with a minimal alignment cost. In overall, the concepts discussed so far are better demonstrated using Table 2.2, i.e., using the global cost matrix associated with the two example series: $X = \{1, 1, 5, 5, 5, 1, 1\}$ and $Y = \{1, 5, 5, 5, 5, 1, 1, 1\}$.

Table 2.2: Global cost matrix of two DTW warped series

		Time Series Y							
		1	5	5	5	5	1	1	1
Time Series X	1	0	16	32	48	54	54	54	54
	1	0	16	32	48	54	54	54	54
	5	16	0	0	0	0	16	32	48
	5	32	0	0	0	0	16	32	48
	5	48	0	0	0	0	16	32	48
	1	48	16	16	16	16	0	0	0
	1	48	32	32	32	32	0	0	0

In addition to demonstrating how global cost matrices get computed, Table 2.2 also shows the possibility that a DTW global cost matrix could have a group of cells with equal global cost values. Consequently, in practice, DTW's optimal warping path is often not unique. This fact is further demonstrated in Figure 2.1 where we have plotted the possible warping paths associated with the global cost values computed in Table 2.2. In Figure 2.1, we have shown the different warping paths using different sets of colors. These possible warping paths give different warping of the aligned series. However, all of them warp the two series at zero alignment costs (DTW distance). In practice, this cost can quickly be referred from cell $(M,\ N)$ of the global cost matrix. Even though DTW proved to be useful in different algorithms [14], [15], [21], [23], it also presented some undesired behaviors. For instance, in practice, DTW warping paths that are far from the *"diagonal"* of a global cost matrix

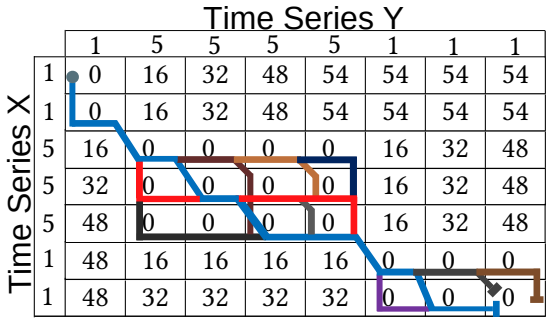
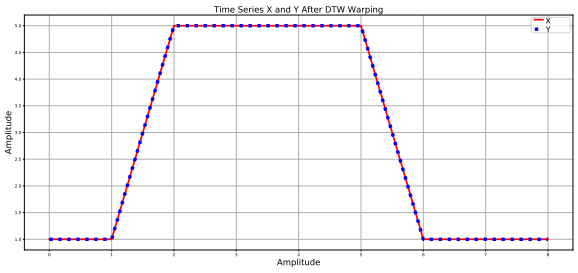


Figure 2.1: Multiple similar cost DTW warping paths for the time series $X = \{1, 1, 5, 5, 5, 1, 1\}$ and $Y = \{1, 5, 5, 5, 5, 1, 1, 1\}$. The warping paths identified by different colors align the two series at a similar alignment cost. However, warping paths closer to the diagonal of the cost matrix, such as paths indicated by light blue color, often minimize possible shape distortions caused by DTW warping.

results in a significant increase in the dimension of the warped series. This dimensional increase in the worst case can raise to $M + N$, i.e., for an $(M \times N)$ global cost matrix. Furthermore, such warping paths can also introduce significant shape distortions in at least two possible scenarios. In the first case, a warping path can sequentially includes global cost matrix cells that are located at either $(i, j+k)$ or $(i+k, j)$ for $k = \{0,\ 1,\ 2,\ \ldots,\ M\ (N)\}$. When this is the case, due to the repeated inclusion of a single coordinate value of one of the two warped series, at least one of them will have a segment that has a constant (*"flat"*) shape. In another scenario, a DTW warping path could have such constant entries followed by a sudden change in direction that is immediately followed by consecutive constant entries of the form $\{(i + k, j)\}$ or $\{(i, j + k)\}$, i.e., for $k = \{0,\ 1,\ 2,\ \ldots\ M\ (N)\}$. When this is the case, there will be a sudden vertical line with a "pinching" effect on at least one of the warped series. In DTW based averaging heuristics, such kinds of shape distortions are known as shape distortion due to *Pathological* associations. In this regard, some DTW based averaging techniques propose to customize the original version. In this aspect, the most common proposal is to devise a mechanism that encourages the selection of warping paths that are relatively close to the *"diagonal"* of the global cost matrix [41], [42]. Thus, they harvest the advantages of DTW while maintaining shape distortions introduced due to DTW warping to an acceptable level.

With this technicality in mind, for our example series, we have manually selected a warping path that is close to the *"diagonal"* of the global cost matrix, i.e., along the light blue color shown in Figure 2.1. With this selection, the two time series gets warped from $\{\ X = \{1, 1, 5, 5, 5, 1, 1\},\ Y = \{1, 5, 5, 5, 5, 1, 1, 1\}\ \}$ to $\{\ X = \{1, 1, 5, 5, 5, 5, 1, 1, 1\},\ Y = \{1, 1, 5, 5, 5, 5, 1, 1, 1\}\ \}$. In other words, DTW transformed the two series from a $\mathbb{R}^7$ and $\mathbb{R}^8$ spaces to a $\mathbb{R}^9$ space. The original and the corresponding transformed series are shown in Figure 2.2. According to Figure 2.2 (b), DTW has identified the two series to be a shifted versions of each and it has aligned them a zero DTW distance. However, even though such alignments could portray DTW as a flawless algorithm, a deeper investigation reveals the contrary. In this aspect, the first limitation that immediately becomes evident is its computational complexity. In this regard, the computation of a $(M \times N)$ global cost matrix requires $M \times N$ calculations. Thus, if we consider the time needed to search for a warping path to be insignificant, we can safely assume DTW has a computational complexity of $O(M \times N)$. Moreover, this computational complexity could significantly

(a) Time series X and Y before DTW warping



(b) Time series X and Y after DTW warping

Figure 2.2: Dynamic time warping of $X = \{1, 1, 5, 5, 5, 1, 1\}$ (red) and $Y = \{1, 5, 5, 5, 5, 1, 1, 1\}$ (blue). The algorithm has identified the two series to be a shifted version of each other. Thus, it has aligned them so that they have a zero euclidean distance.

grow when DTW gets utilized under different setups. For instance, if we aim to utilize DTW while estimating the average of a set that has K members in $\mathbb{R}^M$, we are often at least required to make K DTW warping. Thus, in this case, the computational complexity of DTW grows to $O(K \times M \times M)$. Moreover, for a single increment in the dimensions of the individual series, DTW's computational complexity increases by a factor of $K \times (2 \times M + 1)$.

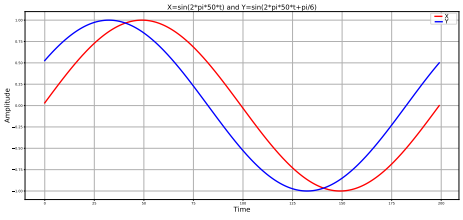
Practically, the computational complexity of DTW is not the only limitation associated with it. Another additional challenge is the presence of a hard min operation in (2.1). This operation makes DTW to be a non-differentiable distance function. Moreover, when DTW gets embedded into objective functions, such as (1.1), it often makes them non-smooth and non-convex [34], [37]. In practice, non-convex objective functions are prone to local minimas that can easily become a major source of non-optimal solutions. Furthermore, the non-smoothness of an objective function prohibits the direct utilization of optimization techniques based on partial derivatives, for instance, the gradient decent. Finally, to make matters further challenging, DTW distance is also known to be a non-metric distance function. In practice, given the time series $\{X,\ Y,\ Z\}\ \epsilon\ \mathbb{R}^M$, we call a given distance function metric if it satisfies the following properties [43]:

- $d(X,\ Y)\ \geq\ 0$ (Property of positiveness)
- $d(X,\ Y)\ =\ 0$ if X=Y (Property of identity of indiscernibles)
- $d(X,\ Y)\ =\ d(Y, X)$ (Property of symmetry)
- $d(X,\ Z)\ \leq\ d(X,\ Y)\ +\ d(Y\ ,\ Z)$ (Property of triangular inequality)

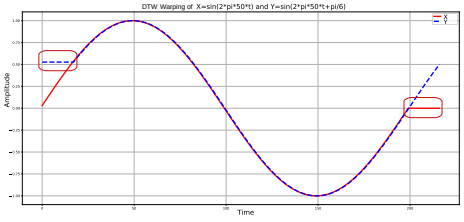
In this aspect, DTW does not meet the properties of triangular inequality and identity of indiscernibles. Thus, this makes DTW warping variant which at times is a problem in some temporal data mining techniques [43]. With such limitations in mind, currently, different variants of DTW are aimed at addressing specific limitations [42], [44]. In the following three sub-sections, we will present three variants of DTW as an example. In reality, the variants presented in the coming subsections focus on addressing the problems associated with pathological associations, non-smoothness, and quadratic computational complexity. Practically, we emphasized on DTW variants focusing on these limitations since they are often mentioned as major limitations in DTW based on averaging heuristics [15], [16].

2.1.1 Weighted Dynamic Time Warping

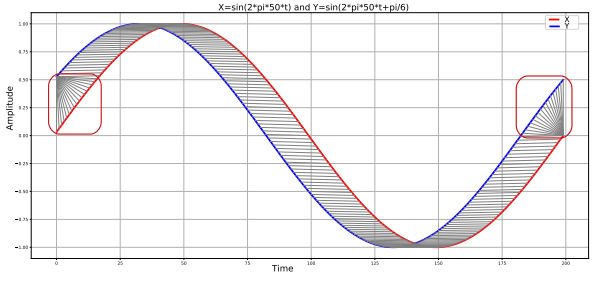
Practically, the basic Dynamic Time Warping (DTW) algorithm does not consider the phase (distance) difference between associated coordinates. To this end, at times, it introduces shape distortions that are mainly related to *pathological* associations [16], [42]. To visually demonstrate this point, we can consider the warping of two sinusoidal signals that are defined as $X = \sin(2 \times \pi \times 50 \times t)$ and $Y = \sin(2 \times \pi \times 50 \times t + \pi/6)$. To extract a discreet time series from these continuous sinusoidal signals, we will use the *Nyquist* criterion and take samples at $f_s \geq 2 \times f_m$. In *Nyquist* criterion, f_s and f_m are respectively the sampling frequency and the maximum frequency component within a sampled signal [45]. With this understanding, we set our sampling frequency to 10,000 Hz, i.e., we take samples every 0.0001 seconds. In reality, this sampling frequency is far greater than the maximum frequency component within the two sinusoids, i.e., 50 Hz. Thus, it will provide a smooth time series representation of the continuous sinusoids which could easily be reconstructed to their continuous form using basic low pass filters [45]. Finally, for our demonstration, we only consider the samples within the single cycle of the sinusoids, i.e., within 0.02 seconds. Overall, Figure 2.3 (a) depicts the time series representations of the sinusoids which have lengths of 200 time stamps. Moreover,



(a) Before DTW warping



(b) After DTW warping

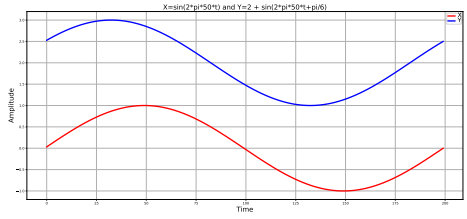


(c) DTW association plot

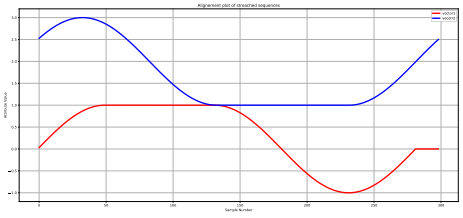
Figure 2.3: Dynamic Time Warping of two time series defined by taking samples from two sinusoids, i.e., $X = sin(2 \times \pi \times 50 \times t)$ (red) and $Y = sin(2 \times \pi \times 50 \times t + \pi/6)$ (blue). The red boxes shown in the DTW coordinate association (c) demonstrate the *pathological* association of one coordinate to multiple coordinates of its counterpart. These textit pathological associations are the underlying reason behind the shape distortions introduced by DTW as shown with the red boxes in (b).

Figure 2.3 (b) demonstrates the DTW warping of the two sinusoids. However, in Figure 2.3 (b), the warping has introduced constant horizontal (*"flat"*) lines that were not evident in the original series. These constant horizontal shapes correspond to the *pathological* associations and they are marked with red boxes in Figure 2.3 (c). In the figure, the *pathological* association to the left happens when $Y's$ first coordinate value pairs with multiple coordinate values of X. Similarly, such associations are also evident when the last coordinate value of X gets paired with multiple coordinate values of Y.

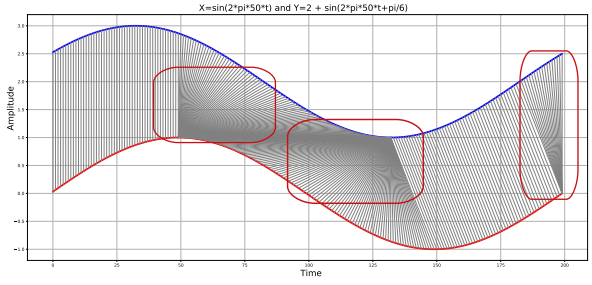
In reality, the shape distortion presented in Figure 2.3 (b) might appear insignificant as compared to the use of DTW. However, in practice, such *pathological* associations could introduce major shape distortion for minor reasons. For instance, if we introduce a minor constant offset to one of the two sinusoids say $Y = 2 + sin(2 \times \pi \times 50 \times t + \pi/6)$, i.e., as shown in Figure 2.4 (a); the shape distortions that were previously evident at the edges of the two warped sinusoids now gets shifted and magnified as shown in Figure 2.4 (b). This happens due to the constant offset that pushes the negative amplitude values of Y above the zero axis. Thus, the basic DTW could not now find a proper match for the non-shifted and shifted negative amplitude values of X and Y. For instance, the shape distortion in X gets introduced since DTW identifies the first positive peak of X as the only optimal match for the shifted negative values of Y. On the contrary, the shifted first negative peak of Y becomes the only suitable match for the negative values of X. Thus, causing the distortion observed in the warped Y. With such observations in mind, the authors in [42] proposed a variant of DTW namely the Weighted Dynamic Time Warping (WDTW). In contrary to the basic DTW, WDTW penalizes DTW associations based on their phase difference. In practice, WDTW defines this penalty using the *Sigmoid* function given in (2.2) where g and W_{max} are hyper-parameters that respectively determine the slope and maximum weight penalty. Moreover, $|i - j|$ is the absolute value of the phase difference between two DTW associated coordinates [42].



(a) Before DTW warping



(b) After DTW warping



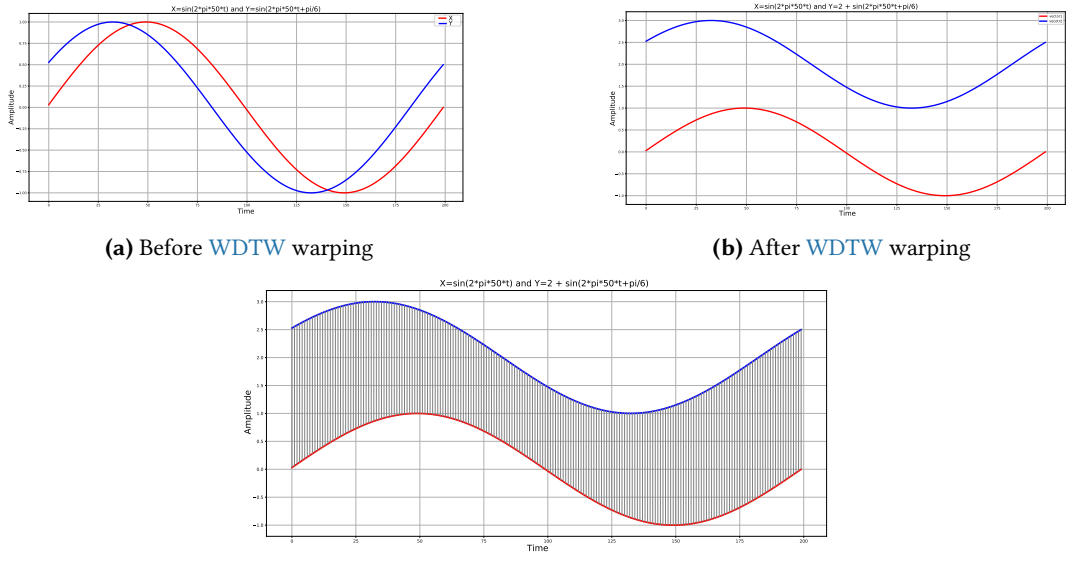
(c) DTW association Plot

Figure 2.4: A demonstration on the effects of a constant amplitude offset on DTW, i.e., $X = sin(2 \times \pi \times 50 \times t)$ (red) and $Y = 2 + sin(2 \times \pi \times 50 \times t + \pi/6)$ (blue). The constant offset on Y forces DTW to *pathologically* associate positive peak of X to offseted negative values of Y and offseted negative peak of Y to negative values of X, i.e,, as shown in (c). These *pathological* associations generates the shape distortions observed in the warped series as shown in (a).

$$W_{\{i,j\}} = \frac{W_{max}}{1 + exp^{-(g \times |i-j|)}} \tag{2.2}$$

In practice, WDTW uses these weight penalties while computing the local cost values, i.e., $d(x_i, y_j) = (w_{\{i,j\}} \times (x_i - y_j)^2)$. Thus, this way, it discourages the association of coordinates that have a higher

phase difference. In other words, WDTW starts encouraging warping paths that are closer to the *"diagonal"* of the global cost matrix. However, in reality, such pushes often result in an alignment that could have a higher alignment cost. Thus, in WDTW, balancing between cost and shape preservation is a hyper-parameter tuning process. In this regard, we can manipulate the slope of the weight penalties by varying g, where a manipulation could result in one of the three possible tuning scenarios. In the first case, we can set g to zero which reduces (2.2) to W_{max}. Thus, in this case, WDTW penalizes each associated coordinate equally irrespective of their phase difference. In other words, it behaves as the basic DTW. However, if we let $0 \leq g \leq 1$, WDTW will start to penalizes associations that have higher phase differences. In other words, WDTW starts to encourage warping paths closer to the diagonal of the global cost matrix. Finally, if we set $g > 1$, WDTW will start discouraging the slightest phase difference. Thus, in this case, it behaves as an euclidean distance. With this said, we will finalize our discussion of WDTW by revisiting the shifted sinusoids shown in Figure 2.4 (a). According to Figure 2.5 (b), WDTW preserved the shapes of the two warped sinusoids. Consequently,



(a) Before WDTW warping

(b) After WDTW warping

(c) WDTW association Plot

Figure 2.5: Weighted Dynamic Time Warping (WDTW) of two time series defined from sinusoides $X = sin(2 \times \pi \times 50 \times t)$ (red) and $Y = 2 + sin(2 \times \pi \times 50 \times t + \pi/6)$ (blue). We have set the maximum and slope of the *Sigmoid* weight penalties as $W_{max} = 4$ and $g = 0.25$. By penalizing DTW associated coordinates based on their phase difference, WDTW was able to identify the two time series to be an offseted and shifted version of each other.

in practice, WDTW was often put to use in shape-based classification tasks [42]. However, in most cases, WDTW was found to increase the alignment cost. In the context of our sinusoidal examples, due to the introduction of the constant offset, the WDTW distance has only increased from zero to two, i.e., the offset value. However, in practice, such ideal alignment is not always evident. To this end, different works have proposed alternatives that could preserve shapes of warped series at a lower alignment cost [27], [41]. However, despite such improvements, the overall warping process remained quadratic and non-smooth. With this in mind, the authors in [37] proposed to address the non-smoothness issue with Soft Dynamic Time Warping (SDTW).

2.1.2 Soft Dynamic Time Warping

Following the trends observed in WDTW, Soft Dynamic Time Warping (SDTW) targeted a specific problem associated with DTW, i.e., DTW's non differentiability using Soft Dynamic Time Warping (SDTW). However, unlike WDTW, SDTW aimed at addressing this issue in the context of a predecessor work, i.e., Global Alignment Kernel (GAK) [32], [46]. Practically, kernels get intensively used in some time series classification algorithms, such as the Support Vector Machine (SVM) [46]–[48]. In practice, given two vectors $\{X,\ Y\}\ \in\ \{\mathbb{R}^N,\ \mathbb{R}^M\}$, SVM often aim to maximize their inner product ($\langle X,\ Y\rangle$) [47]. However, direct inner products of temporal datasets often give sub-optimal results due to the presence of temporal distortion. In some cases, researchers propose to overcome this problem by warping vectors (series) before inner products. However, in most cases, researchers often propose to minimize kernels (functions) to maximize inner products. They propose this approach since it often produces the same outcome with less computational overhead [46]. For instance, if we choose to minimize the kernel $k(X,Y)\ =\ \exp^{-}\{\frac{1}{M}||X-Y||_{l2}\}$, we would indirectly be maximizing the inner product of ($\langle X,\ Y\rangle$). This is because if we want to maximize the kernel, we have to minimize the L2 norm of the two vectors. This, in turn, implies a higher correlation or inner product of the two vectors. In general, in practice, kernels are often chosen since they often have desirable mathematical behavior such as smoothness and convexity. Consequently, they are easy to integrate into optimization setups. With these understanding in mind, the authors of SDTW had previously proposed the Global Alignment Kernel (GAK) that is based on DTW, i.e., K_{GA}^{γ} given in (2.3). In (2.3), A and $\triangle(X,\ Y)$ are an $(M\ \times\ N)$ alignment and DTW local cost matrices. Moreover, $\langle A,\ \triangle(X,\ Y)\rangle$ is the inner product between the two matrices.

$$\begin{aligned} k_{GA}^{\gamma}(X,Y)\ &:=\ \sum_{A\ \epsilon\ A_{M,N}} exp^{-\frac{\langle A,\ \triangle(X,Y)\rangle}{\gamma}} where, \\ DTW\ (X,Y)\ &:=\ \min_{A\ \epsilon\ A_{M,N}}\ \langle A,\ \triangle(X,Y)\rangle \end{aligned} \tag{2.3}$$

In reality, GAK can be minimized under different contexts, for instance, either while evaluating the inner products of two vectors or when aligning two series. In this regard, if we see GAK from the context of temporal alignment, it enables SDTW to see the overall DTW alignment process from a different perspective. In this aspect, in SDTW, a warping path is defined with a $(M\ \times\ N)$ alignment matrix, where $A\ \in\{0,\ 1\}^{M\ \times\ N}$. Consequently, given an alignment matrix $A_{M,N}\ \subset\ \{0,\ 1\}^{M\ \times\ N}$, a cell (i, j) is set to one if a warping path includes cell (i, j) of DTW's global cost matrix ($\triangle(X,\ Y)$). With this definition at hand, [37] argued that the basic DTW is differentiable if the alignment matrix $A_{M,\ N}$ is unique. This is because, given a unique warping path, a small perturbance in one of the aligned series will have a smaller chance of breaking DTW associations. This mainly arises from the fact that neighboring global cost matrix cells will have higher warping costs [37]. However, in practice, a unique warping path is not often evident in DTW. To this end, in order to make DTW differentiable, i.e., irrespective of its warping path uniqueness, [37] proposed to smoothen $\triangle(X,\ Y)$ using soft minimums (2.4); where $\{a_1,\ a_2,\ \ldots,a_n\}\ \in\ \mathbb{R}$.

$$Soft_{Min}^{\gamma}\{a_1,a_2,\ldots,a_n\} == \begin{cases} \min_{i=1}^{n}\{a_i\}, & \text{if } \gamma\ =\ 0 \\ -\gamma \ln \sum_{i=1}^{n} exp^{-\frac{a_i}{\gamma}}, & \text{if } \gamma\ >\ 0. \end{cases} \tag{2.4}$$

Practically, such differentiable functions are desired in most optimization techniques based on partial derivatives [34]. In this aspect, the authors further showed that it is possible to compute the partial derivative of DTW using two approaches. In the first case, [37] showed that it is possible to analytically compute $\nabla_X DTW(X,Y)$ using (2.5), where $\left(\frac{\partial \triangle(X,Y)}{\partial X}\right)^T$ corresponds to the Jacobian of the global cost matrix [37]. Moreover, $\mathbb{E}_\gamma[A]$ is the average alignment matrix under Gibbs distribution $p_\gamma \propto exp^{-\langle A, \frac{\triangle(X,Y)}{\gamma}\rangle}$ which is defined for all alignment matrices $(A_{M,N})$. However, [37] also acknowledged that the computational complexity of $\mathbb{E}_\gamma[A]$ is $O(m^2n^2)$. Thus, they proposed Algorithm 1 as an alternative solution, where $\delta_{i,j},\ r_{i,j},\ e_{i,j}$ correspond to the entries of local cost, global cost and $\mathbb{E}$ matrices.

$$\begin{aligned} \nabla_X DTW_\gamma(X,Y) &= \left(\frac{\partial \triangle(X,Y)}{\partial X}\right)^T \mathbb{E}_\gamma[A] \ \ where, \\ \mathbb{E}_\gamma[A] &:= \frac{1}{K^\gamma_{GA}(X,Y)} \sum_{A \ \epsilon \ A_{M,N}} exp^{-\langle A, \frac{\triangle(X,Y)}{\gamma}\rangle} A \end{aligned} \tag{2.5}$$

Algorithm 1: Backward recursion to compute $\nabla_X DTW(X,Y)$ [37].

Inputs: $(X\ ,\ Y,\ )\ \epsilon\ (\mathbb{R}^N,\ \mathbb{R}^M), \gamma\ >\ 0$ and distance function δ *operating on* $(x_i, y_j)\ \epsilon\ (X,Y)$
$(.,R) = DTW_\gamma(X,Y), \triangle = [\delta(x_i,y_j)]_{i,j}$
$\delta_{i,m+1} = \delta_{n+1,j} = 0,\ i\ \epsilon\ [n],\ j\ \epsilon\ [m]$
$e_{i,m+1} = e_{n+1,j} = 0,\ i\ \epsilon\ [n],\ j\ \epsilon\ [m]$
$r_{i,m+1} = r_{n+1,j} = -\infty,\ i\ \epsilon\ [n],\ j\ \epsilon\ [m]$
$\delta_{n+1,m+1} = 0,\ e_{n+1,m+1} = 1,\ r_{n+1,m+1} = r_{n,m}$
for j=1…m **do**
  for i=1…n **do**
    $a = exp^{\frac{1}{\gamma}(r_{i+1,j} - r_{i,j} - \delta_{i+1,j})}$
    $b = exp^{\frac{1}{\gamma}(r_{1,j+1} - r_{i,j} - \delta_{i,j+1})}$
    $c = exp^{\frac{1}{\gamma}(r_{i+1,j+1} - r_{i,j} - \delta_{i+1,j+1})}$
    $e_{i,j} = e_{i+1,j} \times a + e_{i,j+1} \times b + e_{i+1,j+1} \times c$
  end for
end for
Output: $\nabla_X DTW(X,Y) = \left(\frac{\partial \triangle(X,Y)}{\partial X}\right)^T \mathbb{E}_\gamma[A]$

In practice, the advantage of SDTW is not only limited to a differentiable distance function. In this aspect, [37] argued that a proper selection of a γ value could smooth out the DTW version of (1.1). To better demonstrate this concept, we can rewrite the DTW version of (1.1) as (2.6) [31], where $P = \{p_1, p_2, \dots p_k\}$ is a set of warping paths such that $p_i\ \epsilon\ \mathbb{R}^\tau$. Moreover, $X_j\ \epsilon\ \mathbb{R}^M$ and $\mu\ \epsilon\ \mathbb{R}^N$ are members of the averaged set and an estimated mean, where $M\ \leq\ N$.

$$F(\mu) = \frac{1}{K} \sum_{j=1,\ P}^{K} \delta_P(X_j, \mu) \tag{2.6}$$

With this definition, the Fréchet function becomes a function of two variables, i.e., μ and P, which leaves us with three possible ways of minimizing (2.6). In the first case, we could fix μ and search for a warping path configuration that minimizes (2.6). Alternatively, we could fix the set of warping

paths and search for an optimal μ. Finally, we could simultaneously search for optimal warping path configurations and μ. However, in all cases, (2.6) is considered as non-convex. To better demonstrate why this is the case, we could adopt the approach utilized in [31] and define component functions as (2.7) [31], where R is a warping path configuration in P.

$$F_R(\mu) = \{p_1, p_2, \dots p_k\} \frac{1}{K} \sum_{j=1}^{K} d(X_j, \mu) \tag{2.7}$$

In addition to this, if we assume that after DTW warping we use $L2$ norm or (1.4) as a distance function, we can safely assume $F_R(X, \mu)$ resembles a set of quadratic functions that has a generic form of $F(x) = (X \pm k)^2 \pm C$. With this assumption, if we take three component functions $\{F_{r1}, F_{r2}$ & $F_{r3}\}$ as an example, we can plot each component functions against different values of μ as shown in Figure 2.6. In the figure, the component functions are plotted as parabolic curves. However, in reality, these parabolic curves are higher dimensional bowls. Moreover, in the Fréchet function, we desire to identify a μ that minimizes the overall component functions rather than individual configurations. To this end, we can assume it to be the point-wise minimum of the individual configuration functions shown in Figure 2.6, i.e., using the solid black line. From Figure 2.6, we can see that the Fréchet

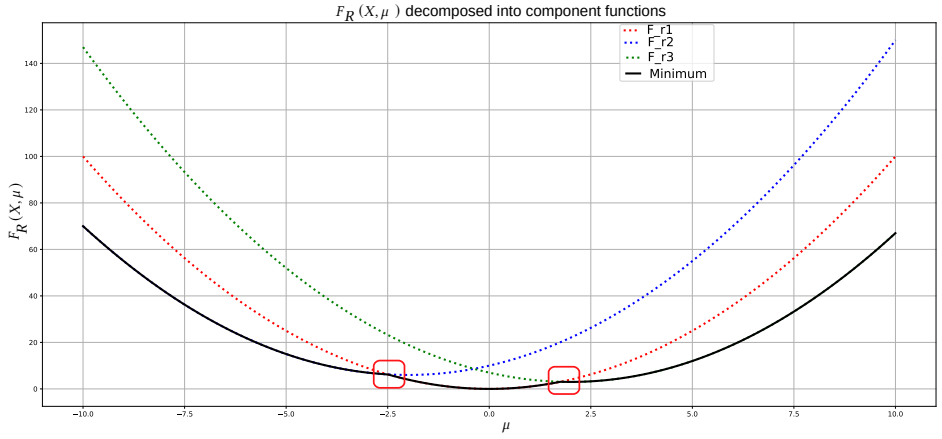


Figure 2.6: The Fréchet function as a point-wise minimum of component functions. In the figure, the Fréchet function, i.e., shown with the solid black curve, is constructed by taking the point-wise minimum of the three component functions F_{r1},F_{r2},F_{r3}. Due to the point-wise minimum operation, the Fréchet function has three minimas and two discontinuities marked by red boxes.

function has two local minimas and discontinuities marked with the red boxes. However, in SDTW, we can control the individual quadratic curves through the γ parameter of the *soft minimums*. To this end, given an appropriate γ value, one can angle a component function such that the points of discontinuities get smoothed out. For instance, in Figure 2.6, if we adjust γ and manipulate the component function F_{r3} such that it approaches F_{r2} from the right, then we can reduce the possibility of getting stuck at discontinuity and local minima located to the right. However, in practice, high γ values could flatten the parabolic curve of the overall component function. Thus, they could sometimes lead to the identification of sub-optimal estimates. To this end, in practice, averaging heuristics that rely on SDTW often treat γ as a hyper-parameter that needs careful tuning [20].

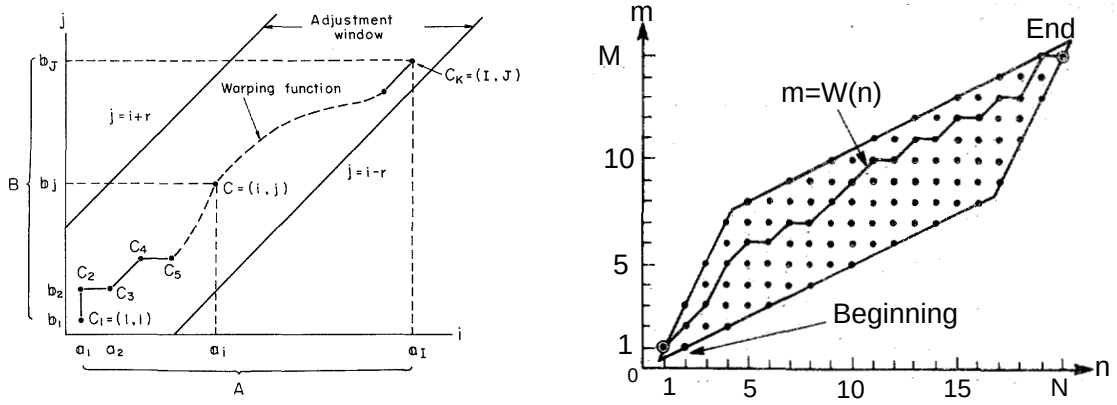
2.1.3 Fast Dynamic Time Warping

Another major limitation which immediately came evident after DTW's introduction was its computational complexity. In earlier times, the computational capabilities of most computing devices were relatively lower than today's computing devices. To this end, earlier alternatives of DTW mainly focused on reducing the computation complexity to at least a linear scale. The first proposal in this regard was the remarks made in the original paper [32]. In [32], the authors proposed to compute global cost values within a constraining window later called the Sakoe-Chuba band [32]. Moreover, the mathematical constraint of the Sakoe-Chuba band was stated as $|i - j| \leq r$, where $i,\ j$ corresponded to the rows and columns of the global cost matrix and r a window size. Thus, since the number of computations is limited to the number of cells, i.e., cells within the constraining window, constrained DTW has a relatively lower computational complexity. However, due to the constraining, a warping path is searched only within the constrained window that could produce sub-optimal warping [41]. Furthermore, mathematically speaking, the sakoe-chuba band gives a relatively loose constraint since r is a hyper-parameter expected to be manually tuned. To address this issue, a later work proposed a relatively well-constrained window known as the Ikatura parallelogram [49]. In reality, the Ikatura parallelogram was mainly proposed to match a reference pattern (R(K)), which is a mathematical model of word sound utterances, to a correlation vector of an input sound. Unlike the sakoe-chuba window, the Ikatura parallelogram placed a tighter constraint on how the n^{th} coordinate values of an input signal get mapped to the m^{th} coordinate value of its reference pattern. In reality, (2.8) formulates a parallelogram in the global cost matrix. Thus, giving raise to the name the Ikatura parallelogram. In general, to give a better visual aid of the two window constraints, we have extracted their graphical representation from their respective original papers, i.e., as shown in Figure 2.8.

$$\begin{cases} m = W(n), & m,\ n \in \mathbb{R}^k,\ \mathbb{R}^N \\ W(0) = 0,\ W(N) = R(K) & \text{Boundary condition} \\ W(n) - W(n-1) = 0,\ 1,\ 2 & \text{if } W(n) \neq W(n-1). \\ W(n) - W(n-1) = 1,\ 2 & \text{if } W(n) == W(n-1). \end{cases} \tag{2.8}$$

However, even though the window constraints significantly reduced the computational requirements, we are still expected to compute the values of the cells within the constrainted window. Thus, in a sense, we are still computing a smaller global cost matrix that has a relatively lower quadratic computational complexity. With this in mind, a relatively recent proposal suggested fast DTW [44]. Fast DTW aimed to linearize the computational complexity of DTW by taking the following three key steps:

- **Coarsening:** This process reduces the dimensions of the warped series by taking the averages of two consecutive time stamps. In fast DTW, this process gets repeated for several iterations where an iteration reduces the dimensions of the warped series by a factor of two.
- **Projection:** At this step, fast DTW first tries to identify a warping path using the coarsened time series. It then projects the estimated warping path to its higher dimensional equivalent.



(a) Sakoe-Chuba band [32] **(b)** Ikatura parallelogram [49]

Figure 2.7: Two proposed window constraints for the global cost matrix of DTW which are used to reduced its computational complexity [32], [49].

For instance, for a global cost matrix that gets reduced by half, a warping path traversing through a cell of the reduced matrix will traverse through two cells of the original cost matrix. In general, similar to the coarsening step, the projection step gets performed iteratively.

- **Refinement:** At this stage, fast DTW refines the warping path that was projected from the lower resolution. To meet this objective, fast DTW runs a constrained DTW only in the neighborhood of the re-projected warping path.

In general, the authors depicted these three key steps through Figure 2.8. Moreover, they also showed that in the worst case, the time complexity of fast DTW is $N \times (8 \times r + 14)$; where N is the dimensions of the warped series. On the other hand, r is the window size utilized for the constrained DTW. In addition to this, they also showed that, the worst case space complexity of fast DTW is $N \times (4 \times r + 7)$. Thus, for a very small window size (r) the time and space complexity of fast DTW can be assumed to be linear ($O(N)$) [44].

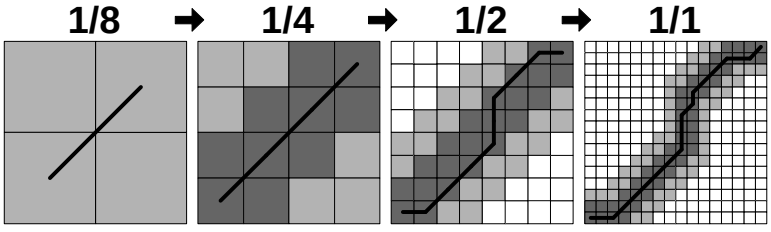


Figure 2.8: The three key steps taken by fast DTW, i.e., coarsening, warping path projection and refinement. These steps has helped fast DTW to linearize the computational complexity of DTW [44]

2.2 Averaging Techniques Based on Dynamic Time Warping

Over the past four decades, a range of well-defined time series averaging techniques has been proposed [14]–[16], [25], [31], [33], [50]. In practice, out of these proposals, more than half of them rely on DTW to align members of the averaged set. For instance, Non Linear Averaging and Alignment

Filter (NLAAF), Prioritized Shape Averaging (PSA),Dynamic Time Warping Barycenter Averaging (DBA),and Stochastic SubGradient (SSG) are some examples. In reality, we can group such DTW based averaging heuristics into two broad categories, i.e., sequential and template based, depending on how members of the averaged series get warped. In this regard, we can categorize NLAAF and PSA as sequential averaging approaches since they propose to warp members of the averaged set sequentially. On the contrary, averaging techniques such as DBA proposed to warp members of the averaged set to a pre-selected template, i.e., making it a template-based approach. With this said, we will next present some of DTW based averaging heuristics which we believe have laid the foundation for time series averaging in DTW space.

2.2.1 Non Linear Averaging and Alignment Filter

The NLAAF was the first temporal averaging heuristic that acknowledged the impact of temporal distortion on the quality of estimated time series averages [14], [16]. To minimize this effect, NLAAF proposed estimating averages by warping members of the averaged set pairwise. Consequently, given an averaged set that has K members in $\mathbb{R}^M$, NLAAF first randomly divide the averaged set into $\frac{K}{2}$ pairs. Following this, NLAAF proposes to align the paired series using DTW and take the arithmetic mean of the warped series as intermediate estimates. These estimates are next grouped into $\frac{K}{4}$ pairs which are also aligned and averaged as in the previous step. In general, NLAAF continues with this iteration until a single estimate remains. To visually demonstrate this process, we consider the Cylinder-Bell-Funnel (CBF) dataset from the University of California Univariate Time Series Repository (UCR)) as an example [2]. This dataset contains time series representing three geometric shapes (classes), i.e., cylinders, bells, and funnels. Figure 2.9 shows the steps followed by NLAAF while estimating an average for the Funnels class. The class contains 8 temporal datasets that have 128 time stamps. In addition to demonstrating the estimation process, we have also compared the estimations of NLAAF with its arithmetic counterpart, i.e., as shown in Figure 2.10. According to Figure 2.10, the

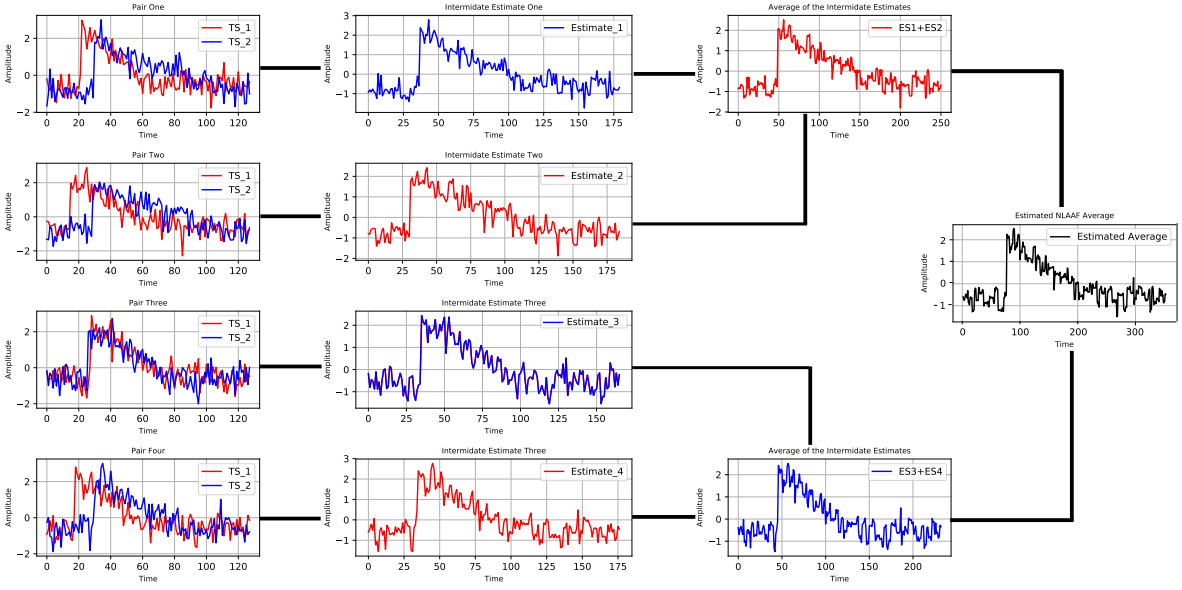
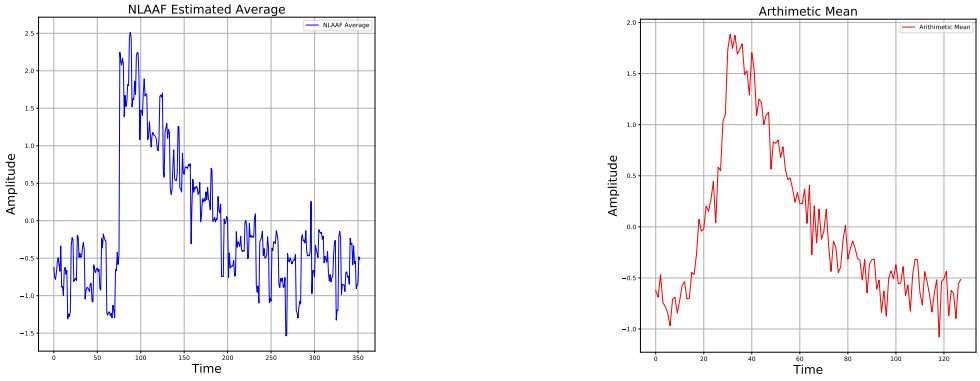


Figure 2.9: A demonstration of NLAAF using the Funnel class of the UCR archive's CBF dataset

NLAAF estimate has preserved the sharp edge observed in the Funnel class. However, if we access the quality of the arithmetic and NLAAF estimates in terms of their WGSS or (1.2), they respectively obtained an average WGSS of 3.2901 and 3.7404. In reality, there are different contributing factors behind this better performance of arithmetic mean that displays a significant shape distortion. In this aspect, the first reason that quickly become evident is the dimension (length) of the estimated averages. In this regard, the dimension of the NLAAF estimate has grown from 128 to 351. This dimensional growth is almost twice the dimension of the arithmetic mean. Consequently, it is logical that the NLAAF estimate shows a slight increase in terms of WGSS. In reality, such dimensional growth of NLAAF estimated averages could quickly become out of hand [15], [16]. This is because NLAAF



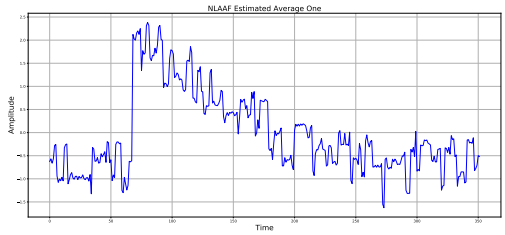
(a) NLAAF mean **(b)** Arithmetic mean

Figure 2.10: Visual comparison for NLAAF estimated (a) and an arithmetic mean (b) for the Funnel class of the UCR archive's CBF datasets. The arithmetic mean has failed to preserve the per-class features due to temporal distortion.

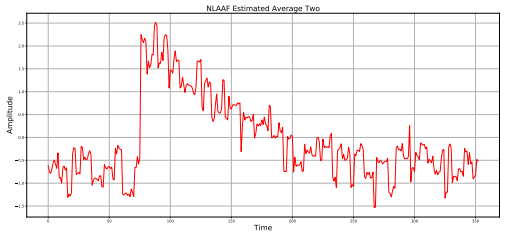
takes the average of each DTW associated coordinates while estimating the averages. To this end, in the worst case, the dimension of NLAAF estimate could grow up to $2 \times K \times M$. In addition to this limitation, NLAAF also assumes an averaged set has an even number of time series. Consequently, it could either leave out one of the averaged series or warp it to an intermediate estimate. However, both approaches could significantly impact the quality of the estimated average [15]. Finally, in NLAAF, the quality of an estimated average depends on the way pairs are selected. This is better demonstrated in Figures 2.11 (a) & 2.11 (b), where the estimates respectively correspond to time series parings of $\{(0,\ 2)\ ,(1,\ 4)\ ,(5,\ 6)\ ,\ (3,\ 7)\}$ and $\{(0,\ 6),\ (1,\ 7),\ (5,\ 4),\ (3,\ 2)\}$. In reality, the two estimations respectively have a WGSS of 3.7407 and 3.7717. To address this particular issue, the authors in [15] proposed the Prioritized Shape Averaging (PSA).

2.2.2 Prioritized Shape Averaging

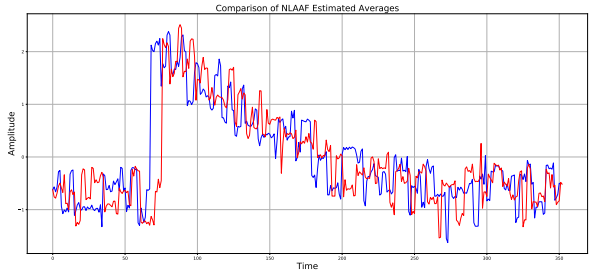
The Prioritized Shape Averaging (PSA) proposed to utilize *agglomerative* clustering in order to minimize the effects of pair selection on the quality of the estimated averages. In this regard, PSA initially identifies the two most similar series to generate the first intermediate estimate. Furthermore, PSA also assigns weights to the clustered series and their respective estimates. In this regard, any series joining a cluster is assigned a weight of one. On the contrary, intermediate estimates that are generated from a cluster containing K members are assigned a weight factor of K [51]. In addition to



(a) Averages estimated from $\{(0, 2), (1, 4), (5, 6), (3, 7)\}$.



(b) Averages estimated from $\{(0, 6), (1, 7), (5, 4), (3, 2)\}$



(c) Visual comparison NLAAF estimates based on the parings given in (b) & (b)

Figure 2.11: Discrepancy among NLAAF estimates due to the difference in pair selection. In addition to the visual difference, the estimate shown in (a) and (b) respectively have an alignment costs of 3.7407 and 3.7717.

this, in order to reflect on these weight factors, PSA also proposed to utilize a variant of DTW that could incorporate the weighting factors into the warping process [15], i.e., the Scaled Dynamic Time Warping (Scaled DTW). In PSA, given two time series $\{X, Y\} \in \{\mathbb{R}^N, \mathbb{R}^M\}$, their respective weights $\{\lambda_1, \lambda_2\}$ and a warping path $p \in \mathbb{R}^\tau$: the coordinate values of an intermediate (final) average are computed using (2.9).

$$z_i = \sum_{i=1,p}^{\tau} \frac{\lambda_1 x_i + \lambda_2 y_i}{\lambda_1 + \lambda_2} \tag{2.9}$$

In general, after computing an initial intermediate estimate, PSA next tries to identify the two most similar series to series or series to intermediate estimate pairs. These pairs are then warped with their appropriate weight factors to generate a new intermediate estimated average and possibly a new cluster. PSA iteratively continues with such hierarchical (agglomerative) cluster formations until a final estimated average remains. With this technicality in mind, we have revisited the Funnel class of the CBF dataset that was introduced in Figure 2.9 and computed the class average using PSA as shown in Figure 2.12.

According to Figure 2.12, the Funnel time series located at the indices of {0, 1, 2, 3, 4} formulate the first three clusters, i.e., $\{(0, 3), (2, 4), (1, 6)\}$. Moreover, from these clusters, the first three intermediate estimations got generated. Following this, the intermediate estimates got assigned a weight factor of two. With these at hand, the series located at the 5^{th} index of the original Funnel datasets and the intermediate estimate generated from the pair $(0, 3)$ were identified as the two most similar series. To this end, the two series were aligned with Scaled DTW and formulated a new cluster and an intermediate estimate with a weight factor of three. Furthermore, this intermediate estimate was close to the intermediate estimate obtained from $(1, 6)$. Consequently, it was later aligned to its new match to generate another intermediate estimate with a weight factor of five. Similar to its

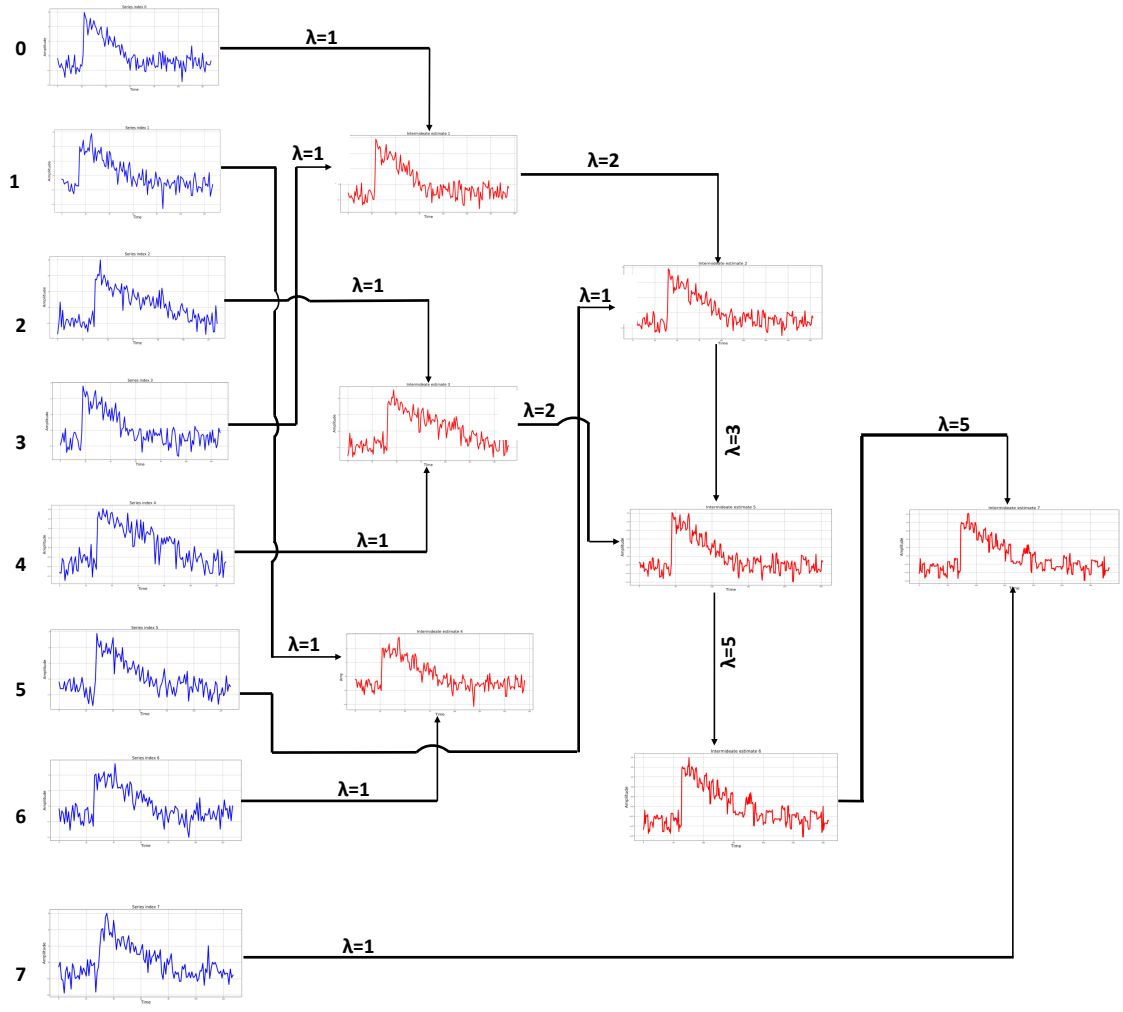


Figure 2.12: A demonstration of PSA using the Funnel class of the UCR archive's CBF dataset

predecessor, this estimate also got matched with the intermediate estimate obtained from the cluster containing time series $(2, 4)$. Thus, raising the weighting factor of the large cluster to seven. Finally, the time series located at the 7^{th} index of the Funnel class joins the bigger cluster to generate the final estimate. With this final estimate, PSA was able to obtain a WGSS of 4.1363. This is a bit higher as compared to NLAAF's 3.7407 WGSS score. However, in practice, such superior performance of NLAAF was not often evident [15], [16]. Moreover, it should also be noted that, the PSA's final estimated has a dimension of 353 that is significantly larger than the dimension of the NLAAF's estimate, i.e., 251. Thus, it logical that the PSA's estimate has a slightly higher WGSS cost. In conclusion, PSA has not also accounted for the increase in the dimension of its estimates. To this end, the authors in [16] proposed Dynamic Time Warping Barycenter Averaging (DBA) as a way out.

2.2.3 Dynamic Time Warping Barycenter Averaging

Dynamic Time Warping Barycenter Averaging (DBA) for the first time avoided approaching time series averaging through sequential warping. On the contrary, it proposed to approach time series averaging as a multiple alignment problem [16]. In this aspect, DBA for the first time associated time

series averaging to the multiple alignment problem well known in the Steiner theory of biological computation [16]. Through this association, [16] acknowledges that time series (sequences) are best summarized (averaged) through simultaneous (multiple) alignment or by minimizing (1.2) [16], [23]. However, in the context of DTW, this is practically intractable for at least three reasons. First, if we desire to align the K series simultaneously, we will be required to define and store a global cost matrix different from a two dimensional array. In reality, the memory requirement of such a matrix could easily come out of hand as the dimension of the warped series increase. Moreover, even if we somehow construct and manage such a matrix, it is not clear how to search for a warping path. Finally, even if we can find a way, the computational complexity of a single iteration would be significantly large. To this end, time series averaging through multiple DTW warping gets identified as one of the NP hard problems [16], [50].

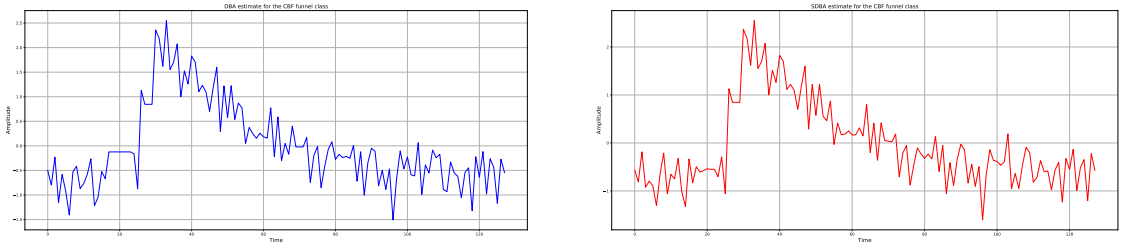
With these understandings, the authors in [16] proposed to mimic multiple alignments by registering the averaged set to their warped space arithmetic mean. However, in the time domain, the warped arithmetic mean average gets represented by a template with the same dimension as members of the averaged set. Furthermore, the authors proposed either to randomly initialize the template or to use one of the averaged series. In the context of the estimation qualities, the authors identified the latter initialization provided better results [16], [19]. In general, in DBA, a template is first initialized using one of the initialization techniques. Following this, all the averaged series are aligned to the template using DTW. Moreover, in the alignment process, DBA keeps the records of DTW associated coordinates. This is because DBA aims to take the *barycenter* of the associated coordinates while generating intermediate estimations. In other words, given a set of DTW associated coordinates $S = \{x_{i1},\ x_{i2},\ x_{i3},\ \ldots,\ x_{in}\}$, the *barycenter* of associated coordinates (z_i) is computed using (2.10).

$$z_i = \frac{z_i + x_{i1} \ + \ x_{i2} + \ x_{i3} \ + \ \ldots \ + x_{in}}{n+1} \tag{2.10}$$

The *barycenter* averaging ensures that the dimension of the final estimate is equivalent to the dimension of the averaged series. Thus, we can alternatively assume the *barycenter* averaging as a re-transformation technique, i.e., re-transforming the estimated average to its time domain representation. In general, DBA is often iterated over the warping and *barycenter* averaging steps until (1.1) falls below a pre-selected tolerance value or a final iteration count is reached.

Even though the *barycenter* averaging approach enabled DBA to avoid an ever-increasing dimension of estimates, it also constrained the estimates to be in $\mathbb{R}^M$ where M is the dimension of the averaged series. In this context, a latter work showed that an estimate in $\mathbb{R}^\tau$ has an equivalent in $\mathbb{R}^M$ where $M \ < \ \tau$ [52]. In other words, they demonstrated how an estimate in $\mathbb{R}^\tau$ cloud get reduced to an estimate in $\mathbb{R}^M$, i.e., without a significant loss of quality. However, there were still some limitations that became evident through time. For instance, DBA still inherited the non-smooth and non-convex objective function (the Fréchet function). Thus, it was relatively difficult for DBA to utilize classical optimization techniques such as gradient descent. In this regard, a relatively recent work proposed a sub-gradient optimization approach that aimed to utilize gradient descent on the individual curves of the component functions shown in Figure 2.6 [31]. On the other hand, the introduction of soft DTW

has helped with the introduction of a better performing differentiable DBA variant, i.e.,SDBA [20], [53]. In general, even under the mentioned limitations, DBA and its variant proved to be the best performing DTW based time series averaging techniques. However, their computational complexity and non-smoothness inhibit them from utilizing the powers of modern-day optimization setups such as neural networks. To this end, in recent years, researchers have started to shift their focus toward warping techniques that can easily get integrated into neural networks. However, before we proceed to this discussion, we would like to finalize the discussion of DBA by presenting DBA's estimate for the CBF Funnel class. In this regard, we have computed the DBA and SDBA estimates of the dataset as in Figure 2.13. To generate the estimates, we have executed both algorithms for 100 iterations while SDBA's SDTW γ value was set to 0.01. With these estimations, DBA and SDBA respectively scored a WGSS of 3.5484 and 3.4530 which are better than NLAAF's 3.7407 and PSA's 4.1363. In conclusion, we would like to point out that averaging techniques based on DTW are not limited to the three algorithms presented so far. On the contrary, in recent years, different types of DTW based averaging heuristics have been proposed. For instance, in [31], a subgradient version of DBA was proposed in order to overcome the non smoothness of the Fréchet function. On the other hand, in [50], a compression algorithm for constrained averages was proposed. Overall, we find the three algorithms presented in this section to convey the general concepts behind DTW based averaging techniques.



(a) DBA estimate **(b)** SDBA estimate

Figure 2.13: A demonstration of DBA and SDBA using the Funnel class of the UCR archive's CBF dataset

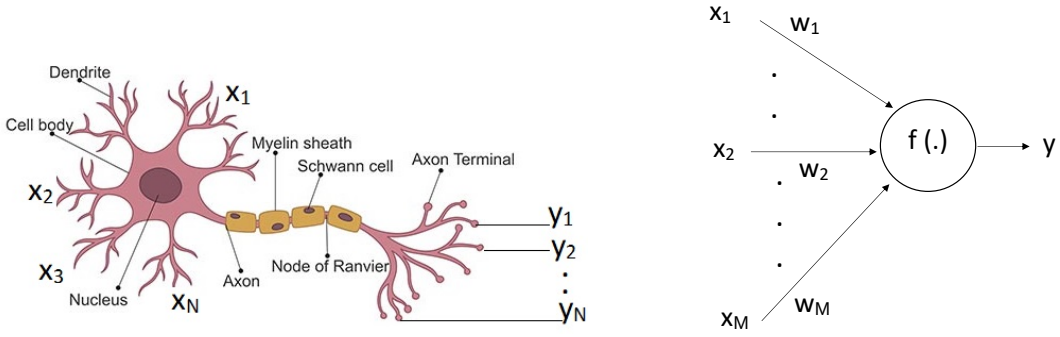
2.3 Deep Neural Networks and Time series Averaging

In earlier days, neural networks were often associated with the realization of basic logical operations. However, as time progressed, researchers realized that neural networks were capable of optimizing various objective functions with a proper modeling [29], [54]. The first neural network component proposed in this regard was the neuron. The neuron initially assumed to take an input of the form $X = \{x_1, x_2, \ldots, x_M\}$ and generates an output using (2.11), where w, b and f are the weights, bias and the activation function of a neuron.

$$y = \sum_{i=1}^{M} f(w_i x_i) + b \tag{2.11}$$

Implementation-wise, various neuron models were initially proposed with different objectives in mind. In this regard, the *McCulloch Pits*, the *Perceptron* and the *ADAptive LINear (ADALIN)* were some of the

well known early neuron models [54]. In practice, such neuron models were often represented with the block diagram shown in Figure 2.14 (b). Moreover, in most literature, Figure 2.14 (a) is often presented to show the resemblance of the neuron model to its natural counterpart. In this aspect, we can take the *dendrite* and *axon terminal* of a natural neuron to correspond to the input and output of the neuron model. However, despite this generalized representation, different neuron models followed different



(a) A natural neuron

(b) Model of a neuron.

Figure 2.14: Similarities among a natural neuron and its model in neural networks

approaches to meet underlying objectives. For instance, the *McCulloch Pits* model was proposed to realize the functionalities of digital logic gates, i.e., *AND*, *OR*, *NOT*. In order to meet this objective, the *McCulloch Pits* proposed to utilize *Binary (B)*, *Ramp (R)* and *Sigmoid (S)* activation functions that are mathematically modeled using (2.12) [54]. However, even though the *McCulloch Pits* was able to model the functionalities of digital logic gates, the neurons were not trainable. Consequently, the weights of *McCulloch Pits* get set manually. With this in mind, Frank Rosenblatt proposed the *Perceptron* neuron model which updates its weights using $\eta\ (y - y_p)\ w_i$, where η, y and y_p were the learning rate, a true value and predicted output. Following the same trend, the *ADALIN* also proposed to update the weights of a neuron similarly. However, unlike *McCulloch Pits*, it was mainly activated using a linear activation function ($f(x) = x$). Nevertheless, researchers quickly realized a single neuron is not sufficient to model complex objective functions. To this end, they proposed to organize neurons using layers to meet the demands of inputs and target objective functions.

$$B(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } m \geq 0 \end{cases} \qquad R(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } 0 \leq x \leq \beta \\ 1, & \text{if } x > \beta \end{cases} \qquad S(x) = \frac{1}{1 + exp^{-x}} \tag{2.12}$$

2.3.1 Neural Network Layers

Today, there are different types of neural network layers that vary depending on how they extract features and analyze inputs. In practice, researchers often rely on this aspect and organize layers under a suitable architecture to manage underlying objective functions. In this subsection, we will give a brief review on three types of layers, i.e., the *Dense*, *Convolutional* and Long Short Term Memory (LSTM) cells. Practically speaking, each layer has its advantages and disadvantages. For instance, a *Dense* layer is known to be capable of learning global features [7], [29]. However, if we expect to

identify features irrespective of their location, a *Convolutional* layer is often preferred [7]. With this in mind, we will further our discussion with the *Dense* layer since it is historically the first to be proposed [29], [54].

2.3.1.1 Dense Layers

In neural networks, a *Dense* layer often assume its inputs are univariate M dimensional vectors. Thus, given an input of the form $X = \{x_1, x_2, \ldots, x_M\}$, a *Dense* layer is built from N neurons that are either connected to each and every values of the inputs (x_i) or to some of them [29], [54]. In practice, if the former approach gets used, a *Dense* layer is said to be in a fully connected configuration (fully connected *Dense* layer). Figure 2.15 demonstrates this configuration of a *Dense* layer where each outputs (y_i) are computed using (2.11). In practice, a fully connected *Dense* layer is considered good

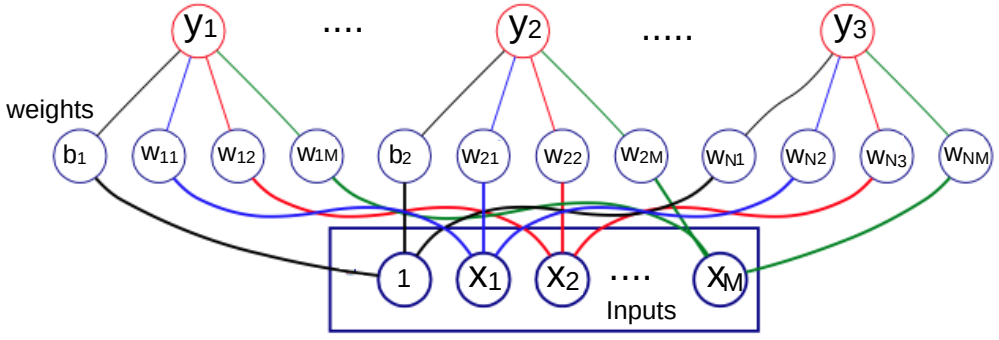


Figure 2.15: A demonstration of a fully connected *Dense* layer [29]

at learning global features [29]. In reality, in *Dense* layers, a neuron is aware of all the possible input values. However, in practice, such full connections often become computationally demanding. In this aspect, for a *Dense* layer with N neurons that has input in $\mathbb{R}^M$, we have $M + 1$ trainable weights associated with each neuron. This requirement becomes more demanding as the number of nodes (neurons) and *Dense* layers increase. Besides this intensive computational requirement, fully connected *Dense* layers are often prone to over-fitting. In neural networks, we say an over-fitting has occurred when its performance is better on training inputs rather than on validation (unseen) datasets. One underlying reason behind such behaviors of neural networks is the possibility of learning complex functions for simple input and objective functions. In this aspect, a large number of trainable weights, i.e., as in the case of fully connected *Dense* layers, are known to be one contributing factor. For instance, if we assume l stack of neurons modeled with (2.11) have linear activation functions ($f(\sum_i^M w_i\ x_i) = \sum_i^M w_i\ x_i$), we can think of them as a trainable polynomial function of degree $> l$. Moreover, in this perspective, $x_i \in X$ becomes the coefficient of the trainable parameter $w_i \in W$. At this point, if we also assume we are trying to learn an optimal regression line for input in $\mathbb{R}^2$, a stack of neurons that have 3 connections are capable of learning at least a quadratic polynomial that could be a perfect fit for a training input in $\mathbb{R}^2$. However, if we add more neurons and stack them as a fully connected *Dense* layer, the network will learn a complex function for a simple objective. Hence, when this is the case, it will often be difficult for a neural network to generalize well for most unseen datasets. On the contrary, the network will likely train to perfectly fit the most abundant and

relatively easy training inputs. To this end, in some practical cases, it is often proposed to randomly drop out some of the *Dense* layer connections to keep the computational requirement and over-fitting problem at an acceptable level [29].

However, despite such modifications, there are times when a *Dense* layer is not an optimal choice for processing some inputs. In this regard, a simple example would be when the inputs are two dimensional images. In such cases, a *Dense* layer expects the inputs to get flattened into a one dimensional column vector. However, such flattening operations often lead to the destruction of spatial information that could, in turn, affect the performance of a neural network. With this understanding, researchers have proposed a range of layers with different types of inputs in mind. For instance, researchers have proposed *Convolutional* layers that are known to perform better on image and shape analysis.

2.3.1.2 Convolutional Layers

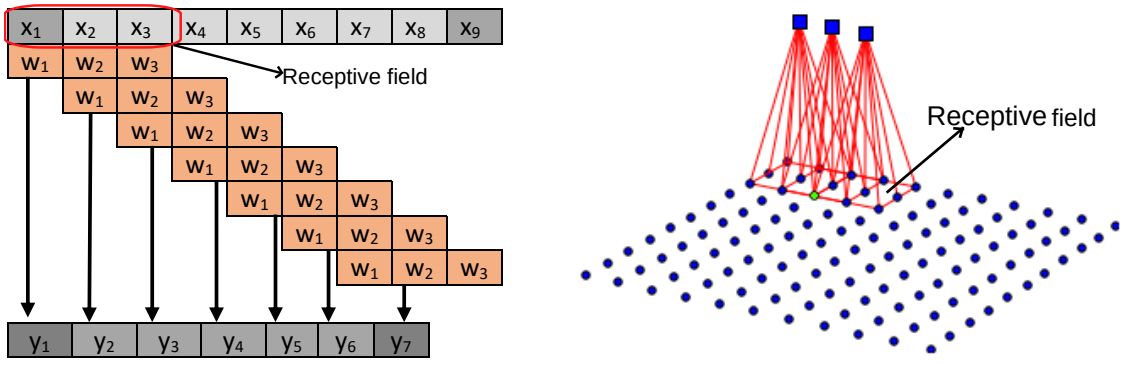
In neural network based optimization setups, sometimes we desire to identify descriptive features irrespective of their location [7], [55], [56]. For instance, if we propose to use a neural network in a face recognition system, then we often desire the network to have the ability to identify basic features on a human face, i.e., eyes, eyebrows, nose, mouth, etc. Furthermore, we expect the network to identify such descriptive features irrespective of the location of a human face in a given image. However, if we design this network using a set of *Dense* layers, we should at least expect to face the two difficulties. First, a *Dense* layer expects one-dimensional inputs. Thus, it requires input images to get flattened. However, the flattening of input images will deform the spatial information of the features we aim to extract. Secondly, since a *Dense* layer's neuron gets connected to every input value, it will often mix and process irrelevant information, for instance, a blank space within an image. With such observations in mind, researchers proposed to process such kinds of inputs with *Convolutional* layers.

Unlike its predecessor, a *Convolutional* layer does not process its input all at once. Instead, a *Convolutional* layer introduced two parameters that are sufficient for its basic operation, i.e., a *Convolutional kernel* and *stride* [7], [29], [56]. On one hand, the *kernel* of a *Convolutional* layer defines the number of trainable weights a *Convolutional* neuron is expected to have. On the other hand, a *Convolution* layer uses the *stride* to define the steps taken by a *kernel* while sliding along the axes of the input. These functionalities are better demonstrated in Figures 2.16 (a) & (b) which correspond to the one and two dimensional *Convolution* operation. For instance, in Figure 2.16 (a), given an input $X \in \mathbb{R}^M$, each output values (y_i) are computed using (2.13) where K is the size of the *Convolutional kernel*, $0 \leq j \leq K$ and $1 \leq i \leq M - K$.

$$y_i = f(\sum_{i,j=0}^{K} w_i x_{i+j} + b) \tag{2.13}$$

On the contrary, the two dimensional *Convolutional kernel* of $M \times N$ shown in Figure 2.16 (b) slides along the horizontal and vertical axes while an output is computed using (2.14). In general, despite these differences, in both cases, an area that excites a neuron at a given time is known as the receptive field of a *Convolutional* layer. In Figure 2.16, the receptive fields of the *Convolutional kernels* are marked using red boxes. In practice, receptive fields play a significant role in the features a given*Convolutional* layer extracts. To this end, in most practical cases, the receptive fields of layers

$$y_i = f(\sum_{i}^{M} \sum_{j}^{N} x_{i,j}\, w_{i,j}) \tag{2.14}$$



(a) One dimensional *Convolution* with single *stride*.

(b) Two dimensional *Convolution* with single *stride* [29].

Figure 2.16: A Demonstration of a one and two dimensional *Convolutional* layers

are systematically tuned for better performance. For instance, instead of directly increasing the *kernel* size, the effective receptive field is often increased by stacking *Convolutional* layers [57], [58]. In reality, this has at least two advantages. First, when *Convolutional* layers are stacked, consecutive layers get the chance to work on a more refined input. Secondly, due to the stacking, it would be possible to deploy larger *kernels* with smaller ones with relatively lower computational demand. Moreover, with smaller *kernels* we are in sense enabling a larger receptive field to focus on smaller details.

Generally, we can safely correlate the operations of a *Convolutional* layer to concepts in discrete linear time invariant (LTI) systems. In practice, input/output relationship for such systems gets governed by the convolution of their impulse response h and an input signal. In other words, given an input $X \in \mathbb{R}^M$ and an impulse response $h \in \mathbb{R}^K$, the output of a discrete LTI system is computed using (2.15) [45].

$$y_i = \sum_{i,j=0}^{K} h_j x_{i-j} \tag{2.15}$$

However, unlike the convolution in discrete LTI systems, a neural network's *Convolutional* layers have a trainable impulse response. Moreover, neural network's *Convolutional* layers could utilize a non linear activation function, i.e., after the convolution operation. In addition to this, a *Convolutional* layers have dynamic capabilities that can either be introduced through additional parameters or by

manipulating existing ones. For instance, if we set the *stride* (S) to be greater than one, a *Convolutional* layer can reduce the dimension of its input by at most a factor of S. In practice, such capabilities are useful when we desire to reduce the dimension of an input in a more intelligent manner [59]. On the contrary, we can also use *Convolutional* layers in their transposed form to perform an intelligent up-sampling. However, in this case, we are expected to slide each input value (x_i) along a *Convolutional kernel* that has a size of K. Thus, this way, it up-samples the input by a factor of K. Furthermore, if we do not desire to either upscale or reduce an input's dimension, we can also introduce a *Padding* parameter. In addition to these possibilities, a *Convolutional* layer is also capable of learning multiple *kernels* at once, where a collection of multiple *kernels* are often called *filters* [29]. In practice, the outputs of each *kernels* gets organized into channels where a channel often identifies a certain unique feature of an input. In general, a *Convolutional* layer is capable of learning multiple *kernels* whose output dimensions are governed by (2.16), where $D_x,\ D_y$ are the dimensions of the input and the output. Moreover, $P,\ K,\ S$ are the *padding*, *kernel*, and *stride* size of the *Convolutional* layer.

$$D_y = \lfloor \frac{D_x + 2 \times P - K}{S} \rfloor + 1 \tag{2.16}$$

With these dynamic capabilities, *Convolutional* layers have intensively get utilized in most renowned neural network architectures. Typical examples in this regard are, the Visual Geometric Group 16 (VGG16), the Residual Network (ResNet)), the Inception, InceptionTime, etc [57], [58], [60], [61]. However, in reality, there are additional reasons behind the large deployment of *Convolutional* layers. First, *Convolutional* layers have a smaller number of trainable weights ($N \times (K + 1)$) where N and K are the number and size of a *Convolutional kernel*. To this end, they often require fewer computational resources compared to their counterparts. Moreover, in *Convolutional* layers, it is possible to target specific features of an input by varying their receptive field (*kernel*) size. With these observations in mind, we construct the main blocks of our proposals using *Convolutional* layers and utilize *Dense* layers to terminate network modules. However, when we discuss the practical aspect of our proposals, i.e., in chapter four, one of our works utilizes LSTM cells. Thus, to further aid this discussion, we will finalize this subsection by presenting the Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) cells.

2.3.1.3 Layers in Recurrent and Long Short Term Memory Neural Networks

In machine learning, different types of inputs place different sets of requirements on neural networks. For instance, if we aim to utilize neural networks to process inputs that follow a Markovian chain behavior, then we expect the deployed network to have a memory [29], [62]. For instance, if we consider word predictors commonly found in renowned search engines such as *Google*: we expect users to enter a part of a sentence and to get presented with options that could fill their sentence in a meaningful manner. Thus, in such cases, we expect an underlying neural network to be aware of contexts which in turn requires remembering a range of predecessor words. To practically address such requirements, researchers initially proposed Recurrent Neural Network (RNN) which were later upgraded to the Long Short Term Memory (LSTM) networks [29]. In RNN, layers sequentially process their inputs by introducing the concept of states [29]. In this regard, given an input that has a form

$X = \{x_1, x_2, x_3, \ldots, x_M\}$, a RNN layer computes the output at timestamp i using two steps. In the first step, it takes the input at x_i and scales it with W_{input}. Following this scaling, it combines the scaled input with the weighted version of a previous *state* (s_{i-1}) using (2.17), where f_{state} corresponds to the activation function of a state. Finally, as a final and second step, a RNN layer computes the output of the current time stamp i using (2.18).

$$s_i = f_{state}(W_{input} \times x_i + W_{state} \times s_{i-1}) \tag{2.17}$$

$$y_i = f_{out}(W_{output} \times s_i) \tag{2.18}$$

In most literature, this time recursion of an RNN layer is visually interpreted as shown in Figure 2.17 [29], [62]. According to Figure 2.17 [29], [62], the weights W_{input} , W_{state} and W_{output} of a RNN layer are shared across the time stamps. To this end, in RNN, we could face two extreme cases as the layer propagates through time. In the first case, the magnitude of the weights could significantly increase. Thus, when the partial derivative of errors with respective layer weights gets taken, they could easily give an exploding gradient. On the contrary, if the magnitude of the weights is much smaller, they could easily give vanishing gradients [29]. However, in practice, we rely on gradients to update layer weights through back-propagation. With this problem in mind, Hochreiter and Schmidhuber proposed the LSTM [63].

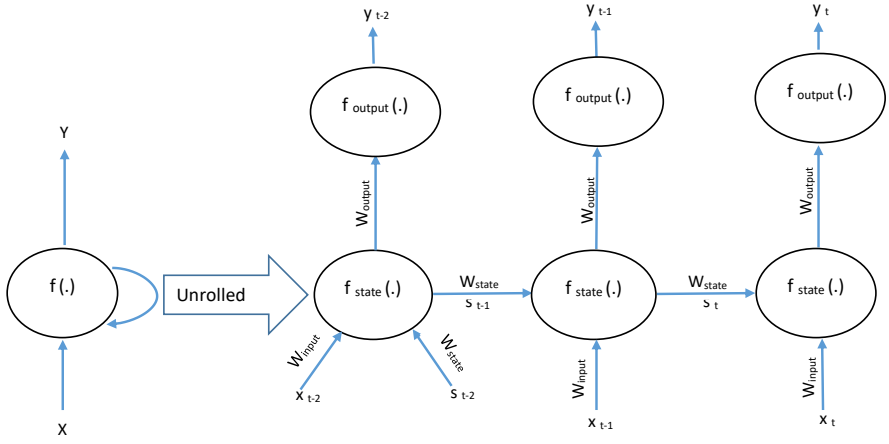


Figure 2.17: The unrolling of a Recurrent Neural Network (RNN) layer

In LSTM network, layers (cells) introduced a third parameter called cell memory state (c_t) that controls how much of the state information propagates to the current output. In this regard, Figure 2.18 depicts how this memory control is achieved in a LSTM cell. In this aspect, internally LSTM cells utilize two sets of activation functions, i.e., *Sigmoid (σ)* and *hyperbolic tangent (tanh)*. In reality, each σ or tanh activation function gets deployed using fully connected *Dense* layers, where the number of neurons depends on the embedding utilized for each coordinate (x_i) of an input. In general, a LSTM cell distribute the *tanh* and *Sigmoid* activation functions among three gates, i.e., the *input*, *output*, and *forget* gates[29]. In Figure 2.18, the left-most *Sigmoid* activation defines the *forget* gate. In reality, a *Sigmoid* activation is within the range of $[0, 1]$. Consequently, it can define how much of the previous cell memory gets passed to the current cell.

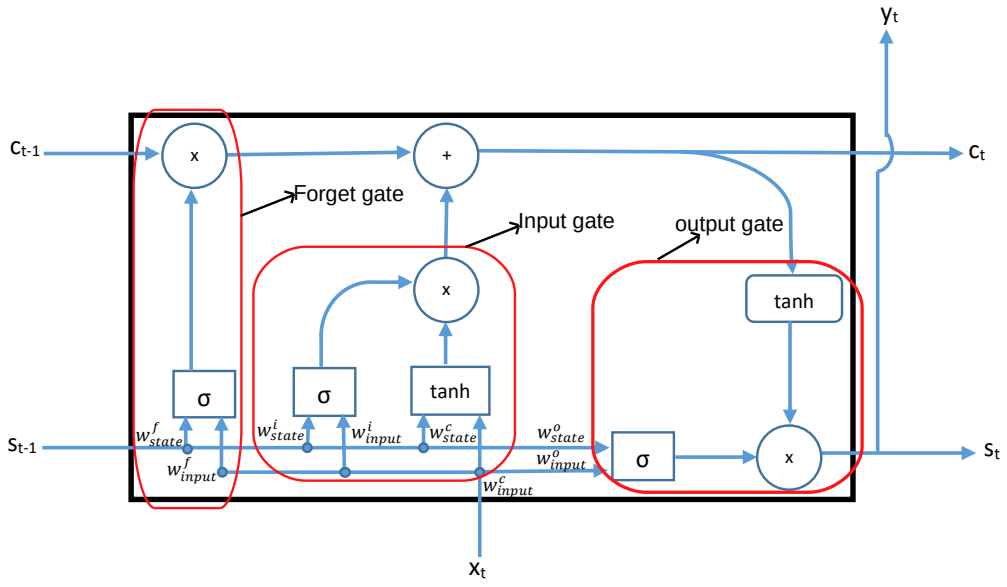


Figure 2.18: A Long Short Term Memory (LSTM) cell

Overall, the selection process at the *forget* gate gets mathematically expressed as:

$$f_G = \sigma\left(w^f_{state} \times s_{t-1} + w^f_{input} \times x_t\right) \tag{2.19}$$

Next to the forget gate, we have two activation functions serving as the input gate. The activation functions determine which of the input features gets selected and how much of them gets combined with the memory of a current cell. In this regard, the *tanh* activation determines the type of input and selected previous state features. On the other hand, the *Sigmoid* activation determines how many of these features get selected. Mathematically, these selection processes are defined using (2.20), where $i^{selected}_t$ represents the selected input features and i^{weight}_t determines how much of the selected input features get combined with a cell's memory.

$$i_G = \begin{cases} i^{selected}_t = tanh\left(w^c_{state} \times s_{t-1} + w^c_{input} \times x_t\right), & \text{Selected input features} \\ i^{weight}_t = \sigma(w^i_{state} \times s_{t-1} + w^i_{input} \times x_t), & \text{Input weighting factor} \end{cases} \tag{2.20}$$

Following the same trend, the output gate controls how much of a LSTM cell forget and input gate outputs get passed as the current hidden state. This selection process is summarized using (2.21).

$$o_G = \sigma(w^o_{state} \times s_{t-1} + w^o_{input} \times x_t) \tag{2.21}$$

Thus, in LSTM, a cell first scales the output of the tanh activated input and previous state features with their respective weighting factor, i.e., using 2.22.

$$i^{gated}_t = i^{weight}_t \times i^{selected} \tag{2.22}$$

Following this, LSTM scales the previous cell memory with the output of the forget gate using 2.23.

$$c_t = f_G \times c_{t-1} \tag{2.23}$$

The output of the input gate and the current cell's memory later get combined and passed through a *tanh* activation as a candidate output using 2.24.

$$O^{candidate} = tanh(c_t + i_t^{gated}) \tag{2.24}$$

Finally, LSTM cell computes its current hidden state (output) by weighting the candidate output with the activation value of the output gate, i.e., using 2.25.

$$O_{LSTM} = O^{candidate} \times o_G \tag{2.25}$$

In general, through the three gating operations, LSTM cells can choose to either completely forget or retain their memory states. To this end, LSTM can retain its memory in a controlled manner, i.e., without rapidly exploding or vanishing gradients. However, even if LSTM is better capable of capturing Markovian chain behaviors, they are resource intensive compared to *convolutional* layers. In this aspect, in LSTMm the three gates are constructed from fully connected *Dense* layers. Thus, as the dimension of the input embedding and layer stacking increases, the number of trainable weights significantly grow [29]. With this said, we will finalize the discussion of layers in Recurrent Neural Network (RNN). Moreover, we will leave concepts related to other versions of recurrent neural networks, such as the Gated Gated Recurrent Unit (GRU) to the interested reader.

2.3.2 Back-propagation, Activation Functions and Layer Initialization

In neural networks, we aim at tuning the weights of neurons or layers in general to meet the demands of a given cost (objective) function. In earlier days, the weights of neurons were either updated manually or through simple difference operations [54]. However, with such approaches, it is often difficult to construct deep neural networks that handle complicated tasks. To address this issue, the authors in [64] proposed the concept of back-propagation designed to update weights using gradients. To further elaborate on this matter, let us assume that we have a neural network built from three stacked *Dense* layers, i.e., L_1, L_2, L_3. Moreover, let us also further assume the layers respectively have a fully connected N_1, N_2, N_3 number neurons, where $N_i \geq 1$. In addition to this, to make the network more generic, let us also assume the neurons have a generic activation ($f(.)$) and use (2.11) to generate outputs. In practice, in such setups, L_1, L_2, L_3 are respectively called input, hidden, and output layers. In practice, we often utilize such setups to minimize an objective function that has the form given in (2.26), where S is a function of the neural network weights and activation functions, and Y is a true value. In practice, if the exact values of Y are known, i.e., at least at the time of training, then the network is said to be trained under a supervised setup. However, when this is not the case, the network is said to be trained under an unsupervised setup [29]. Overall, in both cases, we only have control over S which is a composition of weights and inputs. Thus, in practice, neural networks

utilize two passes to update S to minimize C, i.e., forward and backward pass.

$$C = f(S, Y) \tag{2.26}$$

In reality, a forward pass updates the magnitude of S, whereas a backward pass (back-propagation) updates the components that make up S or specifically weights. In this aspect, in our example network, a forward pass will compute the output of each layer using the format given in (2.27), where $Z_i = \{z_1^i,\ z_2^i, \ldots,\ z_{N_i}^i\}$ is the outputs of a layer i that has N_i neurons. Moreover, $X_i = \{x_1^i,\ x_2^i, \ldots,\ x_{M_i}^i\}$ corresponds the input of layer i. Additionally, in reality, layers in deep neural networks have two sets of weights, i.e., weights that connect neurons to their inputs and weights that connect neuron outputs to the next layer. In order to represent this concept in (2.27), we have written the individual weights of individual neurons as $w_{i,j}^{l,k}$ for $1 \leq l \leq N_i$ and $1 \leq k \leq M_i$. In the given representation, if $i = j$, we are talking about weights connecting neurons at layer i to their inputs. Moreover, the specific neuron under discussion gets indicated by the value of l. On the contrary, if $i \neq j$, they represent weights connecting the outputs of given layer neurons to its successive layer. In reality, the output of a given layer is an input for its successor. To this end, we can express the later representation of the weights with the former one. For instance, $w_{1,2}^{1,1}$ represents the weight connecting the output of the first layer's first neuron to the first neuron of layer two. This could also be written as $w_{2,2}^{1,1}$.

$$Z_i = \begin{bmatrix} z_i^1 \\ z_i^2 \\ \vdots \\ z_i^{N_i} \end{bmatrix} = f\left(\begin{bmatrix} b_{i,j}^1 \\ b_{i,j}^2 \\ \vdots \\ b_{i,j}^{N_i} \end{bmatrix} + \begin{bmatrix} w_{i,j}^{1,1} & w_{i,j}^{1,2} & \ldots & w_{i,j}^{1,M_i} \\ w_{i,j}^{2,1} & w_{i,j}^{2,2} & \ldots & w_{i,j}^{2,M_i} \\ \ldots & \ldots & \ldots & \ldots \\ w_{i,j}^{N_i,1} & w_{i,j}^{N_i,2} & \ldots & w_{i,j}^{N_i,M_i} \end{bmatrix} \begin{bmatrix} x_i^1 \\ x_i^2 \\ \vdots \\ x_i^{M_i} \end{bmatrix} \right) = f(W_{i,j}\ X_i\ +\ B_{i,j}) \tag{2.27}$$

With these understandings, we can now write the input output relationship of the three *Dense* layers for a forward pass as (2.28).

$$\begin{aligned} Z_1 &= f(W_{1,1}\ X_1\ +\ B_{1,1}) \\ Z_2 &= f(W_{2,2}\ Z_1\ +\ B_{2,2}) \\ Z_3 = S &= f(W_{3,3}\ Z_2\ +B_{3,3}) \end{aligned} \tag{2.28}$$

In a neural network, backward pass (back-propagation), is designed to intelligently update S such that the cost function C (2.26) is minimized. Based on the derivation presented so far, we can achieve this objective by observing the rate of change of C with respect to each weight. Mathematically speaking, this can be achieved by taking the partial derivative of the cost function with respect to the weights, i.e., gradients of the cost function. However, in reality, there are weight matrices that are not directly related to the cost function say for instance $W_{1,1}$ in (2.28). In this aspect, [64] proposed to utilize the chain rule of partial derivatives. For instance, for (2.28), we can compute the rate of change of $C(S, Y)$ with respect to $W_{1,1}$ using (2.29).

$$\frac{\partial C(S,\ Y)}{\partial W_{1,1}} = \frac{\partial C(S, Y)}{\partial Z_2}\ \frac{\partial Z_2}{\partial Z_1}\ \frac{\partial Z_1}{\partial W_{1,1}} \tag{2.29}$$

In addition to this computation, [64] also proposed to update the weights ($W_{i,j}$) and biases ($B_{i,j}$) using (2.30), where η is a weighting factor of the gradients which is often called learning rate. Moreover, in practice, (2.30) is commonly called the gradient decent or back-propagation.

$$\begin{aligned} W_{i,j} &:= W_{i,j} - \eta \frac{\partial C(S,Y)}{\partial W_{i,j}} \\ B_{i,j} &:= B_{i,j} - \eta \frac{\partial C(S,Y)}{\partial B_{i,j}} \end{aligned} \tag{2.30}$$

Practically, different factors affect the performance of gradient descent. In this regard, one major factor would be how often we update the weights of the network layers. In this regard, we have three possibilities that could lead to three different performance outcomes. In the first case, we can choose to update the weights of a network after every forward pass of an input. When this is the case, we often will have slow convergence of the network since every backpropagation will pull the gradients in a different direction. In other words, we will zigzag towards a global optimum that could get missed due to short interval updates. On the contrary, instead of updating a network per input sample, we can also wait to see every training input and update the weights by taking aggregated gradients. However, in this case, we also have a slow convergence due to a slow rate of update that could easily worsen as the number of training samples increases. To this end, in most practical cases, neural networks are often updated using batches of the training input [29].

In reality, the rate of weight updates is not the only factor affecting the operation of gradient descent. In this context, different variants of the gradient descent have got proposed to address different gaps observed in the algorithm [65]. If we pause at this point and leave the details of such works to the interested reader, we can speculate additional influencing factors by just looking at (2.28) and (2.29). In (2.28), we can see that the activation values get well embedded into the input and output of each layer as the input progresses through the network. Thus, even if it is not explicitly shown in (2.29), it will have a say in the outcomes of (2.29) and 2.30. This, in turn, could have either a negative or positive effect on the overall performance of a neural network. Additionally, in neural networks, layer weights are often initialized with random values. This initialization often dictates from where the gradients start to decent while looking for global minima. Thus, in practice, an improper weight initialization could force gradient descent to get stuck in local minimums. Moreover, it could also produce weights that could vanish or explode while computing the gradient descent. To this end, in the next two subsections, we emphasize these two key factors and present challenges and trends associated with them. Moreover, such assessments have helped us to make proper neural network parameters selection in our search for a time series average augmentation (generative) neural network setup. Additionally, it will also help the reader understand the underlying reason behind the selection of parameters in our proposed approaches.

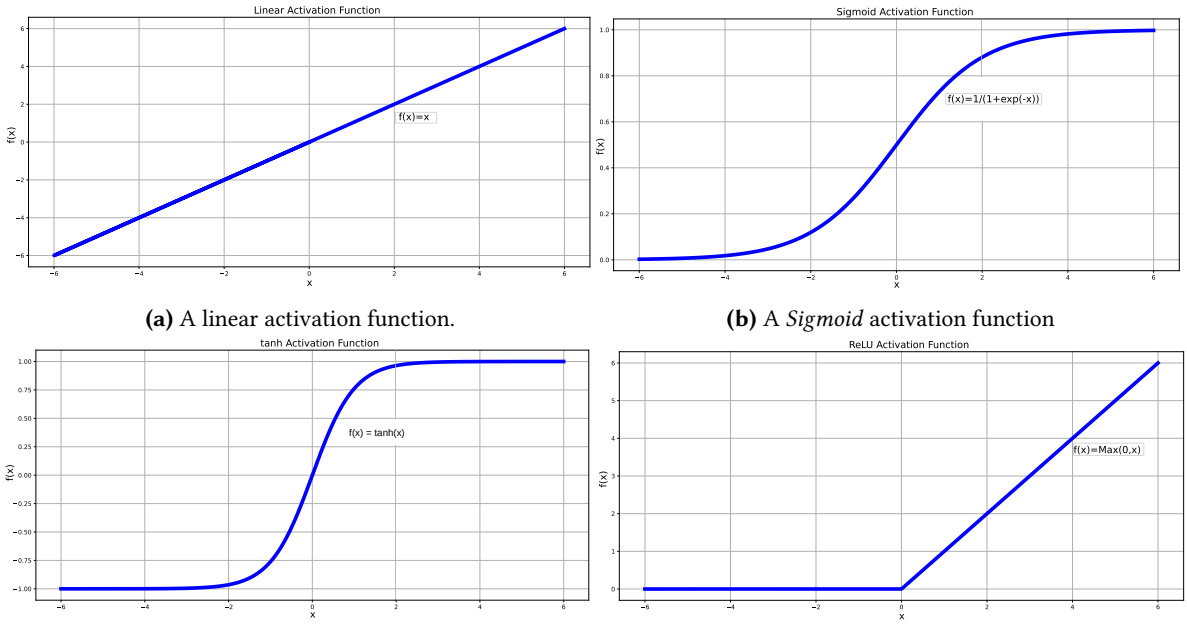
2.3.2.1 Activation Functions

In neural networks, activation functions and layer initialization are key parameters that play a role in the performance of proposed architectures. In this context, on one hand, proper activation functions

enable layers to perform complex transformations fitting to the task at hand. On the other hand, layer initialization significantly influences where gradient vectors start to decent along the curves of an objective function [29], [54], [66]. In other words, they play a key role in whether we settle for a local or global optimum. In terms of activation functions, in earlier days, neural networks mainly utilized linear activation functions. However, even though a linear activation function is easy to deploy, it has at least two basic limitations [29], [54]. First, in deep neural networks (networks that have multiple layers), the magnitudes of linear activation could quickly grow after small training iterations (epochs). Thus, a deep neural network that fully utilizes linear activation is often susceptible to an exploding gradient problem, i.e., (2.30) could significantly increase. Moreover, neural networks based on linear activation functions assume that the optimized objective function can get modeled using linear functions or the composition of linear functions. However, in practice, this is not always possible [23], [54]. To this end, researchers have proposed a range of non-linear activation functions to meet the demands of relatively challenging objective functions. In this aspect, Figure 2.19 shows some of the most common non-linear activation function, i.e., the *Sigmoid* (2.19 (b)), hyperbolic tangent (*tanh*) 2.19 (b) and Rectified Linear Unit (*ReLu*) 2.19 (d). Furthermore, the governing mathematical equation of these activation are also shown in (2.12) and (2.31).

$$R(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \qquad S_{max}(x_i) = \frac{\exp^{x_i}}{\sum_{i=1}^{C} \exp^{x_i}} \qquad tanh(x) = \frac{1 - \exp^{-2x}}{1 + \exp -2x} \tag{2.31}$$

In practice, contrary to the others, the*Softmax* activation is commonly deployed at the end of classifier networks that are trying to identify class labels from C categories. This is because the output values of N *Softmax* activated neurons sum up to one. Consequently, *Softmax* activation values often serve as indicators of the probability of a given input belonging to a certain class. However, despite this unique



(a) A linear activation function.

(b) A *Sigmoid* activation function

(c) A hyperbolic tangent activation function

(d) A *ReLU* activation function

Figure 2.19: Some practically available neuron activation functions

nature of *Softmax*, the activation values of *tanh*, *Sigmoid* and *Softmax* are bounded in magnitude, i.e., within the rages of $[-1,\ 1]$, $[0,\ 1]$ and $[0,\ 1]$ [29]. In practice, such output bounding could be useful in mitigating the exploding gradients problem which is evident in linear activation functions. However, in reality, the output bounding also squashes a large portion of input values into smaller regions. To this end, if care is not taken, such bounded activation values could also be a major source of vanishing gradients in deep neural networks [66]. Consequently, in some cases, networks based on these functions could experience a slow rate of convergence due to small gradients. With these observations in mind, most recent neural network architectures often propose to dominantly utilize the Rectified Linear Unit (ReLU) activation function [57], [58], [60], [61]. This is because the zeroing of negative values enables ReLU to often provides sparse and nonlinear transformations. To this end, unlike *Linear* activation, ReLU is often able to avoid the exploding gradient problem. Moreover, this capability of ReLU helps it to intelligently drop out neurons and their respective weights which at times helps to avoid overfitting. However, in some cases, sparse transformations could result in neural networks that have a smaller number of active neurons. This in turn could affect the learning capability of a network [29]. In practice, to overcome this limitation, researchers also proposed variant of ReLU, i.e., (Leaky Rectified Linear Unit (LReLU)), where $LReLU(x) = a \times x$ when $x < 0$ and $0 < a < 1$ [67]. In general, the type of selected activation function determines the overall learning process. Thus, it is up to the user to carefully select activation values that suit the underlying data. Moreover, in this paper, we only gave a brief review of activation functions that are pioneering. Interested readers can refer to Keras's documentation for the extended list of practically available activation functions[68].

2.3.2.2 Impact of Layer Initialization on Deep Neural Networks

In practice, activation functions are not the only hyper-parameters that play a role in how objective functions get optimized. In this aspect, layer weight initialization also plays a critical role in how activation values and gradient vectors behave. For instance, if we propose to initialize layer weights with zero initial values, then we are practically setting all $W_{i,j}$ of (2.28) to zero. This, in turn, will cause the gradients in (2.30) to vanish, thus making a network untrainable. On the contrary, if we choose to initialize $W_{i,j}$ with large constant values, the gradients would explode and oscillate over global minima. Such high-level analysis reveals that we should not set out to initialize layers with fixed constant values. On the contrary, the initialization should be randomized. However, the critical question becomes, what are the proper statistical parameters? Moreover, what is the implication of the parameters on the overall performance of a given network? In this regard, we find the research conducted in [66], [67] to be the most relevant to the question at hand.

In [66], the authors investigated the impact of weight initialization on the gradient and activation values of *Sigmoid*, *tanh* and *Softsign* ($f(x) = \frac{x}{1+|x|}$) activation functions. To conduct these assessments, the authors built networks that have one up to five fully connected *Dense* layers. These networks got set to have 1000 neurons in their hidden layers stacked to formulate a deep neural network performing multi-class classification. For the classification task, the authors proposed to utilize two broad categories of input datasets, i.e., Finite and Infinite datasets. In the finite input

datasets, the authors utilized images obtained from *MNIST* digits, *CIFAR-10* and *Small-ImageNet* [66]. In terms of size and dimension, the *MNIST* digits respectively constituted of 70,000 gray scaled 28×28 images of handwritten digits $(0-9)$. On the contrary, *CIFAR-10* datasets contained 50,000 gray scaled 32×32 images of an airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck. Finally, the *Small-ImageNet* contained 110,000 37×37 grayscale images of animal or objects (reptiles, vehicles, birds, mammals, fish, furniture, instruments, tools, flowers, and fruits) [66]. In general, the contents of these datasets got divided among 10 different balanced classes. Moreover, while performing the classification, the authors proposed to reserve 20,000 datasets that got equally split to formulate validation and test datasets. On the contrary, for the infinite datasets, the authors used synthetically generated images of geometric shapes (squares, parallelograms, and ellipses). In reality, since these images got generated synthetically, an infinite number of these datasets were available. Furthermore, to make the classification task more complex, the authors also proposed to generate images that could contain multiple geometric shapes which could be scaled, rotated, or shifted versions of the original shapes. In general, for all input categories (C), the authors set the cost function of the networks to the average negative log likelihood ($-\frac{1}{C}\sum_{i=1}^{C}\log p(x_i|y_i)$), were x_i and y_i are the true label of an image and its corresponding *Softmax* activation value [66].

With these setups at hand, the authors first proposed to access the implication of random weight initialization on activation values. In this aspect, they first proposed to initialize the weight of each layer using (2.32), where we have adopted the notations given in (2.27) and set $i = j$ to indicate the weights of a layer. Moreover, $1 \leq i, j \leq 5$, n is the number of neurons in a preceding layer, and U is the uniform distribution.

$$W_{i,j} = U\Big[\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\Big] \tag{2.32}$$

However, the bias ($B_{\{i,j\}}$) of each neuron were set to zero since their implications were assumed to be insignificant. After training the network, the authors observed the activation values of each layer at different iterations (epochs). To aid our discussion, we have extracted the graphical depiction of the activation values from [66] and presented them in Figures 2.20 & 2.21. These activation values got obtained when the network was trained using samples obtained from the infinite datasets category. In Figure 2.20, the solid horizontal lines correspond to the mean of the activation values, whereas the oscillating vertical lines represent the variance. According to the Figure 2.20, for a *Sigmoid* activation, inner hidden layers quickly saturated. Moreover, the last hidden softmax layer (Layer4) immediately saturated and tried to recover after 100 epochs. The authors mainly associated this behavior of the activation values with the initialization technique. In this regard, the authors argue that neural networks initialized using the weights of pre-trained networks did not exhibit such behavior. In this context, the authors speculated that, with random initialization, inner hidden layers do not immediately learn meaningful features related to the task. Thus, at the output of the classifier network, the *Softmax* activation of $WZ + B$ will initially rely on the bias B and will often be zero [66]. Moreover, since the gradient has to keep discriminating unrelated predictions, it will inform upper layers to set their activation values (Z) to zero. However, since upper layers are also trying to learn and have asymmetric activation function (*Sigmoid*), they will settle at a mid-way and start to oscillate. In reality, for a

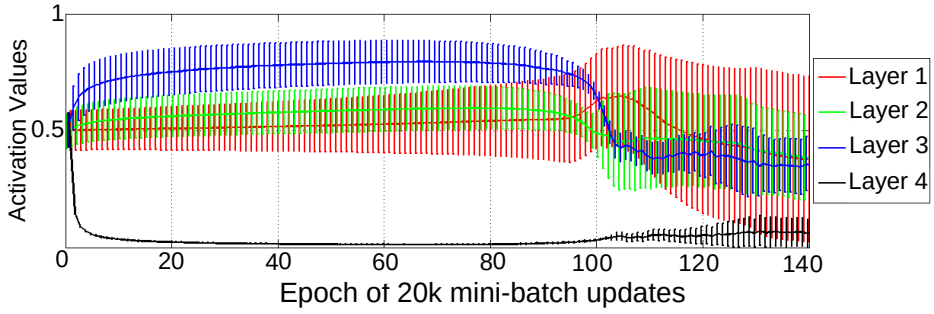
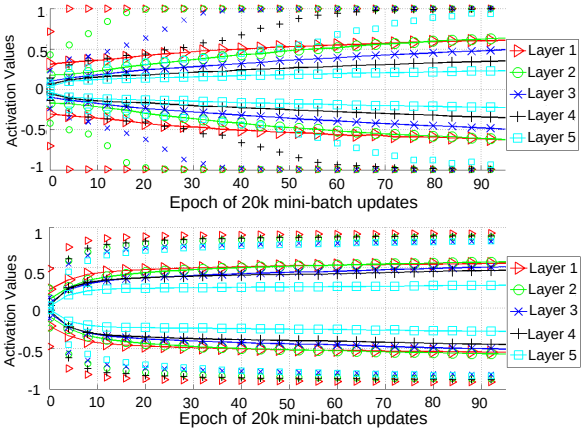


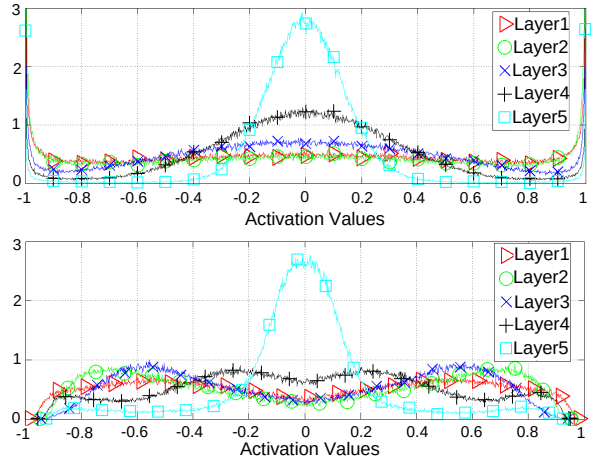
Figure 2.20: Saturation of a *Sigmoid* activated *Dense* layers [66]. The authors in [66] used this plot to demonstrate the impact of improper layer initialization on the overall outcome of neural networks. In this regard, an improper layer initialization has led to a non harmonious back-propagation of gradients across layers which forced the *Dense* layers to operate in a non harmonious manner

Sigmoid activation, each oscillation corresponds to the saturation of the activation function. However, according to (2.29), a saturated activation does not contribute to a learning process (gradients). This, in turn, will slow down the learning process of the *Sigmoid* activated network, i.e., as shown in Figure 2.20. Practically, this is in line with our previous intuition of bounded activation functions requiring more time to converge. In addition to these observations, the saturation of the *Sigmoid* activation reveals how networks based on this activation could suffer from the vanishing gradient problem.

With these observations in mind, the authors then utilized Figure 2.21 to demonstrate how the *tanh* and *Softsign* activation functions behave with the proposed random weight initialization. According to Figure 2.21 (a), both activation functions oscillated between $[-1,\ 1]$. In the figure, the markers represent the top 98 percentile, whereas markers with solid lines correspond to the standard deviation. In general, as the number of epochs progresses, the saturation starts to oscillate closer



(a) Saturation of *tanh* (top) and *Softsign* (bottom)



(b) Normalized histogram plot of the *tanh* (top) and *Softsign* (bottom) activation values

Figure 2.21: Saturation of *tanh* and *Softsign* activated *Dense* layers [66]. In (b), it is clear to observe that there is discord among the activation values of layers due to improper weight initialization. In reality, such a huge difference among the variance of activation values corresponds to unbalanced back-propagated gradients. In practice, such unbalances often get associated with bad performances of neural networks.

to zero. This behavior gets worst at the deep hidden layers. This was better demonstrated using the normalized activation plot shown in Figure 2.21 (b). The figure demonstrates that deep hidden layers have a relatively higher variance of activation values. This is because, generally speaking, deep hidden layers are closer to the output and gets relatively higher gradient updates since there are relatively small in-between saturated layers. On the contrary, layers closer to the input will have lower variance in their activation values due to a higher number of in-between saturated layers and consequently smaller gradient values. Practically, we expect neural networks to perform better when there is a coherent flow of activation values and gradients. However, this is not achievable if there is discord among layer saturation. In this aspect, [66] argued that coherence is achievable if layer initialization techniques encourage a uniform standard deviation of activation and gradient values. In this context, if we assume layers are operating at their linear regions of the *Sigmoid* and *tanh* activation functions, then we can use (2.28) and write the output of the l^{th} *Sigmoid* and *tanh* activated layer as $O = W_{1,1} \times W_{2,2} \ldots W_{l,l} \times X + C$. Moreover, we can safely assume the inputs to be independent and have a variance of $VAR[X]$. Under this consideration, the main contributor to the variance observed in gradients would be the variances of the layer's weight and the weight of a predecessor layer. Moreover, due to the relation between activation values and gradients or (2.29), the variance of the gradient values will have a multiplicative correlation with layer weights. In this context, for the initialization given in (2.32), each layer's gradient values will have a variance of $\frac{1}{3}\frac{1}{n}$ and decrease at a rate of $\frac{1}{n}$ as we progress backward. Consequently, layers closer to the input will have a low variance in their activation due to getting updated with a slowly changing gradient. With this argument, [66] proposed that layers should get initialized with (2.33) which is expected to guaranty uniform variance across gradient values [66]. In (2.33), n_i, n_{i-1} correspond to the number of neurons at layer i and its immediate predecessor. The authors called this initialization the normalized initialization. However, in practice, it is commonly known as the *Xavier/Glorot* uniform layer initialization technique (*He* uniform) [68]. In [66], the authors demonstrated that (2.33) was able to meet its main objective using the plots shown in Figure 2.22 [66]. In the plots, layers that were initialized with (2.33) showed a uniform variance across the activation and gradient values of layers, i.e., according to the two bottom plot in Figure 2.22 (a) & 2.22 (b).

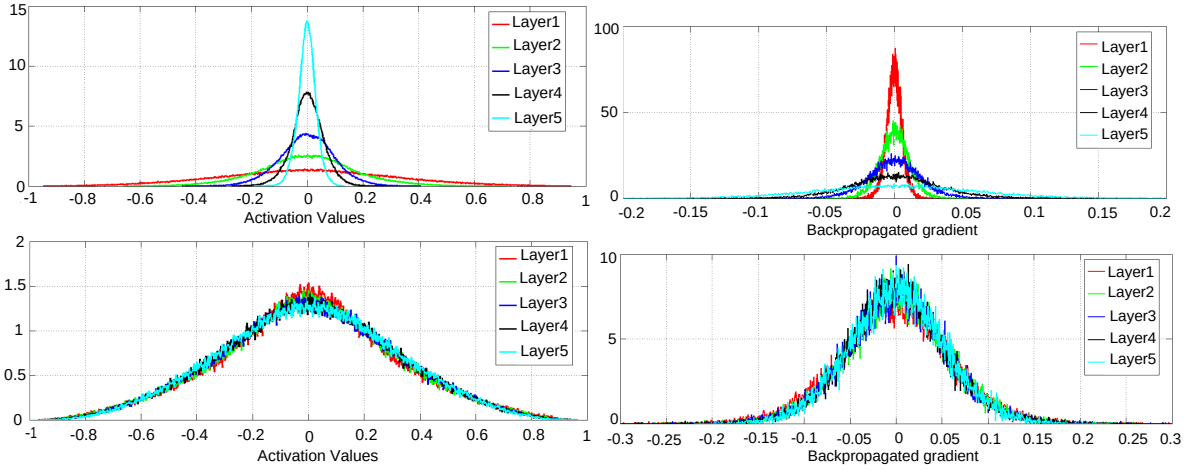
$$W_{j,i} = U\left[\frac{-\sqrt{6}}{\sqrt{n_i + n_{i-1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i-1}}}\right] \tag{2.33}$$

However, even though [66] provided an exceptional insight on layer initialization, a latter work argued that (2.33) often works well with *Sigmoid* and *tanh* activation functions [67]. In this regard, [67] presented the *ReLu* as an example and argued that in this case we should aim for a variance of $\sigma^2 = \frac{2}{n_{i-1}}$ rater than $\sigma^2 = \frac{2}{n_i + n_{i-1}}$. Furthermore, the authors showed that *ReLu* activated layers should be initialized using (2.34) rather than (2.33). This proposal later came to be known as the *He* uniform initialization [68].

$$W_{j,i} = U\left[\frac{-\sqrt{(6)}}{\sqrt{n_i}}, \frac{\sqrt{(6)}}{\sqrt{n_i}}\right] \tag{2.34}$$

In [67], the effects of (2.34) were assessed using a 30 layered neural network which was composed of *Convolutional* and *dense* layers [67]. However, in reality, due to the complexity of the proposed

network architecture, the authors found it difficult to attribute all of the observed performance improvements to the initialization [69]. Nevertheless, most practical neural networks utilized the initialization techniques presented in this subsection. Moreover, in practice, their normal distribution forms are also defined and widely deployed. However, when this is the case, the variance associated with layers weights become $\sigma^2 = \frac{2}{n_i}$, i.e., for *he* normal, and $\sigma^2 = \frac{2}{n_i+n_{i-1}}$ for *Xavier/Glorot* normal initialization.



(a) *tanh* activation values initialized with (2.33) , i.e., bottom, and (2.32) top

(b) *tanh* backpropagated gradients initialized with (2.33), i.e., bottom, and (2.32) top.

Figure 2.22: Normalized histogram plots of back-propagated gradients and activation values of *tanh* activated *Dense* layers. The plots demonstrate the impact of layer initialization on activation values and gradient propagation [66]

With these basic technicalities of neural networks in mind, we will next present a recent neural network-based time series averaging technique, i.e., the Diffeomorphic Temporal Alignment Network (DTAN) [25]. In reality, this approach introduced neural networks into the domains of time series averaging for the first time. Moreover, it obtained a state-of-the-art morphed (warped) space registration.

2.3.3 A Neural Network Based Time Series Averaging

Practically, we can safely assume that the time warping performed in averaging heuristics based on DTW as a registration process. In this context, if we assume members of the averaged sets are vectors in $\mathbb{R}^M$, then DTW based averaging techniques first utilize DTW to transform the vectors into a $\mathbb{R}^\tau$ space where $\tau \geq M$. In practice, in dominant DTW based averaging techniques such as DBA, the warping is performed to minimize the discrepancy of the warped series to a warped template (landmark). In other words, in such cases, averaging techniques aim at registering the warped series in $\mathbb{R}^\tau$ to the warped version of the template. If we see such averaging techniques from this perspective, we can see that there are approaches following similar transformations in image processing and functional data analysis [33], [70]–[72]. Generally, the alternatives assume sequential data sets such as time series as samples of underlying continuous functions. Thus, in such techniques, there is an assumption that a given averaged set gets generated by taking samples of a continuous

function. Moreover, if members of the averaged set are expected to represent similar entities, then temporal distortions (phase variations) are mainly assumed to occur due to a difference in sampling rates. Consequently, most of the alternative warping (registration) techniques propose to take the following three key steps to minimize such phase distortions [70]. First, they try to identify and propose a technique that governs the trajectories of the original series's time stamps. In reality, given a time series $X = \{x_1, x_2, \ldots, x_M\}$, the alternative registration techniques assume each $x_i \in X$ are obtained by taking samples from a continuous function f(t), i.e., $x_i = f(t_i)$. Thus, given a time series $X = \{x_1, x_2, \ldots, x_M\}$ and its time stamps $t = \{t_1, t_2, \ldots, t_M\}$, we can assume the estimation of new trajectories as defining new sampling times $\beta = \{\beta_1, \beta_2, \ldots, \beta_M\}$. In reality, these new trajectories get defined with the intention of re-sampling $f(t)$ such that the re-sampling X minimizes its discrepancy compared to a landmark. To this end, the next step taken by the alternative warping techniques is to define a re-sampler. Moreover, they get expected to evaluate the quality of the registration. In this aspect, as a final and third step, they define a landmark that the morphed series gets compared with. In this regard, they can choose to follow two possible approaches. In the first scenario, they could select a given time stamp β_i and expect all warped (morphed) series to have similar values at β_i. This approach is commonly called landmark-based registration. However, in this case, the landmark is often selected based on prior knowledge about the location of the most descriptive shapes of the underlying continuous functions, for instance, peaks, troughs, etc. Thus, this approach requires some degree of knowledge about the underlying continuous function [70]. Contrary to this, an alternative solution is to generate a full landmark from the transformed series, for instance, the arithmetic means of the transformed series [25], [73]. When this is the case, we often call the warping (morphing) process registration. In general, despite such differences, the advantage of alternative registration techniques is the possibility of a warping technique that could easily get integrated with neural networks.

Despite this potential, investigations aiming to utilize this alternative form of registration for time series averaging are limited in number. To the best of our knowledge, we can identify two proposals in this regard, i.e., Diffeomorphic Temporal Alignment Network (DTAN) [25] and the Square Root Velocity Field Registration Network (SrvfRegNet) [73]. However, as compared to DTAN, the performance of SrvfRegNet was evaluated on a limited number of data sets. Moreover, SrvfRegNet mainly emphasized on the quality of the warping rather than the average. To this end, in this subsection, we give more emphasis to DTAN.

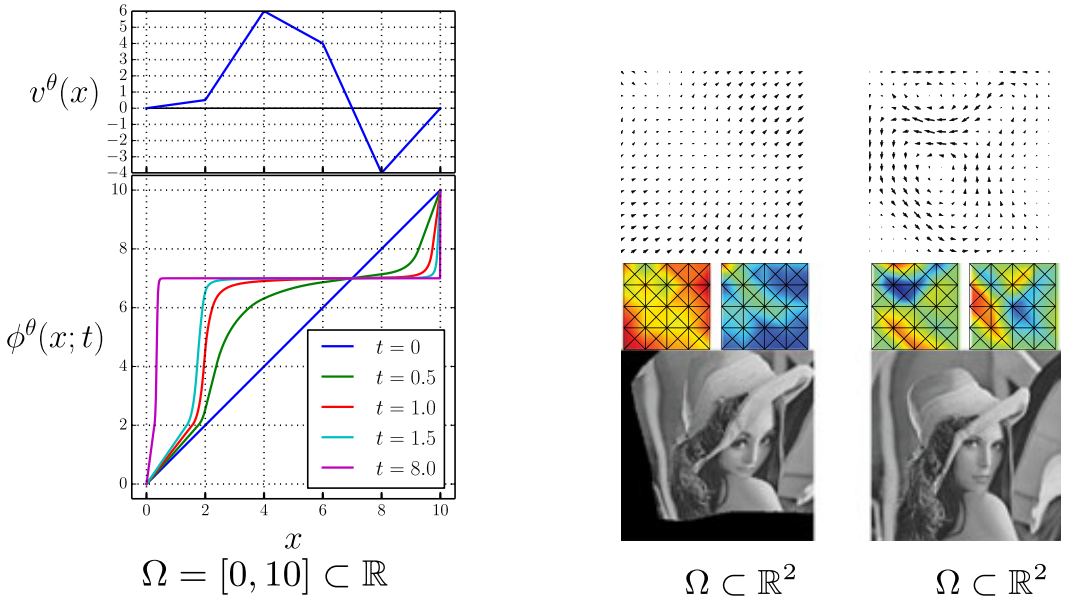
2.3.3.1 Diffeomorphic Temporal Alignment Network

The foundation for the concepts of DTAN were laid in [74]. The main contribution of [74] was to establish the concept behind a velocity field based diffeomorphic transformation. In this aspect, [74] proposed to utilize continuous piecewise functions that define the trajectories of time stamps. In [74], these piecewise functions were called Continuous Piecewise Affine velocity fields (CPA fields). Based on classical physics, a velocity field is known to have both magnitude and direction, where the magnitude shows the rate of change of distance ($v(x;t) = \frac{dx}{dt}$). With this understanding in mind, [74] argued that a given time stamp can be moved from point A to point B by integrating a velocity field

directed from A to B. This concept was mathematically formulated as (2.35), where $\phi^{\theta}(x,t)$ is the parametric trajectory of a time stamp x by the parametric velocity fields $v^{\theta}(x,t)$.

$$\phi^{\theta}(x;t) = x + \int_0^t v^{\theta}(\phi^{\theta}(x;\tau))\, d\tau \tag{2.35}$$

To demonstrate the concepts discussed so far, we have extra Figure 2.23 from [74]. For instance, in Figure 2.23 (a), the closed interval $[0,\ 10]$ was divided among a set of tessellations (vertical girds). Moreover, within these tessellations, a one dimensional parametric CPA velocity field $v^{\theta}(x)$ was defined. Finally, $v^{\theta}(x)$ was integrated for different durations of t to define different parametric trajectories of x ($\phi^{\theta}(x;t)$). On the contrary, in Figure 2.23 (b), two dimensional CPA velocity fields were utilized for a diffeomorphic transformation of an image.



(a) 1D diffeomorphism **(b)** 2D diffeomorphism

Figure 2.23: A demonstration of CPA velocity field based diffeomorphic transformation [74]

However, practically, projecting time stamps by itself does not guarantee meaningful and useful warping of temporal datasets. To this end, in practice, additional constraints are placed that preserve some desirable mathematical properties. For instance, in practice, we often desire transformation techniques to be affine that has the form $Y = A\,X\ +\ B$, where $A\ \in\ \mathbb{R}^{(N,\,M)}$ and $\{X,\ Y,\ B\}\ \in\ \mathbb{R}^{M}$. Thus, this way, a transformation can preserve lines and parallelism. Overall, under affine transformation, the inevitable matrix A is often expected to scale, rotate, etc. On the contrary, the vector B gets used for translation. With this understanding, [74] went further and showed that the CPA fields meet the affine requirements. In general, [74] proposed to divide the time axis of the morphed series into tessellations. It then defined the CPA fields using piecewise linear functions bounded within each tessellation. Thus, they were able to meet the mathematical behaviors of an affine transformation [74]. From the perspective of time series averaging, the affine nature of the CPA velocity field diffeomorphism enables the morphing not to entangle coordinate values of the morphed series. Moreover, it helps to establish an affine $\mathbb{R}^M$ to $\mathbb{R}^M$ warping not evident in previous time series averaging techniques.

However, before deploying such a transformation technique, two additional questions should get answered, i.e., in addition to the affine requirement. In this regard, the first question that comes to light is, how do we estimate the appropriate velocity field for a given data? Moreover, given a velocity field, how do we guide a time warping in a manner that it achieves registration of a transformed set to a landmark? The first proposal that addressed these questions in the context of images was the Spatial Transformer Network (STN) [71]. The STN aimed to learn spacial invariant feature maps in *Convolutional* networks. To meet this objective, [71] proposed to estimate the velocity fields using *Convolutional* networks. It then used the estimated velocity fields to mitigate the effects of translations on extracted feature maps of input images. To extend this concept to temporal data sets, [25] proposed the Temporal Transformer (TT) layers shown in Figure 2.24. Generally, the proposed transformer

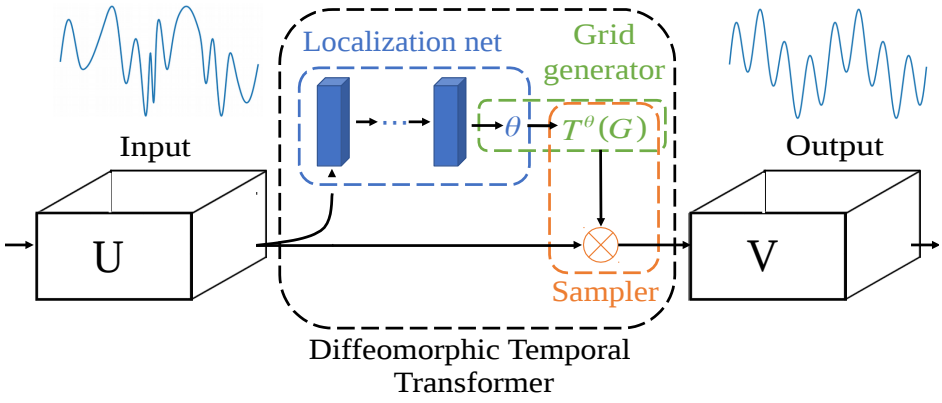


Figure 2.24: Temporal Transformation (TT) layer [20]

layer has three components, i.e., the localization net, the grid generator and a sampler. In a TT layer, the localization network is fed with a time series ($\mathcal{U} \in \mathbb{R}^M$). Given the series, a localization network outputs the parameters θ of v^{θ}. Following this, a parametric grid generator outputs M evenly space one dimension grid points $G = (p_n)_{n=1}^{M} \subset [-1, 1]$. In reality, the grid generator is expected to estimates the parametric trajectories that were supposed to be computed using (2.35). However, (2.35) does not clearly state how to compute the values (amplitudes) of the new trajectories. In this regard, [20] proposes a differentiable re-sampler. The re-sampler outputs the transformed versions $V_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,N}\} : V_{i,k} \in \mathbb{R}^M$ of the original time series $U_i \in \mathbb{R}^M$ using (2.36), where $v_{i,n}$ and $u_{i,m}$ are the values of the transformed and original time series at time stamps n and m. Moreover, $p_{i,m}^{warped}$ is the m^{th} entry of the sampling grid.

$$v_{i,n} = \sum_{i,m=1}^{N} u_{i,m} \max(0, 1 - |p_{i,m}^{warped} - m|) \tag{2.36}$$

In addition to the re-sampler, [20] also defined the partial derivatives of $V_{i,n}$ with respective to $U_{i,m}$ and $p_{i,m}^{warped}$ as shown in (2.37) & (2.38). These partial derivatives become handy in the computations of gradients.

$$\frac{\partial V_{i,n}}{\partial U_{i,m}} = \max(0, 1 - |p_{i,m}^{warped} - m|) \tag{2.37}$$

$$\frac{\partial V_{i,\,n}}{\partial p_{i,m}^{warped}} = \begin{cases} 0, & \text{if } |m - p_{i,m}^{warped}| \geq 1 \\ 1, & \text{if } m \geq p_{i,m}^{warped} \\ -1, & \text{if } m < p_{i,m}^{warped} \end{cases} \tag{2.38}$$

The TT layer answered how to perform the warping of input time series. However, the transformation would be meaningless in the context of time series averaging without an appropriate guidance (objective function). In this context, [25] proposed to minimize the objective function shown in (2.39), where $w,\ K,\ l2$ are respectively the weight of the localization network, number of averaged time series and L2 norm given in (1.4).

$$F_{registration}\big((U_i)_{i=1}^{K};\ w\big) = \frac{1}{K}\sum_{i=1}^{K}\left\|v_i^{\theta}(U_i;\ w) - \frac{1}{K}\sum_{j=1}^{K} v_j^{\theta}(U_j;\ w)\right\|_{l2}^{2} \tag{2.39}$$

In reality, in (2.39), we assumed the averaged set is composed of a single class. However, in practice, such as in the case of the *CBF* dataset, there could be numerous classes within a single averaged set. If this is the case, (2.39) gets computed for each class label. To this end, DTAN gets considered as a supervised averaging technique, i.e., it requires class labels at training. However, practically, all of the DTW based averaging techniques can also be considered supervised in the context of multi-class time series averaging. This is because, in such cases, we had to manually separate the classes before deploying the DTW averaging techniques. However, unlike DTW based techniques, DTAN is capable of transfer learning. In this regard, we can train DTAN using a training split and later utilize the trained network to morph unseen datasets. Thus, this way, it is possible to combine the previously morphed series with the new one to update a previously estimated average. To this end, DTAN is capable of updating its estimate without the need for costly re-runs.

However, similar to DTW based averaging techniques, DTAN also accessed the quality of the estimated mean in the transformed (morphed) space. In this context, in [25], the quality of the estimated averages gets accessed using a one nearest centroid classification on the morphed series and their respective arithmetic mean [18]. To conduct the classification, DTAN utilized 83 data sets obtained from the UCR [2]. The repository contains 128 univariate time series collected from different application domains. Moreover, the datasets are organized using train and test splits containing two or more classes. In the evaluation process, DTAN initially morphs the test splits using a trained TT layer. It then conducted one nearest centroid classification using euclidean distance, an average estimated from a training split, and the morphed test split. In the context of estimation quality, DTAN achieved a state-of-the-art registration of morphed series to their morphed space arithmetic means. Practically, DTAN did not access the implication of this outcome in the time domain or in a space that was not utilized for the morphing. However, in reality, this is also evident in DTW based averaging techniques which measure the quality of their estimates in DTW space [14]–[16]. In general, in this regard, we are not able to identify an averaging technique that evaluates its estimate in a neutral space, i.e., a space that is not utilized in the estimation process.

2.3.4 On Some Renown Convolutional Neural Network Architectures

In Neural networks, layer organization (architecture) has a significant role in the performance of the networks. In this dissertation, we mainly base our proposals on *Convolutional* layers and *Convolutional* networks in general. We make this choice since *Convolutional* layers were found to perform better on shapes and feature analysis which is in line with our main objective [7], [29], [55], [57], [58], [60], [61]. Moreover, due to their definition, networks constructed from *Convolutional* layers often have less computational requirement as compared to *Dense* and LSTM based networks. To this end, we find it convenient to review some of the renowned *Convolutional* network architectures we customize in our proposals. In this regard, we will give a review of the Visual Geometric Group 16 (VGG16), the Residual Network (ResNet) and the Inception architectures [57], [58], [60].

2.3.4.1 The Visual Group Geometry Group 16 Architecture

The VGG16 architecture was proposed in [57] with the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVR) in mind. In the competition, the VGG16 achieved a 92.7% top 5 classification accuracy which helped it to secure second place in the domain. Practically, the VGG16 architecture achieved this classification accuracy by systematically stacking *Convloutional* layers. In this regard, [57] proposed six different layer configurations which were named as A, $A-LNR$, B, C, D, E. However, from these configurations, only $A-LRN$ utilized the Local Response Normalization (LRN) layer. A LRN is a non-trainable layer proposed in a predecessor network architecture named AlexNet [75]. In [75], a LRN layer performed a square normalization of its input [75]. However, in [57], a LRN layer had little impact in the context of network performance. To this end, the authors only utilized this layer in one of the six investigated configurations. Besides this uniqueness of $A-LNR$, all configurations were composed of five stacks of *Convolutional* layers terminated by three fully connected Dense layers. In general, the number of layers in the configurations ranged from 11 to 19, i.e., while excluding the *MaxPooling* layers. Moreover, the channel size of the *Convolutional* stacks sequentially increased from 64 to 512 channels of features. Additionally, in most configurations, the *Convolutional* kernel size was fixed to 3. However, for the VGG16-C setup, [57] introduced a 1×1 *Convolutional* layers to increase the non linearity of the network without affecting its receptive fields. In the context of activation functions, all except the last *Dense* layer used the ReLu activation function. On the contrary, the last *Dense* layer used the *Softmax* activation function since the task at hand was a multi-class classification. Finally, all configurations got trained using 224×224 colored images that have 1000 categories. In general, in [57], the overall layer arrangements of the six VGG16 architectures were summarized as shown in Table 2.3. In the table, the *Convolutional* layers are identified as $conv\langle\ x\ \rangle - \langle\ y\ \rangle$; where x, y correspond to the kernel and filter size. Moreover, for the Dense layers (*Dense-x*), x corresponds to the number of neurons.

As compared to its predecessor, i.e., AlexNet, the VGG16 avoided the utilization of large *Convolutioal* kernel sizes such as a (5×5), (7×7) and (11×11) kernels. Practically, the main argument behind kernel reduction is twofold. First, large *Convolutional* kernels (receptive fields) implied more number of trainable parameters. For instance, if we assume K stacked *Convolutional* layers that

Table 2.3: Different Versions of the VGG16 architecture [57].

A	A-LNR	B	C	D	E
Total number of layers excluding *MaxPooling* layers					
11 layers	11 layers	13 layers	16 layers	16 layers	19 layers
Input 224 × 224 RGB image					
conv3-64	conv3-64 LNR	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
MaxPooling					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
MaxPooling					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
MaxPooling					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
MaxPooling					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
MaxPooling					
Dense-4096					
Dense-4096					
Dense-1000					
Softmax					
Number of parameters in millions					
133	133	133	134	138	144

have (7×7) *Convolutional* kernels and C channels, then the stack will have $K \times (49\ times\ C^2)$ trainable parameters. This is huge as compared to the $K \times (9 \times C^2)$ trainable parameters of a stack built from a layers that have (3×3) *Convolutional* kernel. Secondly, in practice, large number of trainable parameters are known to have a high correlation to over-fitting. In this regard, [57], [75] acknowledged that overfitting is a problem in AlexNet which utilized large *Convolutional* kernels. On the contrary, the VGG16 architecture performed better in this aspect. Moreover, [57] also argued that the VGG16 can mimic an effective receptive field of (5×5) and (7×7) by stacking two or three (3×3) *Convolutional* layers [57]. With these technicalities in mind, the six versions of the VGG16 got trained using the parameters shown in Table 2.4 [57]. In Table 2.4, *L2* regularization was used as a weight (kernel) penalty that controls layer weights from increasing significantly. In practice, large

Table 2.4: Parameters that are used to train the different VGG16 architectures [57]

Parameters	Values
Weight initialization	$\mathcal{N}(0,\ 10^{-2})$
Batch	256
Momentum	0.9
Dropout rate	0.5
L2 regularization	5×10^{-4}
Learning Rate (LR)	$10^{-}2$
LR decreasing rate	10
No. of LR decreases	3
Training Split	1.3 Million
Validation split	50K
Test Split	100K

weight values correspond to overfitting. To this end, in reality, *L2* regularization often gets used to discourage such weight values in neural networks [29]. However, as discussed in a previous subsection, weight initialization also have a significant role in network performance. In this aspect, [57] proposed to utilize a normal distribution with a variance of $\sigma^2 = 10^{-2}$. However, the authors later acknowledged that they became aware of the Glorot initialization [66] after they submitted their original work. After training the networks with these configurations, they got evaluated using a top 1 and top 5 classification accuracy. The former evaluation assumes an input is correctly classified if the highest *Softmax* activation corresponds to the label. On the contrary, the latter assumes correct classification if one of the top five Softmax activation values corresponds to the true class label. With this evaluation setup, the VGG16-*D* and VGG16-*E* versions of the VGG16 architectures equally achieved the highest classification accuracy. In this regard, they respectively obtained the top 1 and 5 best validation errors of 24.4% and 7.1% [57]. Generally speaking, the authors associated this superior performance of the networks with their depth. However, in practice, building deep neural networks is challenging due to vanishing and exploding gradients, convergence, etc [29]. To this end, the maximum number of layers in VGG16 gets limited to 19, i.e., VGG16-E. However, a later work showed the possibility of building deeper *Convolutional* networks with the help of skip connections in [58].

2.3.4.2 The Residual Network

The ResNet architecture is proposed to address the problems of constructing deep neural networks. In practice, deep neural networks are believed to be capable of learning complex transformations that contribute to better performance [58]. In this regard, the authors in [58] asked the question of *"Is constructing better networks as easy as stacking more layers?"* In practice, the main expected challenge in this regard would be the vanishing and exploding gradient problem. However, [58] acknowledged that this challenge could significantly be solved with proper weight initialization and was shown so in different works [66], [67]. On the contrary, [58] identified that the main problem with deep neural networks was performance degradation. In this aspect, deep neural networks often were observed to first saturate at a given performance and suddenly start to degrade. The authors argued that this phenomenon should not be associated with overfitting. On the contrary, the authors argued that

the degradation is mainly caused by the location of layers which determines the complexity of thier optimization. In reality, in deep neural networks, layers do not often get optimized harmoniously. In other words, in former plain stacked *Convolutional* architectures, inner layers get more refined input than their predecessors. Moreover, the refinement further intensifies as the network goes deep. Thus, as the network's training progresses, there is a higher chance of discord among stacks of layers due to the large difference in their inputs. To address this challenge, [58] proposed to introduce a skip connection (residual links) between subsequent stacks of *Convolutional* layers using the ResNet block shown in Figure 2.25.

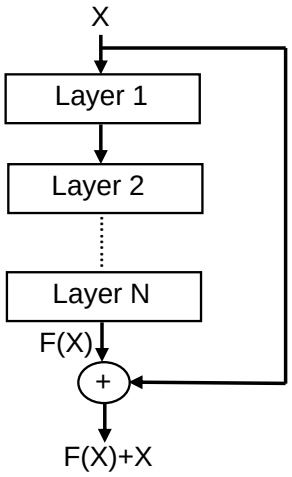


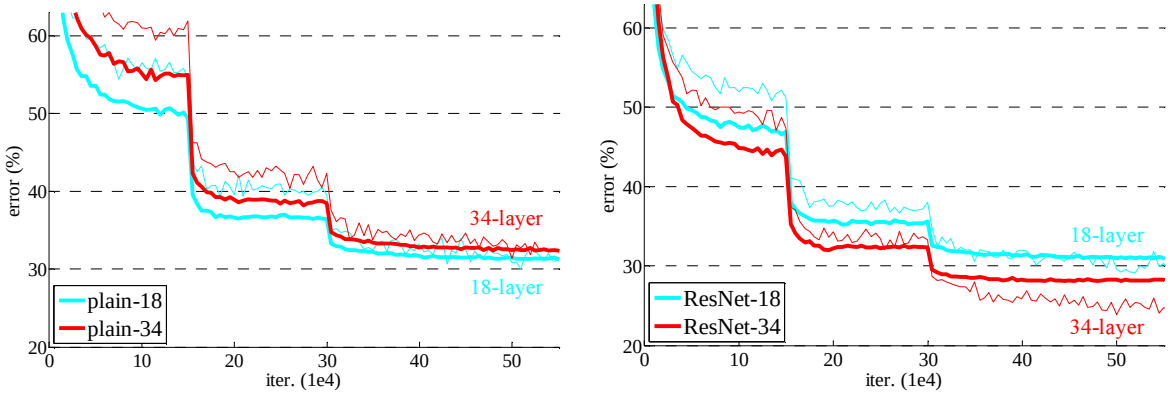
Figure 2.25: A basic Residual Network (ResNet) block [58]

In ResNet, the residual links are used to introduce harmony among the inputs of consecutive *Convolutional* stacks. Moreover, they also serve as a memory link for layers located far within the networks. With this definition at hand, [58] accessed the performance difference between feed-forward *Convolutional* networks (i.e., including some of the VGG16) and their residual counterparts. For networks based on the residue concept, [58] proposed to build networks that had up to 152 layers. Comparatively, this is significantly deep compared to the 19 layers of the VGG16. Overall, the summary of these architectures is shown in Table 2.5. Unlike the setups used in the VGG16 architectures, the authors set the *stride* of the top most *Convolutional* layer in a stack to two. This has helped *Convolutional* stacks to reduce the dimension of their input by a factor of two without the need for *MaxPooling* layers. With these setups, the authors first compared the validation errors of the 18 and 34 layered ResNet architectures, i.e., with themselves and their counterparts. These comparisons are respectively shown in Figures 2.26 (a) & (b) [58]. In Figure 2.26, the relatively thinner lines correspond to training error, whereas the bold lines correspond to validation error. According to Figure 2.26 (a), the plain feed-forward *Convolutional* architectures saturated at a 30% validation and training errors, i.e., irrespective of their depth. This was in line with the initial argument of learning saturation as networks the network depth increases. On the contrary, the ResNet's 34 layered architecture obtained better validation error as shown in Figure 2.26 (b). In general, the ResNet-152 respectively obtained a top 1 and top 5 accuracies of 21.43% and 5.71%. With this performance, the ResNet was able to win the 2015 ICLSVRC competition. This was an improvement compared to its predecessor VGG16 that obtained a top 1 and 5 validation errors of 24.4% and 7.2% [57], [58]. Finally,

Table 2.5: Different version of Residual Network (ResNet) architectures [58].

18 Layers	34 Layers	50 Layers	101 Layers	152 Layers
$kernel = 7 \times 7,\ channel = 64,\ Stride = 2$				
$3 \times 3,\ MaxPooling$				
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
$\begin{bmatrix} 3\times3,256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times\ 3, 256 \\ 3\ \times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\ \times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
$\begin{bmatrix} 3\times3\ ,512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
$AveragePooling$				
$Dense$-1000				
$Softmax$				

it should be noted the authors also tested various parameter tweaking such as, dimensional padding in the skip connections and bottlenecking of ResNet blocks with a (1×1) *Convolutional* layers (i.e., as in the cases of ResNet-50, 101, 152). However, in practice, such parameter tweaking did not significantly improve the performance of ResNet setup [58].



(a) validation and training errors of 18 & 34 layered plain *Convolutional* feed forward networks. **(b)** validation and training errors of 18 & 34 layered ResNet *Convolutional* feed forward networks

Figure 2.26: Impact of residual links in network performance [58]. In (b), the skip connections in a ResNet architecture has enabled the 34 layered neural network to achieve better performance as compared to its plain feed forward implantation and an 18 layered feed forward and ResNet neural networks.

In conclusion, the ResNet approach has significantly improved the depth and performances of *Convolutional* feed-forward neural networks. However, in practice, increasing the depth of neural networks increased their computational requirements due to a large number of training parameters. In this re-

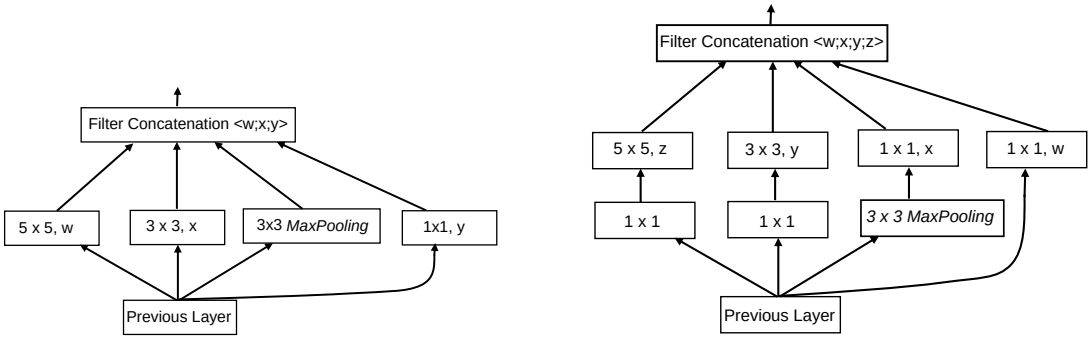
gard, the ResNet architectures shown in Table 2.5 respectively have a Floating Point Operations (FlOPs) of 1.8×10^9, 3.6×10^9, 3.8×10^9, 7.6×10^9, 11.3×10^9. Furthermore, even though layer stacking increases the effective visual fields of *Convolutional* layers, it obtained the increment at an increased computational cost. Moreover, in layer stacking, different kernels are not mixed. Consequently, there is no diversity among the features learned by a given stack. With these technicalities in mind, [60] proposed the Inception neural network architecture.

2.3.4.3 The Inception Network

The basic Inception architecture was proposed by researchers from Google and won the 2014 ILSVRC14 image classification challenge [60]. The Inception architecture aimed to address two core problems associated with deep neural networks. First, deep neural networks often have a large number of trainable parameters. Thus, if the number of training data is limited, they are very susceptible to overfitting. Moreover, the authors argued that using networks with a large number of trainable parameters would be unfeasible in some practical cases. In this regard, [60] argued that neural network architectures should not solely focus on shear numbers (higher accuracies). On the contrary, designs must also consider the increasing utilization trend of smaller computational devices such as smartphones and embedded systems that have limited computational resources [60]. In general, [60] argued that both problems could get solved by finding an optimal local architecture that is sparse in terms of trainable weights. Thus, this way, the local architecture can be spatially repeated to benefit from the advantages of deep neural networks. In this aspect, continuously stacking *Convolutional* layers would capture a certain aspect of an input feature. However, continuous stacking would significantly increase the number of multiplication and addition operations as the filter size and the number of layers increase. Thus, in such architectures, the overall computation would become high. Consequently, such networks will be expected to optimize for a large number of parameters which significantly contribute to overfitting. Moreover, if *Convolutional* layers get continuously stacked, it would in aggregate increase the receptive field of the networks, i.e., as the depth increases. Thus, in a sense, in a stacking approach, we would first focus on a segment of the input and then zoom on the segments as the network depth progresses. However, under such an approach, there is a possibility of missing certain aspects of an input.

On the contrary, if a given input feature gets analyzed by a set of parallel *Convolutional* layers, we could simultaneously capture and efficiently analyze a different aspect of the input. For instance, if certain input features are highly correlated and span a specific region of an input, a (2×2) *Convolutional* kernel would be effective enough for the analysis. Consequently, for inputs that are less correlated and dispersed over a wider region could be captured by a relatively higher receptive fields such as a (3×3) and (5×5) *Convolutional* kernels [29], [60]. In reality, such a parallel approach is not only efficient for feature extraction. It also reduces the number of computations (it has a sparse computational requirement). For instance, if we stack two (3×3, 64 channel) *Convolutional* layers, then the first *Convolutional* layer would perform 3×64 *Convolutional* computations. On the contrary, the second layer will have $3 \times 64 \times 64$ *Convolutional* computations. However, if we perform the two convolutions in parallel, we would only require 6×64 *Convolutional* computations. With these

understandings in mind, [60] proposed two types of Inception modules shown in Figure 2.27 which considered parallel concatenation of *Convolutional* layers.



(a) Novice Inception **(b)** Inception for dimension reduction

Figure 2.27: Basic Inception blocks [60]

In Figure 2.27 (a), [60] proposed to concatenate the outputs of a 5×5, 3×3, 1×1 *Convolutional* and *Maxpooling* kernels. In this concatenation, we can think of the layer that only has the 1×1 *Convolutional* kernels as the residue links in ResNet. Moreover, the authors also suggested that the cost of the 3×3 and 5×5 *Convolutional* kernels would quickly become expensive as more get concatenated and as the network goes deep. To this end, the author also proposed the Inception model shown in Figure 2.27 (b) that has dimensional reduction capability. In this aspect, the the 1×1 *Convolutional* kernels that preceded the 3×3 and 5×5 kernels served as as bottlenecks that reduced incoming dimension. Moreover, the 1×1 kernels also introduce additional nonlinearity often seen as an advantage [60]. With this modules at hand, [60] proposed the GoogleLeNet shown in Table 2.6. In the table, #3 $\times$ 3 and #5 $\times$ 5 reduce imply the dimension reduction performed at the 1×1 bottleneck *Convolutional* layers. Similar to its VGG16 counterpart, the GoogleLeNet was trained on datasets from the ImageNet competition. As a quick reminder, the datasets constituted 1.2 Million, 50,000, and 10,000 labeled images respectively used for training, validation, and testing. On these datasets, the GoogleLeNet obtained a top 5 validation error of 6.67%. This was better than the 7.2% top 5 error of VGG16 that obtained second place. This gets later surpassed by the ResNet (5.71%) in the 2015 ImageNet competition. However, parameter-wise, the ResNet is significantly larger than the GoogleLeNet to only achieve a 0.96% performance improvement.

In conclusion, in this chapter, we have presented concepts related to time series averaging. In this regard, we provided a summary of the pioneering averaging heuristics. Moreover, we have provided insights into the limitations and gaps we have observed with previous proposals. Additionally, since we aim to approach time series averaging as a generative problem, we proposed to base our approach on neural networks. To this end, in this chapter, we summarized concepts concerning neural networks with the main aim of clarifying the underlying reasons behind the choices made in the following two chapters. In this aspect, we have presented the underlying concepts behind the operation of neural network layers and their respective unique advantages. In this regard, we propose to mainly base our proposals on *Convolutional* layers due to their success in shape and feature analysis highly correlated to the shape preservation requirement of the time series averaging

Table 2.6: The GoogleLeNet architecture [60]

Type	Path size stride	#1 × 1	#3 × 3 reduce	#3 × 3	#5 × 5 reduce	#5 × 5	Pool proj.	Params	Ops
Covn.	7 × 7/2							2.7K	34M
MaxPooling	3 × 3/2							0	0
Conv.	3 × 3/1		64	192				112K	360M
MaxPooling	3 × 3/2							0	0
Inception		64	96	128	16	32	32	159K	128M
Inception		128	128	192	32	96	64	380K	304M
MaxPooling	3 × 3/2							0	0
Inception		192	96	208	16	48	64	364K	73M
Inception		160	112	224	24	64	64	437K	88M
Inception		128	128	256	24	64	64	463K	100M
Inception		112	144	288	32	64	64	580K	119M
Inception		256	160	320	32	128	128	840K	170M
MaxPooling	3 × 3/2							0	0
Inception		256	160	320	32	128	128	1072K	54M
Inception		384	192	384	48	128	128	1388K	71M
Avg. Pooling	7 × 7/1							0	0
Dropout (40%)	7 × 7/1							0	0
Dense								1000K	1M
Softmax								0	0

problem [6], [41], [56]–[58], [60]. Following the summaries of the available layers, we have presented concepts related to activation values and layer weight initialization techniques. We focused on these parameters of neural networks since they highly affect the type of features a network extracts. In this aspect, activation functions play a significant role in determining the type of transformations a neural network achieves. Additionally, layer weight initialization techniques impact how activation values propagate through the network [66], [67]. To this end, we expect these parameters to affect the performances of our proposals which mainly rely on latent features of neural networks. Thus, the parameters we selected in this regard gets influenced by the works we reviewed in this chapter, i.e., [57], [58], [60], [66], [67]. Finally, we also note that neural network architectural design is often a challenging and computationally demanding process that is by itself a broad research area [57], [58], [60]. With this understanding, in this chapter, we have presented some renowned neural network architectures which we have used as a basis for our proposals. However, since they get constructed to perform multi-class classification, we are expected to perform proper modifications to meet the demands of the task at hand. Moreover, while modifying the previous proposals, we have also kept the computational resource requirements in mind. With this said, we will conclude this chapter and proceed to present our proposed approaches.

3 Time Series Averages from the Latent Space of Basic and Variational Autoencoders

In all pioneering time series averaging techniques, time series averaging gets often approached as a multiple alignment problem. To this end, all proposed heuristics rely on different time-warping techniques. Moreover, the quality of the estimated averages highly correlates with the performance of the warping technique. This is because, in time series averaging, we desire estimated averages to minimize their discrepancy to the individual members of an averaged set. To meet this objective, all proposed heuristics transform (warp) the original series into an alternative space [14]–[16], [25]. However, mathematically, we expect the transformed (warped) series to be significantly different from the original series. In this aspect, if we assume the original series follows some multivariate distribution, for instance, a Gaussian distribution $\mathcal{N}(\mu,\ \Sigma)$. Where in this case, $\mu\ \in\ \mathbb{R}^M$ and Σ are the mean and covariance matrix of the distribution: then, the transformed (warped) series can also get expressed using a Gaussian distribution of the form $\mathcal{N}(\mu_\tau, \Sigma_\tau)\ :\ \mu_\tau\ \in \mathbb{R}^\tau\ for\ \tau\ \geq\ M$. However, we expect the transformation (warping) to have at least two major effects. The first and obvious difference would be the dimension difference between the two distributions. In addition to this, if we, for simplicity, assume the two Gaussian distributions are in a two-dimensional space, then their two-dimensional plots will have an elliptical shape. However, in the context of time series averaging, the transformation (warping) will stretch the major and minor axis of $\mathcal{N}(\mu_\tau, \Sigma_\tau)$ since it maximizes the correlation among warped series. In other words, we expect the transformed series to be dense (compact) compared to their original counterparts. In reality, this fact gets implicitly expressed in the objective function (1.1) presented in chapter two, where it tries to minimize the discrepancy between the transformed series by aligning them to a common landmark or their τ space arithmetic mean. In terms of Gaussian distribution, this would mean a narrow distribution curve. Besides this observation, we can also go ahead and further correlate the arithmetic averaging of the transformed series to a maximum likelihood estimation. This is because, given a set of K Gaussian multivariate variables, $\mathbf{Y} = \{X_1,\ X_2, \ldots,\ X_K\}\ :\ Y_i\ \epsilon\ \mathbb{R}^\tau$, the parameters of a multivariate Gaussian distribution that maximizes their likelihood is computed as (3.1):

$$\begin{aligned} \mu_\tau &= \frac{1}{K}\sum_{i=1}^{K} x_i \\ \Sigma_\tau &= \frac{1}{K}\sum_{i=1}^{K} (Y_i - \mu_\tau)^T (Y_i - \mu_\tau) \end{aligned} \tag{3.1}$$

Thus, while estimating a mean in $\mathbb{R}^\tau$, we can safely assume that we are making a maximum likelihood estimation using a narrow Gaussian distribution curve. If we pause at this point and see the overall process from a different perspective, we can assume we are performing a very constrained augmentation process. We say constrained augmentation for two main reasons. First, the augmentation gets conducted in a domain different from the original series. Moreover, unlike most common augmen-

tation techniques [40], we direct the augmentation space in a way that encourages μ_τ to meet the requirements of (1.1). However, in reality, meeting the requirements of (1.1) is vague in the context of most previously proposed averaging techniques. This is because, in all the proposed techniques, (1.1) gets minimized in $\mathbb{R}^\tau$ rather than $\mathbb{R}^M$. Thus, proposed averaging techniques guarantee the quality of the estimated average in $\mathbb{R}^\tau$ rather than $\mathbb{R}^M$. To this end, in most cases, the estimates are often paired with the underlying warping techniques either when their quality gets assessed or practically utilized [16], [25]. In this regard, DTW based estimated averages get often paired with DTW distance. On the contrary, DTAN requires the affine transformation of unseen datasets while evaluating the quality of its estimated means.

With these observations in mind, we asked ourselves, can we approach time series averaging as an augmentation problem? If so, we then ask, how can we identify or define a proper augmentation space? We answer the former question empirically. On the contrary, to answer the latter questions, we place the following constraints on the augmentation space:

- First, we desire to mimic the effects of multiple alignment (warping) in the augmentation space. This way, the augmentation space representation of the input time series gets confined to smaller regions of the augmentation space. This in turn is expected to increase the interpretability and representativeness of augmented means in the context of a re-transformer.
- Second, we also desire the augmentation space to be invertible, i.e., we want a mean estimated in the augmentation space to have a time domain representation. This way, we can generate a time domain equivalent that is meaningful for data mining techniques relying on time domain estimates.

In reality, the first constraint can either be advantageous or disadvantageous. On the good side, by defining compact (dense) transformed representations, we minimize the impact of phase distortion evident in the time domain. Moreover, a compact representation will constrain the averaged set into a small region that minimizes the search space of an augmented mean. However, from a re-transformer point of view, given representations from very dense augmentation space, it could be challenging to uniquely identify individual members of an averaged set. To this end, since a mean by itself is inherently a member of the averaged set, an interpreter could map the augmentation space estimated averages to one of the averaged series.

With these technicalities in mind, we propose to augment time series averages from the latent space of neural networks. We aim to base our approach on neural networks since they have a better generalization capability. To this end, it is possible to utilize transfer learning while updating estimates as more datasets become available. Additionally, in recent years, neural networks got shown to be effective at generating synthetic datasets that preserve basic features (shapes) observed in the inputs [38], [39]. This is in line with the objectives of time series averaging that aims to preserve shapes observed in an averaged set, i.e., (1.1). With this said, the next feasible question would be, what should the organization of the augmentation network be? In the context of the second constraint, i.e., invertibility of the augmentation space, one logical choice is an autoencoder. In practice, a basic

autoencoder is a neural network architecture with two sub-architectures, i.e., the encoder and decoder. On one hand, an encoder gets used to extract a lower dimensional representation of inputs, where the lower dimensional representations are often called embedding or latent space representations. On the other hand, the decoder tries to reconstruct the latent space representations to their time domain format, i.e., with the minimum possible reconstruction error [76]. However, in practice, an autoencoder can be designed to perform complicated tasks besides simple encoding and decoding. In this regard, [38] showed that autoencoders can learn a prior distribution that gets later sampled to generate meaningful variants of the input datasets. Moreover, in [39], a structure resembling an autoencoder is used to generate synthetic datasets that highly resemble its inputs. These abilities of autoencoders are in line with our primary objective, i.e., times series average augmentation. Thus, we will first empirically assess the possibility of augmenting optimal time series averages from the latent space of basic autoencoders. We then present the different modifications made to improve the quality of the estimates. However, before diving into the details of our proposals, we first give a description of the datasets used in all of our experimental evaluations.

3.1 Evaluation Datasets from the UCR Archive

In this dissertation, we mainly emphasize the estimation of an optimal average for univariate temporal datasets with equal length. Given these circumstances, we identified the University of California Univariate Time Series Repository (UCR) [2] as one possible candidate for our experimental evaluations. In practice, the datasets from the UCR got intensively utilized in the evaluation of neural network and distance-based time series classification, clustering, and augmentation tasks [7], [13], [40], [56], [61]. The UCR is composed of 128 univariate temporal datasets that span a range of application domains. However, among the 128 datasets, only 114 of them meet our requirement of fixed-length univariate temporal datasets. In this regard, we identified 11 datasets of the UCR to have variable in lengths, i.e., AllGestureWiimote$\{X,\ Y,\ Z\}$, GestureMidAir$\{D1,\ D2,\ D3\}$, GesturePebble$\{Z1,\ Z2\}$, PickupGestureWiimoteZ, PLAID and ShakeGestureWiimoteZ. Additionally, we also identified three datasets to have missing values: *DodgerLoopDay*, *DodgerLoopGame*, *DodgerLoopWeekend*. Besides these irregularities, the UCR datasets gets organized as a train and test split. Moreover, each dataset contains samples belonging to multiple classes (C) that could range from $2 \leq C \leq 60$. In general, Table 3.1 demonstrates the diversity of the UCR datasets. However, even if the UCR is diverse in the context of application domains, in most cases, the number of training per class sample is limited. For instance, the *DiatomSizeReduction* dataset has a single example in one of its classes. In practice, such limited per-class examples could be a major challenge in the context of neural network generalization capability. For instance, if we do not carefully control the size of network parameters, they could easily overfit the training samples. With this said, we will present some demonstrative examples from the categories given in Table 3.1. We believe this will assist the reader to better understand and appreciate the datasets.

Table 3.1: The 114 UCR datasets categorized based on their source

No.	Data source	Total datasets	Dimension ranges	Class ranges
1	Device power consumption measurements	9	90-2000	2-10
2	Bio-potential measurements	10	96-1250	2-42
3	Hemodynamics measurements	3	2000	52
4	HRM-PCR measurements	1	201	18
5	Images	32	46-2709	2-60
6	Motions	17	150-1882	2-12
7	Sensor measurements	20	24-1639	2-39
8	Synthetic (simulated)	8	60-1024	2-8
9	spectrographs or chemical analysis	8	235-1751	2-5
10	SEMG measurements	4	1500-2844	2-6
11	Pedestrian Traffic count	2	24	2-10

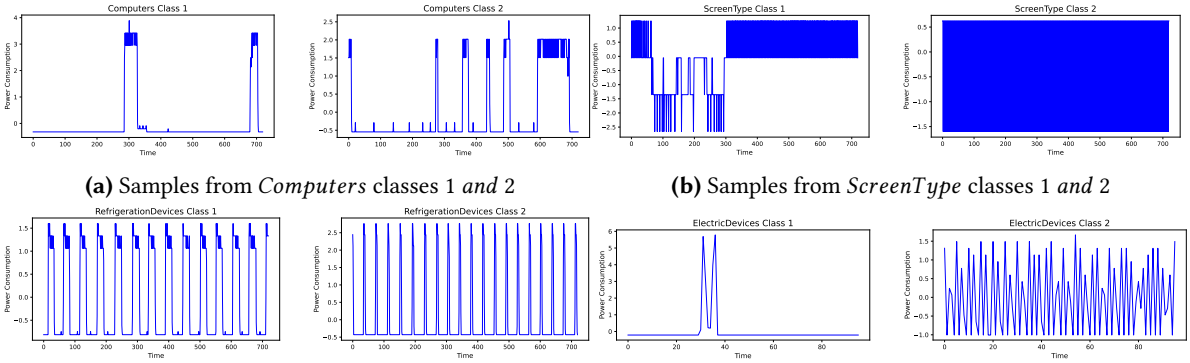
3.1.1 Time Series Extracted from Devices Power Consumption Measurements

The UCR archive has nine datasets that could be categorized as *"Household devices power consumption measurements"* as shown in Table 3.2. Most of the Datasets falling within this category were contributed from a study sponsored by the government of the United Kingdom (UK). The theme of the study was titled *Powering the Nation* [2], [4], [77]. In reality, six out of the nine datasets within this category correspond to the study. Generally, the study focused on understanding how consumers utilize electricity within their homes. Moreover, it also aimed to create awareness of reducing the carbon footprint of the UK by 80% in 2050. To meet this objective, the government of the UK installed a power consumption meter in residential areas at the cost of 11.1 billion euros [4], [77]. The installed

Table 3.2: UCR archive datasets falling within the Device and Power consumption category [2], [77]–[79]

Datasets	Classes	Length	Categories (Classes)
ACSF1	10	1460	Mobile phones (via chargers), Coffee machines, Computer stations (including monitor), Fridges and freezers, Hi-Fi systems (CD players), Lamp (CFL), Laptops (via chargers), Microwave ovens, Printers, and Televisions (LCD or LED)
Computers	2	720	Desktop and Laptop
ElectricDevices	7	96	Computer, Oven/Cooker, Washing Machine, Immersion Heater, Dishwasher, Fridge/Freezer, Kettle
HouseTwenty	2	2000	Aggregate household power consumption, Aggregate Tumble Dryer and Washing Machine power consumption.
LargeKitchenAppliances	3	720	Dishwasher, Tumble Dryer and Washing Machine
PowerCons	2	144	Power consumption in warm weather, Power consumption in cold weather
RefrigerationDevices	3	720	Fridge/Freezer, Refrigerator and Upright Freezer
ScreenType	3	720	CRT TV, LCD TV and Computer Monitor
SmallKitchenAppliances	3	720	Kettle, Microwave and Toaster

meters had the ability to show real-time power measurements of home appliances. However, in the UCR, most of the datasets get defined by taking readings from 251 households every 2 minutes for 24 hours, i.e., each series is 720 timestamps long [2]. Figure 3.1 depicts some of the time series extracted from the study.



(a) Samples from *Computers* classes 1 *and* 2

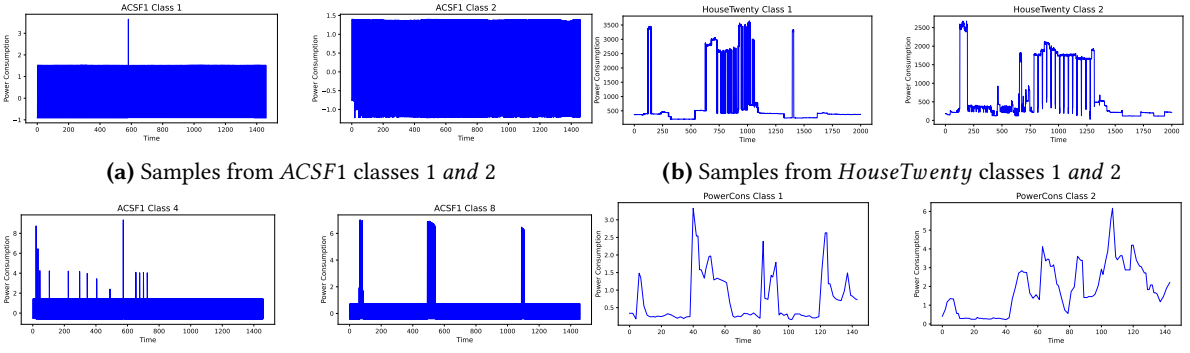
(b) Samples from *ScreenType* classes 1 *and* 2

(c) Samples from *RefrigerationDevices* classes 1 *and* 2

(d) Samples from *ElectricDevices* classes 1 *and* 2

Figure 3.1: Example time series from the UCR Device Category: *Computers* (a), *ScreenType* (b), *RefrigerationDevices* (c) and *ElectricDevices* (d). These Datasets were defined from a study sponsored by the UK government namely *Powering the Nation* which assessed the power consumption of home appliances [77]

In addition to the datasets extracted from the *Powering the Nation* study, the UCR archive also has datasets that are extracted from two different but similar studies. In this context, the *HouseTwenty* dataset was extracted from the project *Personalised Retrofit Decision Support Tools for UK Homes using Smart Home Technology (REFIT)* [78]. The project intended to assess the aggregate power consumption of home appliances in the UK residential area of Loughborough. The first class of the *HouseTwenty* dataset corresponds to the aggregate power consumption of 20 houses located within the study area. However, the second class corresponds to the aggregate electrical load of tumble dryers and washing machines [78]. On the contrary, another study intended to identify home appliances using their



(a) Samples from *ACSF*1 classes 1 *and* 2

(b) Samples from *HouseTwenty* classes 1 *and* 2

(c) Samples from *ACSF*1 classes 4 *and* 8

(d) Samples from *PowerCons* classes 1 *and* 2

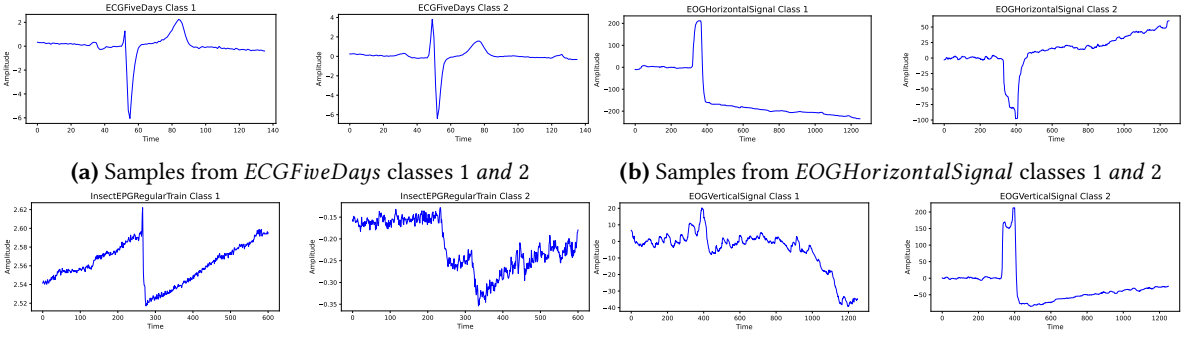
Figure 3.2: Example time series from the *ASCF*1 (a) & (c), *HouseTwenty* (b) and *PowerCons* (d)

power consumption signature rather than studying aggregate power demands [79]. The datasets defined in this study contain the power consumption signature of typical home appliances such as coffee makers, CD players, microwave ovens, etc. Finally, we also found the *PowerCons* dataset to correspond to the device category. The dataset contained the power consumption measurements of

individual households for two different seasons that were generalized as warm and cold. In conclusion, Figure 3.2 demonstrates samples from the *ACSF1*, *HouseTwenty* and *PowerCons* datasets.

3.1.2 Time Series Extracted from Bio-potential Measurements

Besides power consumption measurements, the UCR repository includes time series defined from measurements taken by medical equipment. In this regard, the UCR has ten datasets corresponding to this category. Furthermore, six of the ten datasets correspond to ElectroCardioGram (ECG) measurements. Most of the datasets in this category correspond to studies emphasizing either the autonomous detection of normal and abnormal heartbeats in a fetus (adults) adults [2], [80]–[83] or signal processing on heartbeat bio-potentials [83]. In addition to this, the UCR archive also has two ElectroOculoGram (EOG) measurements that correspond to the potential difference between the retina and the cornea of a human eye [2], [84]. These measurements get taken to assist patients with locked-in syndrome in Eye-writing systems. In practice, an Eye-writing system displays a character corresponding to a line of strokes traced by the eye movement of its user. To identify the various types of stokes, four electrodes got placed at the left, right, top, and bottom of a test subject's left eye. Following this, a two-channel vertical/Horizontal signal was registered by taking the potential difference between the two up/bottom and left/right electrodes [84]. Finally, from different Japanese *Katakana* strokes, 12 classes were introduced that corresponded to a stroke ID [2]. The final dataset corresponding to UCR's human bio-potential includes measurements made with Surface ElectroMyoGraphy (sEMG). A sEMG signal represents the electrical activity of a group of muscles at rest or in movement [2]. The UCR extracted the datasets corresponding to such measurements from [85]. In [85], sEMG measurements from five healthy subjects conducting six movements of hand grasps got taken using elsys' 2-channel EMG system.



(a) Samples from *ECGFiveDays* classes 1 *and* 2

(b) Samples from *EOGHorizontalSignal* classes 1 *and* 2

(c) Samples from *InsectEPGReguralTrain* classes 1 *and* 2

(d) Samples from *EOGVerticalSignal* classes 1 *and* 2

Figure 3.3: Sample time series from the UCR archive that fall within the bio-potential category

Besides human Bio-potential measurements, the UCR archive also contains two datasets used to study the feeding behavior of insects. For instance, the UCR has an Electrical Penetration Graph (EPG) measurements that monitor the voltage changes in an electrical circuit that connects Asian citrus psyllid and its food source [2], [86]. In practice, the Asian citrus psyllid is known to be a source of pathogens causing the citrus greening disease. To this end, researchers study the sequence of their feeding behaviors to understand how the insects acquire and transmit the pathogens. In the study, the researchers first tethered the insects to a gold wire attached to an electrode. To complete the

electrical circuit, they placed another electrode in moist soil at the base of a plant. Finally, they record the voltage difference generated when the insect starts ingesting the plant [86]. In the study, the researchers identified the psyllid has six different feeding states; i.e., C (stylet passage through plant cell), D (contact with phloem tissue), E1 and E2 (phloem salvation and ingestion), G (Xylem ingestion), NP (non-probing). However, in the UCR archive, the six feeding states were aggregated into three classes for unclear reasons. In conclusion, we have summarized the datasets belonging to the different bio-potential measurements in Table 3.3. Moreover, we have also given sample plots corresponding to the various bio-potential measurements in Figure 3.3.

Table 3.3: UCR archive datasets falling within the bio-potential measurements category: *ECG [80]–[83], EOG [84], sEMG [85] and EPG [86]*

Datasets	Category	Classes	Length	Class Interpretation
ECG200	ECG	2	96	Normal heartbeat, Myocardial Infarction
ECG5000		5	140	Normal, R-on-T Premature Ventricular Contraction, Ectopic beat, Premature Ventricular Contraction, Unclassifiable beat
ECGFiveDays		2	136	ECG date: 12/11/1990, ECG date: 17/11/1990
NonInvasiveFetalECGThorax1		42	750	Class information not known
NonInvasiveFetalECGThorax2		42	750	Class information not known
TwoLeadECG		2	82	Signal 0 (unfiltered ECG), Signal 1 (filtered ECG)
EOGHorizontalSignal	EOG	12	1250	12 Japanese Katakana eye strokes measured with horizontal EOG electrodes
EOGVerticalSignal		12	1250	12 Japanese Katakana eye strokes measured with horizontal EOG electrodes
SemgHandGenderCh2	sEMG	2	1500	sEMG of Male/Female
SemgHandMovementCh2		6	1500	sEMG of 6 hand grip movements
SemgHandSubjectCh2		5	1500	sEMG of 5 Males/Females
InsectEPGRegularTrain	EPG	3	601	Psyllid feeding pattern
InsectEPGSmallTrain		3	601	Psyllid feeding pattern

3.1.3 Time Series Extracted from Sensor Measurements

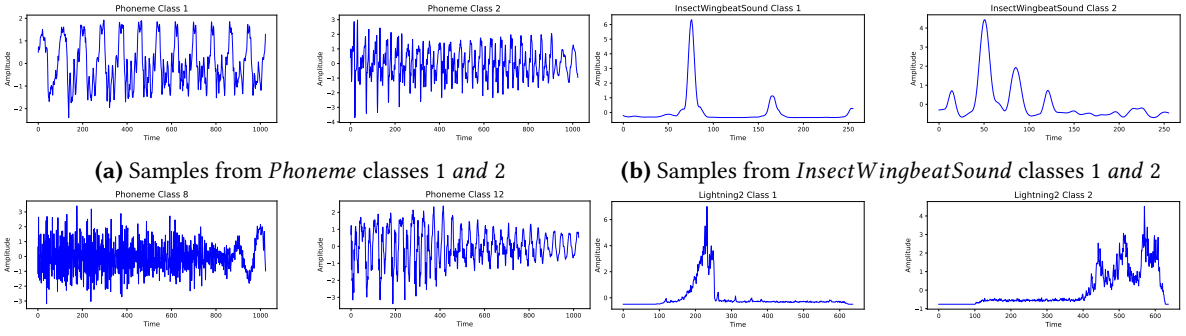
Most practical applications utilize sensors to take measurements for monitoring, recording and analyzing natural and artificial phenomena. In this aspect, the UCR archive has 20 univariate temporal datasets corresponding to sensor measurements in various contexts. For instance, the UCR archive has two datasets used in [87] to analyze the power spectral density of various forms of lightning. The study took the potential measurement of lightning using *FORTE* satellites that can detect transient electromagnetic events associated with lightning using a suite of optical and radio-frequency (RF) instruments. In practice, lightning is a sensitive indicator of storm evolution. Moreover, it gets

Table 3.4: UCR archive datasets falling within sensor measurement category [2], [87]–[89]

Datasets	Classes	Length
Car	Four classes: no class information	577
ChlorineConcentration	Three classes: no class information	166
CinCECGtorso	ECG data for multiple torso-surface sites of 4 people	1639
DodgerLoopDay, DodgerLoopGame, DodgerLoopWeekend	Freeway traffic count from Monday to Sunday Freeway traffic count on Normal & Game Days Freeway traffic count on Weekdays & Weekend	288
Chinatown MelbournePedestrian	Pedestrian traffic count in Melbourne Pedestrian traffic count in Chinatown	24 20
Earthquakes	Earthquake or no earthquake	512
Ford {A, B}	Two classes: no class information	500
FreezerRegularTrain, FreezerSmallTrain	Power demand of fridges //placed in a Kitchen or in a garage	301
InsectWingbeatSound	Insect wing-beat sounds of male/female mosquitoes: Aegypti, Cx. tarsalis, Cx. quinquefasciants, Cx. stigmatosoma, flies: Musca domestica and Drosophila simulans	256
ItalyPowerDemand	power consumption: Oct to March & April to September	24
Lightning2 Lightning7	Two classes: no class information CG, IR, SR, I, I2, KM, O	637 319
MoteStrain	Measurments from humidity & temperature sensors	84
Phoneme	39 different phonemes	1024
Planes	Outlines of Mirage, EuroFighter, F-14 wings closed, F-14 Wings open,Harrier, F-22, F-15	144
SonyAIBORobotSurface1 SonyAIBORobotSurface2	Robot moving on a surface: Cement & carpet Robot moving on a surface: Cement & carpet/field	70 65
StarLightCurves	Three classes: Celestial object brightness Vs time.	1024
Traces	Four classes: simulated instrumentation failures in a nuclear power plant	275
Wafer	Normal/abnormal process control measurements during the processing of silicon wafers	152

associated with severe weather such as tornadoes [87]. To this end, *FORTE* utilizes 2 broadband Very High Frequency (VHF) receivers that can have a 22 MHz sub-band within the 30-300 MHz frequency ranges. Moreover, a given sub-band could get configured to have eight 1 MHz channels. In reality, a measurement gets recorded if five of the eight sub-channels measure a voltage level above a predefined threshold. In such scenarios, data gets recorded for 800 μs at a sampling rate of 1 MHz. Following these measurements, the received signals were Fourier transformed to generate frequency spectrograms later collapsed in frequency to generate a power spectral density. In [87], the spectral densities initially had a dimension of 3181, but they got smoothed out to 631 [2]. Moreover, [87] categorized the measurement into sever different lightning types: Positive Initial Return Stroke (CG), Negative Initial Return Stroke (IR), Subsequent Negative Return Stroke (SR), Impulsive Event (I), Impulsive Event Pair (I2), Gradual Intra-Cloud Stroke (KM), and Off-record (O). In reality, UCR archive datasets

falling within the sensor category are not limited to lightning measurements. For instance, the UCR archive also has datasets corresponding to sensor measurement of insect wing-beat sound. In this regard, the UCR archive has one dataset that contains the measurements of insect wing-beat sounds for four species of male/female mosquitoes and two species of flies [2], [88]. In reality, most of these measurements gets taken to monitor the movements of mosquitoes [88]. In addition to insect wing-beat sounds, the URC archive has datasets corresponding to pedestrian and vehicle traffic counts. The datasets get defined by taking the count of pedestrians or vehicles on highways in different parts of the world and on different days of the week [2]. In another domain, there are also datasets corresponding to segmented audio recordings collected from Google Translate, "oxforddictionaries.com" and the Merriam-Webster online dictionary [89]. These segmented sounds contain male and female speakers later categorized into 39 different phonemes for further studies. Generally speaking, the UCR archive has more than 22 such sensor measurements. To this end, it would consume too much space to individually discuss how each dataset gets extracted. We encourage interested readers to refer to the time series classification web page for further information [2]. However, to make our discussion complete we have summarized the datasets falling in the sensor category in Table 3.4. Moreover, we have plotted samples from the *Lightning*, *InsectWingbeatSound* and *Phoneme* datasets in Figure 3.4 as demonstrative examples.



(a) Samples from *Phoneme* classes 1 *and* 2

(b) Samples from *InsectWingbeatSound* classes 1 *and* 2

(c) Samples from *Phoneme* classes 8 *and* 12

(d) Samples from *Lightning2* classes 1 *and* 2

Figure 3.4: Sample UCR archive time series that are defined from sensor measurements

3.1.4 Time Series Extracted from Images, Motion and Gestures

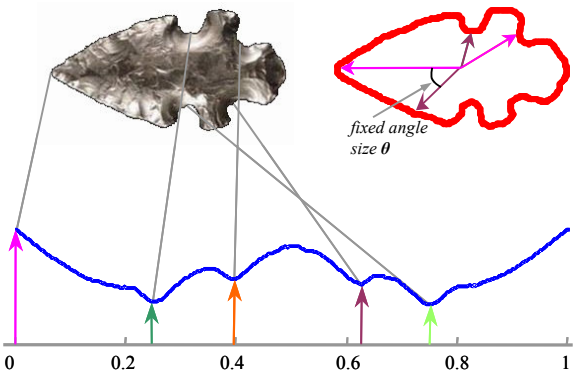
Time series are not only defined from the sensor measurements of biological or non-biological signals. On the contrary, temporal datasets could be defined from a range of application domains if the right conversion technique gets utilized. For instance, in the UCR archive, there are time series extracted from images. This group of temporal datasets gets often defined by taking a distance measurement between a reference point and points on the boundaries that enclose a biological or non-biological object's image [4]. In most cases, the reference gets taken as the central point within the boundaries of the analyzed object's image. Moreover, the amplitude of the extracted temporal datasets gets taken to be the measured distance difference. On the contrary, the timestamps gets defined by taking the order of distance measurement into account. In practice, this conversion process is performed on segmented images to simplify the boundary (edge) detection process. In the UCR archive, some of the demonstrative examples in this regard are the *BeeteleFly*, *ChickenBird*, *ADIAC*, *ArrowHead*, etc [4],

[90], [91]. If we, for instance, consider the *ArrowHead* dataset as a demonstrative case, it gets defined

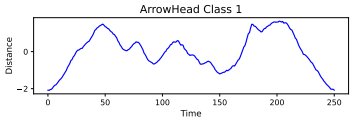
(a) *Avonlea* arrow head images **(b)** *Clovis* arrow head images **(c)** Arrow heads with *Mixed* shapes

Figure 3.5: Different shapes of ancient stone arrow heads [91]

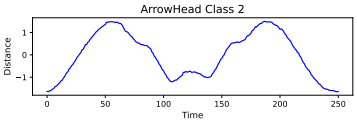
using a set of stone arrowhead images corresponding to different ancient civilizations. In anthropology, such arrowheads are categorized depending on their shape, discovery site, etc. For instance, in [91], arrowheads were grouped based on their shapes as *Avonela*, *Clovis* and *Mixed* as shown in Figure 3.5. In [91], the authors aimed to classify the images using a rotational invariant one-dimensional time series shapelet (most descriptive shape) extracted from the images of arrowheads. However, before the classification task, the authors had to convert the images into a one-dimensional temporal series. In this regard, the authors proposed to utilize the angle-based time series extraction technique. In



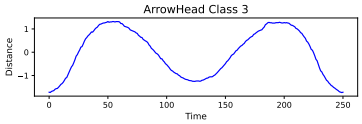
(a) Angle based time series extraction from an image of an arrowhead



(b) A sample time series extracted from an image of an *Avonlea* arrow head



(c) A sample time series extracted from an image of a *Clovis* arrow head



(d) A sample time series extracted from an image of an arrow head that has mixed shape

Figure 3.6: Angular-based time series extraction from the images of ancient stone arrow heads [91]

the extraction process, the processed images get first scaled to have similar dimensions. Following this, a central reference point within the boundaries of the arrowheads gets selected. Finally, a set of distance measurements gets taken between the reference point and the edges of the arrowheads. In reality, the measurement is taken at a fixed angular step value either in a clockwise or counter-clockwise manner. Thus, in the end, the distance values define the amplitude of the extracted time series, whereas the angle of rotation defines the timestamps of the one-dimensional temporal series. Figure 3.5 summarizes the angular-based time series extraction process. However, one additional

point to mention here is that angular-based conversions are sensitive to rotation, i.e., a rotation of an image will shift the extracted series along the time axis. In this aspect, the authors proposed to concatenate an extracted shapelet with its copy so that it becomes rotation invariant [91].

Besides the temporal datasets extracted from images, the UCR archive also contains datasets representing the movement and gestures made by human beings and earthworms while performing different tasks. For instance, the UCR's $Cricket\{X,\ Y,\ Z\}$ archive gets defined from three-dimensional accelerometer measurements taken from test subjects playing cricket [92]. The three-dimensional accelerometer measurements get used to identifying one of the 12 gestures in cricket: Cancel Call, Dead Ball, Four, Last Hour Leg Bye, No Ball, One Short, Out, Penalty Runs, Six, TV Replay, and Wide [2], [92]. In reality, the UCR archive contains a range of datasets extracted from movements and gestures that might differ in how they get extracted. For instance, the *Haptic* dataset gets extracted by recording the X-axis movements of people entering a passgraph, i.e., a code to assess a system protected by a graphical authentication system [2]. In summary, we have given the list of the UCR archive datasets extracted from images, movements, and gestures in Table 3.5.

Table 3.5: UCR archive datasets that are defined from images, movements and gestures [2], [91]

Datasets	Extracted from
Adiac, ArrowHead, BeetleFly, BirdChicken , DiatomSizeReduction, DistalPhalanxOutlineAgeGroup, DistalPhalanxOutlineCorrect, DistalPhalanxTW, FaceAll, FaceFour, FacesUCR, FiftyWords, Fish, HandOutlines, Herring, MedicalImages, MiddlePhalanxOutlineAgeGroup, MiddlePhalanxOutlineCorrect, MiddlePhalanxTW, OSULeaf, PhalangesOutlinesCorrect, ProximalPhalanxOutlineAgeGroup, ProximalPhalanxOutlineCorrect, ProximalPhalanxTW, ShapesAll, SwedishLeaf, Symbols, WordSynonyms, Yoga, Crop, MixedShapesRegularTrain, MixedShapesSmallTrain	Images
CricketX, CricketY, CricketZ, GunPoint, Haptics, InlineSkate, ToeSegmentation1, ToeSegmentation2, UWaveGestureLibraryAll, UWaveGestureLibraryX, UWaveGestureLibraryY, UWaveGestureLibraryZ, Worms, WormsTwoClass, GunPointAgeSpan, GunPointMaleVersusFemale, GunPointOldVersusYoung	Motion and gestures of humans and earth worm

3.1.5 Time Series Extracted from Simulations, Spectrography, Hemodynamics and High Resolution Melting Point Measurements

The final subgroup of datasets we found in the UCR archive is obtained from simulations (synthetic data), food quality spectrograph measurements, biological or non-biological object's melting point radiation spectrum, fluid pressure measurements, and from luminescence measurements of stones [2], [93]–[97]. To give a general picture of these datasets, we will present a brief description of some of the datasets from each category. For instance, we can consider the *SmoothSubSpace* dataset from the simulated category [98]. This dataset was used in [98] to evaluate if a clustering algorithm can identify smooth subspaces while clustering time series. In this context, the *SmoothSubSpace* dataset contains three different classes that correspond to a continuous subspace spanning five timestamp

Table 3.6: UCR archive datasets corresponding to simulation, spectrograph, hermodynamics and HRM measurements [2], [94]–[96]

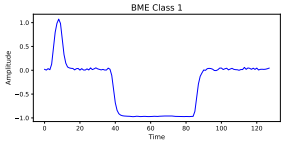
Datasets	category	classes	Length	Class Information
BME	Simulated	3	150	Bell Shape: Begin, Middle, End
CBF		3	128	Cylinder, Bell, Funnel
Mallat		8	1024	No class information
ShapeletSim		2	500	Shapes corrupted by noise
SmoothSubspace		3	15	Segments of 3 continuous sub space
SyntheticControl		6	60	Control charts:Normal, Cyclic, Increasing trend Decreasing trend, Upward shift, Downward shift
TwoPatterns		4	128	Patterns from decision tree: down-down, down-up, up-up
UMD		3	150	Bell Shape: Begin, Middle, End
Beef	Spectrograph	5	470	Beef: Pure, adulterated with; heart, tripe, kidney and liver
Coffee		2	286	Coffe beans:Arabica, Robusta
EthanolLevel		4	1751	Ethanol: E35, E38, E40, E45
Ham		2	431	Spanish & French dry-cured hams
Meat		3	448	Chicken, P and Turkey
OliveOil		4	570	Olive oil from alternative countries
Strawberry		2	235	strawberries and adulterated strawberries and other fruits
Rock		4	2844	Marfic , quartzite, marble, schist
Wine		2	234	No class information
Fungi	HRM	18	201	18 species of Fungi
PigAirwayPressure	Hermodynamics	52	2000	52 pigs airway pressure
PigArtPressure		52	2000	52 pigs arteries pressure
PigCVP		52	2000	52 pigs CV pressure

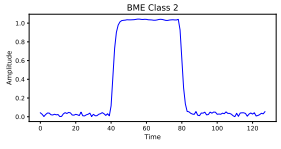
values. In reality, class one corresponds to a subspace spanning the timestamps 1-5, whereas clusters 2 and 3 correspond to the timestamp ranges of 6-10 and 11-15. Moreover, in all classes, the segment that did not correspond to a smooth subspace was filled by randomly generated values, where the *SmoothSubSpace* dataset is 15 timestamps long.

In another category, i.e., in the spectrograph category, time series were extracted from real-world measurements. For instance, the UCR archive's *Beef* dataset was defined from the spectrograph of different kinds of beef [94]. In practice, most foods are composed of various minerals and water. Thus, when they get bombarded with light rays, such as a Mid-infrared frequency light ray, the reflected light differs in wavelength and magnitude (intensity) depending on the contents of the food [94]. In [94], this concept was used to assess the quality of different types of beef. To make the assessment, the authors measured the reflected mid-frequency infrared light on five different variants of beef: pure beef and beef adulterated with heart, tripe, kidney, and liver. In reality, in the UCR archive, there are five additional datasets corresponding to similar measurements for different kinds of meats, strawberries, and coffee beans. In addition to these food spectrographs, we have

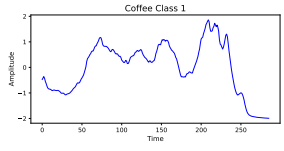
datasets corresponding to high-resolution melting measurements. In this aspect, we can consider the UCR's *Fungi* dataset as an example. The *Fungi* dataset was introduced in [95] by measuring the intensity of the light spectrum emitted by melting fungi. The measurements get later used to classify 18 different fungal species [95]. In practice, different species emit different sets of lights when exposed to high-temperature values, i.e., depending on their genome sequences. In molecular biology, such analysis of species is known as High Resolution Melting (HRM) point analysis. Finally, in the UCR archive, we have datasets corresponding to blood flow (Hemodynamics) measurements. In this regard, the UCR archive contains datasets extracted from [96] aimed to improve the detection time of internal bleeding. To devise a mechanism that improves the detection rate, the authors of [96] studied pressure changes in the airway, arteries, and Central Venous (CV) of 52 pigs before and after deliberately introducing internal bleeding. In the study, the 52 pigs got initially sedated while the measurement equipment was left to rest for 20 minutes. Following this, the pigs were slowly bled at a rate of 20 ml/min while vital sign measurements got taken using a bed-side hemodynamic monitoring system. Finally, two 30-second vital sign samples got taken, i.e., one before internal bleeding and another within 2 minutes after the internal bleeding had started. These samples got later used in studies that aimed to improve the detection time of internal bleeding to be between 10 to 15 minute [96].

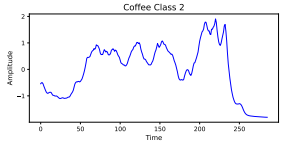
In general, we have summarized these practical and simulated UCR archive datasets in Table 3.6. Moreover, as their counterparts, we have also given sample time series from each subcategory in Figure 3.7. However, the reader must note that our overall discussion of the UCR archive datasets is a higher-level overview. An interested reader can further refer to the sources of each dataset from the 2018 UCR Time Series Archive web page (https://www.cs.ucr.edu/~eamonn/time_series_data_2018/) or from the time series classification web page (https://timeseriesclassification.com/dataset.php). With this said, we will conclude this section and present our approaches whose evaluations are based on the datasets presented in this section.



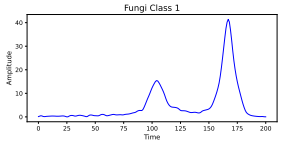


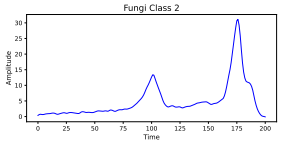
(a) Sample time series form *BME* classes 1 *and* 2



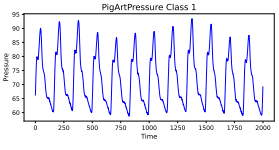


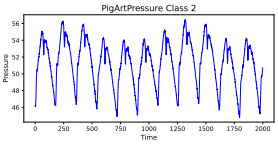
(b) Sample time series from $Coffee$ classes 1 *and* 2





(c) Sample time series from *Fungi* classes 1 *and* 2





(d) Sample time series from *PigArtPressure* classes 1 & 2

Figure 3.7: Sample UCR archive datasets that are extracted from simulation [2], spectrograph [2], [13], hermodynamics [2], [96] and HRM measurements datasets [2], [95]

3.2 Time series Averages from the Latent Space of a Basic Autoencoder

In practice, machine learning algorithms and neural networks obtain superior performances by relying on a lower dimensional abstraction of their inputs [7], [29]. For instance, if we take neural networks, the overall decision process is often based on the outputs of hidden units (layers) processing inputs in a manner that favors positive or negative outcomes. To this end, for a neural network to be intelligent, hidden units are expected to abstract a range of inputs by identifying common regularities that make up the negative and positive outcomes [99]. With this understanding, researchers often emphasized devising data abstraction techniques that could assist machine learning algorithms to focus on relevant information [99], [100]. However, until the introduction of autoencoders, there was no intelligent way of learning such lower dimensional data abstractions that could serve as an input to various learning algorithms [99]. In practice, the basic autoencoder is often built from two symmetrical neural networks, i.e., an encoder and decoder, which coherently work in an unsupervised manner to reconstruct an input from its lower dimensional representation. In practice, the lower dimensional representations (input data abstractions) are often known as the latent space representations (embedding) of an input [59]. In the context of an autoencoder, the encoder gets tasked with extracting the most descriptive lower dimensional representation of inputs. To meet this objective, an encoder gets constructed from at least two layers that consecutively decrease the dimension of an input [59], [76], [101]. The decoder often utilizes the same set of layers in a reversed direction to reconstruct an input from its latent space representation with the minimum possible reconstruction error. Thus, this way, a basic autoencoder is able to learn the dominant features that make up an input which can later be reconstructed to re-generate an input with a minimum loss of information. In general, the basic autoencoder is often generalized using the block diagram shown in Figure 3.8, where X_i can either be a series in $\mathbb{R}^M$, an image, or in general, an N dimensional matrix.

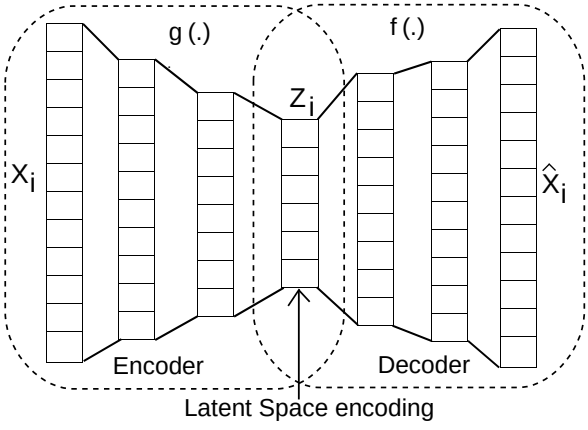


Figure 3.8: Block diagram of a basic autoencoder

Practically, an autoencoder is not the only data abstraction technique. On the contrary, there are linear transformation techniques that proceed with autoencoders which often model inputs using their dominant statistical features [100], [101]. For instance, in Principal Component Analysis (PCA), given N column vectors (series) in $\mathbb{R}^M$, inputs get expressed in terms of the most dominant eigenvalues

and eigenvectors of their covariance matrix. In reality, the eigenvalues and vectors define Principal Components (PC) ($PC \leq M$) that captures the maximum variance within the dataset. In another perspective, we can think of PCA performing a singular value decomposition of inputs (3.2), where $X = \{X_1, X_2, \ldots, X_N\} : X_i \in \mathbb{R}^{M,1}$ are the input vectors, μ_i is the mean of X_i, and $\{\Sigma, \{\mathcal{U} \text{ and } \mathcal{V}\}\}$ are the eigenvalues and orthonormal eigenvectors of the inputs covariance matrix $C \subset \mathbb{R}^{N \times N}$. After decomposition, PCA express the input as the linear combination of K dominant eigenvectors within $\mathcal{U}$ ($\mathcal{V}$) and the covariance matrix C.

$$\begin{aligned} C_{i,j} &= Cov(X_i, X_j) = \frac{1}{M}(X_i - \mu_i)(X_j - \mu_j)^T \\ C &= \mathcal{U}\, \Sigma\, \mathcal{V} \end{aligned} \tag{3.2}$$

In reality, we can think of PCA's eigenvalue and eigenvector selection as its attempt to identify a line, a plane, or such higher dimensional geometric shape from which the transformed series have a higher variance along their first dimension and the minimum residue along their K^{th} axis. With this transformation, PCA is useful as a higher dimensional data visualization and a dimensionality reduction tool in different machine learning algorithms [102], [103]. However, despite its use, PCA inherently assumes a correlation among transformed series. However, in practice, this is not always evident. Thus, in the worst case or when the analyzed dataset has a nearly diagonal covariance matrix, PCA often becomes less useful. However, in this context, an autoencoder makes no such rigid assumptions. Moreover, an autoencoder does not rely on linear recombination to define the lower dimensional representation of the transformed series. On the contrary, it relies on the encoder's and decoder's ability to learn optimal transformation functions (g(.), f(.)). In this aspect, in autoencoders, the most common decoder function f(.) is the reconstruction loss given in (3.3). In this case, a decoder is expected to tune its weights in a manner that minimizes reconstruction error given the inputs latent representations $Z = \{Z_1, Z_2, Z_3, \ldots, Z_N\} : Z_i \in \mathbb{R}^{\tau}$ where $\tau < M$. On the contrary, an encoder's transformation function g(.) gets expected to identify latent space features that are the basis for reconstruction.

$$L_{reconstruction}(X, \hat{X}) = \frac{1}{N}\sum_{i=1}^{N} || X_i - \hat{X}_i ||_{l2} \tag{3.3}$$

Even though latent space representations obtained through a reconstruction loss might seem trivial at first glance, they are often useful while initializing neural networks performing different data mining tasks. For instance, in [6], an autoencoder with a reconstruction loss gets utilized to initialize a neural network performing latent space-time series. In the paper, the authors first trained a basic autoencoder to learn reconstructable latent features. Following this training, the authors discarded the decoder and further trained the encoder using latent space K-mean clustering and Kullback–Leibler (KL) divergence. Additionally, in [56], different architectural setups of autoencoders get assessed in the context of latent space time series clustering. In these studies, an autoencoder's latent space representation gets assumed to be crucial on assisting clustering algorithms. In this regard, it get assumed that it helps them focus on the most relevant features of clustered datasets. In reality, autoencoders have also found a use besides clustering. For instance, in [104], autoencoders have been utilized in a semi-supervised setup to improve classification accuracies. In practice, there are

scenarios where we could have a mixture of labeled and unlabeled datasets. In [104], the authors proposed to utilize the encoder portion of a pre-trained autoencoder as a building block of a classifier. The underlying argument behind the proposal was that an autoencoder learns relatively close latent space representations for similarly labeled input datasets. Thus, even if the labels of some datasets are missing, the encoder portion of a pre-trained autoencoder get expected to guide a classifier network in the right direction. Thus, it serves as a regularizer for a classifier network. In reality, this use of autoencoders also got further investigated in [105]. However, in this investigation, the authors proposed to corrupt input datasets with noise to force the autoencoder to give attention to the most dominant features that enable it to filter out and reconstruct a corrupted input.

In addition to their use in initializing neural networks, in practice, autoencoders by themselves get utilized as the main optimization setup in different data mining tasks. For instance in [106]–[108], autoencoders have been utilized to detect anomalies in input datasets. The underlying concept behind such proposals is the assumption that anomalies often have higher reconstruction loss. In another perspective, [109] showed the better performance of autoencoders in dimensionality reduction. The proposal showed that the autoencoders performed better compared to their linear counterparts. In general, given an appropriate design and guiding objective function, an autoencoder can learn latent features useful under different setups. In this aspect, there is a range of variables contributing to the better performance of an autoencoder, i.e., compared to its predecessors. For instance, if the inputs of an autoencoder have very similar shapes, then we expect their latent representation to get confined within a small region of the latent space. However, when this is the case, the decoder might have difficulties distinguishing between the latent space representation of the input datasets. On the contrary, if the reverse is true, the autoencoder could have difficulties generalizing since the decoder is expected to be able to interpret a wide area of the latent space. Additionally, even in a relatively normal case, the type of $f(.)$ and $g(.)$ learned by an autoencoder are directly or indirectly controlled by different parameters and sub-parameters of the network. For instance, layer organization (architecture), activation function, initialization, regularization, etc. To this end, if we propose to augment time series averages from the latent space of the autoencoder, we have to carefully control and guide such parameters such that the latent space representations meet our requirements. For instance, in the context of time series average augmentation, we desire the autoencoders to extract dense (compact) latent space features given input datasets are highly correlated. Thus, this way, we expect the autoencoders to be able to filter out time-domain perturbance such as phase shift. Moreover, by focusing on such dense principal components, we significantly reduce the search area of the mean to a small confined region of the latent space. This, in turn, will help the decoder portion of the autoencoder to reconstruct a sample, such as the arithmetic mean of latent embedding, in a manner that resembles the time domain inputs. However, it should also be noted that very dense latent space representations could also become a challenge to the decoder in terms of resolvability. Thus, to ensure optimal re-projection of a latent mean, we are expected to establish a balance between dense representations and their resolvability in some manner.

In addition to the compactness of the latent features, another factor to take into account would be the dimension of the latent features. This is because if we significantly reduce the dimension of the latent space features, then we would be losing too much information. In other words, if we see it in the context of PCA, a smaller dimension means we will be focusing on the first N Principal Components (PC) while neglecting the rest. However, in some cases, the lower PC could carry critical information if the encoded series have similar shapes or if they are highly correlated. In general, an average augmentation process should take these extreme cases in mind and must find a balance through different means. With these technicalities in mind, we present our proposals which aim to augment the times series averages from a basic autoencoder mainly constructed from *Convolutional* layers.

3.2.1 Time Series Average Estimation Using Basic Autoencoders

In this subsection, we first propose to augment the time domain average from the latent space of an autoencoder that resembles the Visual Geometric Group 16 (VGG16) C/D presented in chapter two [57]. In reality, we adopt the VGG16 architectural setup for two main reasons. First, we believe the successive stacking of the *Convolutional* layers will significantly refine time domain phase shifts. Secondly, in time series averaging, we aim to preserve shapes observed in the averaged set. In this regard, the *Convolutional* layer stacking in VGG16 helps abstract shapes using a consecutively increasing receptive field. For instance, if we assume we have three *Convolutional* layers within a VGG16 stack, the first *Convolutional* layer kernel will analyze and transform overlapping segments of an input. However, due to the stacking, the internal two layers can zoom into a specific segment since the inner *Convolutional* kernels slide along the output of a preceding layer. From another perspective,

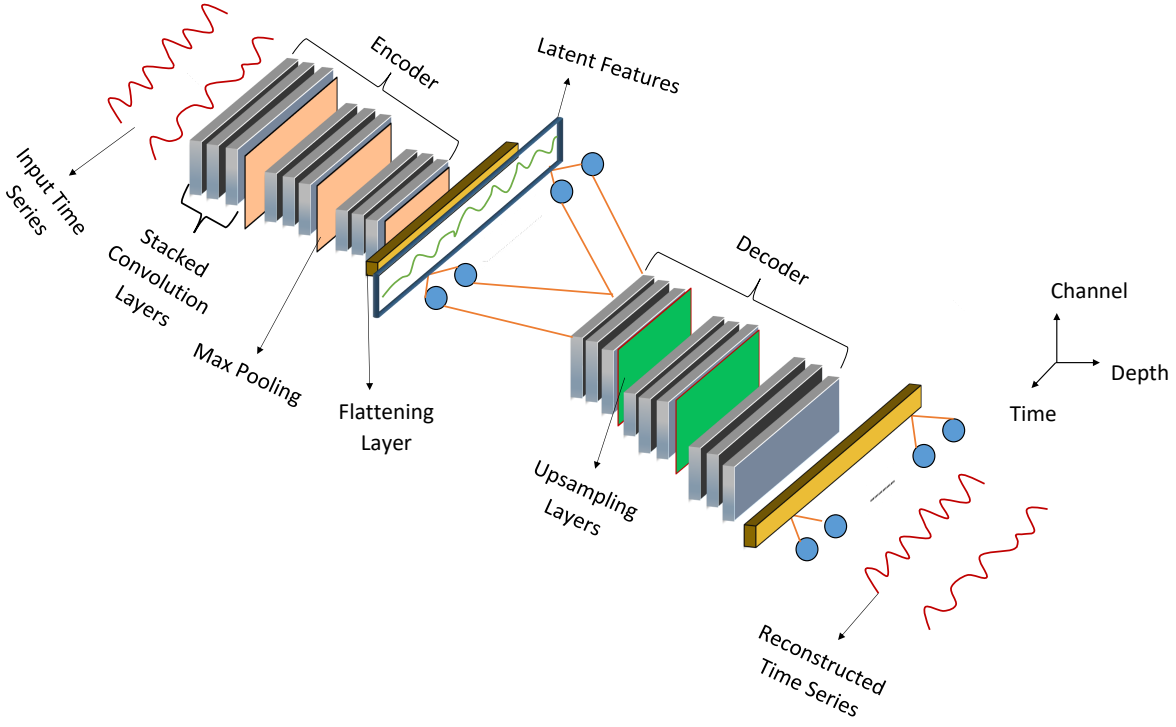


Figure 3.9: Proposed reduced VGG16 autoencoder architecture

we can think of *Convolutional* layer stacking as the zooming touches performed on the screens of smartphones. In this aspect, we expect a first touch to zoom on to a particular point (segment) of interest up to a certain extent. However, since consecutive touches work on the outputs of predecessor touches, we will end up with a significant focus on the point of interest after consecutive zooming touches. In the context of shape abstraction, this capability will aid a VGG16 based autoencoder to focus on unique shapes while filtering out common features. For instance, if an input dataset has peaks and through that are unique to it, then it would be wise to zoom in and analyze the sharpness and the smoothness of the peaks and troughs rather than emphasizing on common shapes. However, it should also be noted that an improper objective function, network architecture, and parameter configuration could also lead us to focus on features that do not align with our interests. With this intuition in mind, we refrained from adopting the full VGG16 C/D architecture. This is because the datasets we work on, i.e., UCR, often have a limited number of training samples. To this end, training a network with large numbers of parameters such as VGG16 has a higher likelihood of overfitting. In other words, given a limited number of training inputs, a larger network could memorize the shapes observed in the training set. This could significantly reduce its generalization capability. Additionally, we desire to base the augmentation process on a setup with an optimal computational requirement. To this end, we believe our proposed network should be relatively shallower (smaller) compared to the architectures presented in Table 2.3. However, with a smaller network, the memory links evident in the ResNet and Inception could significantly introduce the distortion present in the time domain. This is contrary to our initial desire of extracting compact (dense) latent space representations, which is also the underlying argument behind our selection of the VGG16 architecture rather than its counterparts. With these technicalities in mind, we propose the reduced version of the VGG16 architecture shown in Figure 3.9 which we afterward call the **reduced VGG16**.

3.2.2 Architecture Description

In the proposed autoencoder, we use *Convolutional* stacks that are composed of three one dimensional *Convolutional* layers. At the encoder and decoder, we have used three such *Convolutional* stacks. Moreover, we have also used three *MaxPooling* layers at the end of each *Convolutional* stack of the encoder. In neural networks, given a kernel size of K, a *MaxPooling* layer takes the maximum of the values under the receptive field of the kernel. In our architecture, we have set the kernel size to 3. Thus, each encoder *Convolutional* stack successively reduces the dimension of an input time series by a factor of 3. Thus, the total dimension reduction aggregates to a factor of 27. However, at times, some of the input dimensions are relatively small to be scaled by an aggregate reduction factor of 27. When this is the case, we set the *MaxPooling* kernel to 2. On the contrary, at the decoder, we have used two *UpSampling* layers to perform a dummy interpolation while performing the reconstruction. In neural networks, given the coordinate values of a feature map and an *Upsampling* layer kernel size of K, an *UpSampling* layer repeats each coordinate value K times. In our case, each *UpSampling* layers have a kernel size of 3. In addition to these layers, at the end of the encoder and decoder modules, we have used a *Flattening* and a fully connected *Dense* layers. The *Flattening* layers gets used to convert the two-dimensional feature maps of the *Convolutional* layers into a one-dimensional representation. This gets achieved by stacking the columns of the feature maps along their first axis. On the contrary,

the encoder's and decoder's *Dense* layers are respectively used to learn the one-dimensional latent space embedding and to generate the reconstructed series. Finally, we have used the ReLU activation function with the exception of the encoder's first *Convolutional* layer and the decoder's output *Dense* layer. On the two layers, we have used a *Linear* activation function to support negative and positive values at the decoder output and to keep the encoder symmetrical to the decoder. In conclusion, given a time series in $\mathbb{R}^M$, the parameters of each layer are summarized in Table 3.7.

Table 3.7: Layer configurations of the proposed reduced VGG16 autoencoder

Module	Layer (s)	Input dim.	Output dim.	# parameters
Encoder	Reshape	(Batch, M)	(Batch, M,1)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, 1, M)	(Batch, M, 32)	3232
	MaxPooling	(Batch, M, 32)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 32)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	30,912
	MaxPooling	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 64)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{9} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 128)	123,264
	MaxPooling	(Batch, $\lfloor \frac{M}{9} \rfloor$, 128)	(Batch, $\lfloor \frac{M}{27} \rfloor$, 128)	0
	Flattening	(Batch, $\lfloor \frac{M}{27} \rfloor$, 128)	(Batch, $\lfloor \frac{M}{27} \rfloor \times 128$)	0
Latent	*Dense*	(Batch, $\lfloor \frac{M}{27} \rfloor \times 128$)	(Batch, $\lfloor \frac{M}{4} \rfloor$)	$(\lfloor \frac{M}{27} \rfloor \times 128) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$
Decoder	Reshape	(Batch, $\lfloor \frac{M}{4} \rfloor$)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 128)	99,072
	UpSampling	(Batch, $\lfloor \frac{M}{4} \rfloor$, 128)	(Batch, $\lfloor \frac{3\times M}{4} \rfloor$, 128)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \end{bmatrix}$	(Batch, $\lfloor \frac{3\times M}{4} \rfloor$, 64)	(Batch, $\lfloor \frac{3\times M}{4} \rfloor$, 64)	49,344
	UpSampling	(Batch, $\lfloor \frac{3\times M}{4} \rfloor$, 64)	(Batch,$\lfloor \frac{9\times M}{4} \rfloor$, 64)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, $\lfloor \frac{9\times M}{4} \rfloor$, 64)	(Batch, $\lfloor \frac{9\times M}{4} \rfloor$, 32)	12,384
	Flattening	(Batch, $\lfloor \frac{9\times M}{4} \rfloor$, 32)	(Batch, $\lfloor \frac{9\times M}{4} \rfloor \times 32$)	0
Time Domain	*Dense*	(Batch, $\lfloor \frac{9\times M}{4} \rfloor \times 32$)	(Batch, M)	$(\lfloor \frac{9\times M}{4} \rfloor \times 32) \times M + M$

3.2.3 Experimental Setup, Average Estimation and Evaluation Technique

Experimental Setups: We have trained the proposed reduced VGG16 architecture using an 80/20 split, i.e., 80% of the training data gets used for training and 20% for validation. Moreover, we train the network for 600 epochs: with no weight regularization, default weight initialization (Glorot Uniform),

a 10^{-4} learning rate, and batch size that is equal to $\lfloor \frac{1}{4} \times N \rfloor$, where N is the number of the input series. Finally, we have used the Adaptive Moment Estimation (Adam) optimizer for gradient calculation [65].

Average Estimations: After training the network, we take the per class arithmetic mean of the latent space representations of the training datasets to estimate the latent averages. In other words, in the context of the latent means, we are making a maximum likelihood parameter estimation under the assumption of Gaussian distribution. In addition to generating latent space estimations, we use the decoder to project the latent space estimation to the time domain.

Evaluation Techniques: After generating the latent space and time domain estimations from the training set, we used the trained network to project the test datasets into the latent space. We then performed a Nearest Centroid Classification (NCC) using the estimated averages and the latent and time domain representations of the test datasets. For the NCC, we have respectively used Euclidean and DTW distances for the latent and time domain representations of the test datasets. We finally report the outcomes that obtained the best latent space classification accuracy and their corresponding time domain classification outcome. In reality, we base our selection criteria on the latent space classification accuracy since we propose to mimic multiple alignments in the latent space rather than the time domain.

In practice, the quality of a time series average is measured using Within Group Squared Sum (WGSS), i.e., (1.1) [16]. However, we have avoided evaluating the estimates using WGSS since it only gives information about a single cluster. In other words, it does not provide clear information on the quality (representativeness) of the estimated means (centroid) in the context of multi-class (cluster) averages. For instance, an estimated latent mean could show a small WGSS while the underlying multi-class latent space representations are overlapping and indistinguishable for the decoder. In this aspect, a NCC classification is relatively self-explanatory. Moreover, we also expect a NCC to maximize its classification accuracy by minimizing its WGSS. Additionally, NCC could become handy while interpreting different scenarios. For instance, a high latent and time domain NCC accuracy could imply the per-class embedding of the multi-class input time series are separable and compact. Alternatively, a NCC could achieve a high classification accuracy if the per-class embedding or time domain representations of input dataset are a very close neighborhood of each other. In this case, the high NCC accuracy implies the quality of the estimated means and the compactness of the time or latent space embedding. In reality, we can also pair NCC with dimensional visualization tools such as t-Distributed Stochastic Neighbor Embedding (t-SNE) [110] or PCA [100] in order to make conclusive remarks about the different scenarios.

Besides conducting NCC classifications, i.e., using our proposed approach, we have also compared the performance of our proposal to its counterparts. To make the comparison, we have utilized the outcomes reported in [20]. In [20], a NCC evaluation was performed using the estimates of DBA, SDBA and DTAN on 84 UCR datasets. In the evaluation, DTAN got first trained for 2500 epochs and four regularization setups. Next, the authors took the arithmetic mean of the morphed train datasets

as an estimate. Following this, the test datasets got morphed using the trained network. Finally, a NCC classification got conducted using the morphed series, estimated means, and Euclidean distance. The authors then reported the outcomes that obtained the best classification accuracy. On the contrary, for the DTW based averaging techniques, the authors used Tslearner's [111] implementation of DTW, DBA, and SDBA to conduct similar NCC classifications on 84 UCR datasets [20]. In general, the authors executed DBA and SDBA for 100 iterations. Moreover, they evaluated SDBA using five γ values, i.e., for $\gamma = [0.001,\ 0.01,\ 0.1,\ 1\ and\ 10]$. Finally, for all of the DTW based techniques, they reported the outcomes that obtained the best NCC accuracy. In reality, we have also validated the reported outcomes of DBA and SDBA using the same NCC setups utilized in [20]. Our assessment shows that the reported results are not biased. Thus we adopted the reported results as they are. However, since we could not find a standardized implementation of DTAN, we accepted the outcomes reported in [20] to be valid.

Hypothesis Tests for NCC Classification Accuracies: The next logical question that needs to get asked is, how do we compare the outcomes of different averaging techniques? In this regard, some papers utilized win/tile/loss tables and plots to compare classification accuracies of competing algorithms [16], [25]. However, in reality, such comparisons might be misleading for various reasons. For instance, a given averaging technique might lose with a small margin to have a significant practical implication. To account for such ambiguities of wins/ties/losses analysis, we also compare classification outcomes in a statistical sense, i.e., to use the overall maximum, minimum, mean, median accuracies, and box-whisker plots. However, in practice, some statistical parameters, such as median accuracies, tend to get biased by outlier accuracies. Moreover, even though a box-whisker plot is more revealing compared to wins/ties/loss analysis, it still does not take the individual accuracy difference into account. To address these issues, we propose to further evaluate the performances of the averaging techniques using hypothesis evaluation techniques [112]. However, in practice, we have a range of hypothesis evaluation techniques. Thus, careful consideration must be taken while selecting the evaluation techniques. In this aspect, we noted that our experimental evaluations can be taken to be dependent and paired, i.e., different sets of averaging techniques get applied to the same datasets. Moreover, we also noted that we do not expect the outcome of our experimental evaluations to follow a specific distribution curve, i.e., they are non-parametric. With these key factors in mind, we have selected the Friedman signed rank test and Wilcoxon hypothesis tests as our pre and post hypotheses evaluation techniques [112], [113].

In reality, the Wilcoxon hypothesis test initially assumes that classification accuracies get obtained by performing a pair of experiments on a set of test subjects. For instance, while performing NCC classification using the estimates of two averaging techniques and the UCR dataset. Given this condition is met, the Wilcoxon hypothesis test initially assumes a certain observation is true, commonly known as a *null hypothesis*. In this context, we initially assume the classification accuracies of two averaging techniques on a set of UCR datasets are equal or statistically indistinguishable, i.e., the difference between their classification accuracies has zero median [113]. Given such *null hypothesis*, the Wilcoxon hypothesis test first computes the difference between the outcomes of the compared

techniques. It then momentarily takes the absolute values of the differences while keeping the track of the negative and positive differences. Following this, the differences get assigned a rank based on their magnitudes, i.e., the smallest difference gets assigned the smallest rank. Finally, the Wilcoxon hypothesis test separately aggregates the ranks of the positive and negative ranks to compute the likelihood of the minimum of the two aggregated ranks under an F distribution [113]. In reality, the likelihood informs us how rare or likely a given *null hypothesis* is. Thus, the smaller the likelihood, the higher the chance that our initial assumption (*null hypothesis*) is invalid. However, to reject a null hypothesis, one is expected to define a threshold over which a null hypothesis gets rejected. In this regard, Wilcoxon defines the threshold often called *p- value* over which an underlying null hypothesis gets rejected. In practice, the most common *p-value* is 5 % (0.05), thus we have adopted this *p-value* in our evaluations. Practically, the Wilcoxon hypothesis test statistically evaluates two competing techniques at a time. However, in our case, we have multiple averaging techniques that we desire to compare their performances. To this end, in addition to the Wilcoxon test, we utilize the Friedman test as a pre-hypothesis evaluation technique. Unlike the Wilcoxon test, the Friedman hypothesis test assigns a rank by comparing the outcomes of the compared techniques. It then evaluates the likelihood of the aggregate ranks of each technique under a Chi-Square distribution which gets then compared to a *p-value* over which the null hypothesis is rejected [112]. With this understanding, we first evaluate the different averaging heuristics using a Friedman hypothesis test and then assess them pairwisely using a Wilcoxon hypothesis test. In practice, the outcomes of such statistical evaluations get often shown using a Critical Difference (CD) diagram. In our case, a CD diagram will have a scaled horizontal line for the average ranks of the averaging techniques. Based on this scaled line, a set of vertical lines gets drawn to show the average Friedman rank of each averaging technique. Finally, the outcomes of the Wilcoxon tests are shown by connecting two Friedman rank lines if the two averaging techniques are considered statistically indifferent [112]. In conclusion, to plot the CD diagrams, we used a Python implementation developed in [7]. With this said, we will proceed with the discussion of the experimental results.

3.2.4 Experimental Results and Interpretation

In this subsection, we will start our assessment of the averaging techniques using a win/tie/loss evaluation. In Table 3.8, an averaging technique is presumed to be winning if its classification accuracy is better than all of its counterparts. Moreover, a tie gets recorded if two or more averaging techniques have the same classification accuracy. On the contrary, if an averaging technique is neither tied nor winning, it will be considered losing. According to Table 3.8, DTAN is winning on most of the dataset, i.e., on 43 of the 84 UCR datasets to be exact. This performance gets seconded by SDBA followed by the latent space classification of the proposed autoencoder (Enc_Lat). We marked these three top-performing techniques using bold letters in Table 3.8. Additionally, Table 3.8 also reveals that the time domain estimation of the autoencoder (Enc_Time) behaves as an Arithmetic mean.

To further validate these observations, we also assess the classification results using the box-whisker plot shown in Figure 3.10, where the statistical parameters of the plots are summarized in Table 3.9. According to Table 3.9, DTAN has a worst-case classification accuracy (lower whisker) of 25.97%,

Table 3.8: Analysis of wins/ties/losses of the NCC accuracies that are obtained using estimates of the basic autoencoder and its counterparts.

Technique	Wins	Ties	Losses
Arithmetic	1	0	83
DBA	4	4	76
DTAN	**43**	**5**	**36**
Enc_Lat	**6**	**1**	**77**
Enc_Time	1	0	83
SDBA	**21**	6	**56**

whereas 50% of its classification accuracies lie between 58.55% and 85.45%. With these statistics, it obtained a median NCC classification accuracy of 72.94%. On the contrary, in the autoencoder's latent space (Enc_Lat), the worst-case classification accuracy is 19.66%. Moreover, 50% of the autoencoder's latent space classification accuracies are within the ranges of 51.16% and 74.48%, where the median accuracy is 58.66%. Thus, statistically speaking, the registration obtained in the latent space of the proposed autoencoder is worst than the state-of-the-art (DTAN). If we also compare the autoencoder's latent space registration to that of the DTW based techniques, i.e., DBA and SDBA; the two techniques respectively have median accuracies of 65.04% and 69.02%. Moreover, 50% of their NCC accuracies are within the ranges of 54.05 *to* 79.84% and 57.41 *to* 81.22%. These statistics are also better than the

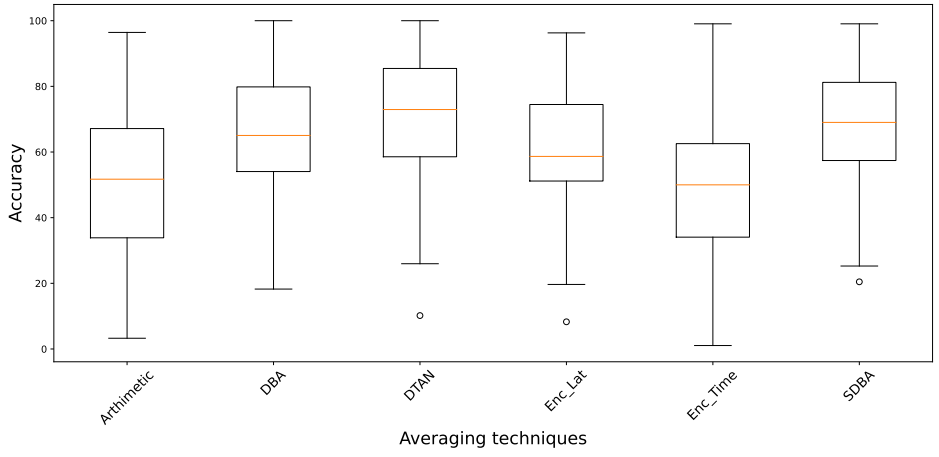


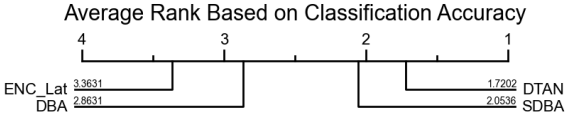
Figure 3.10: Box-whisker plot comparison of NCC accuracies that are obtained using the averages estimated with the basic autoencoder and its counterparts.

autoencoder's latent space accuracies. However, to make matters worse, the time domain projections performed poorly compared to an arithmetic mean. In this regard, the autoencoder projected estimates obtain a median accuracy of 50%. Moreover, 50% of its time domain NCC accuracies are in between 34.07% and 62.55%. However, the arithmetic means obtained a 51.72% median accuracy. Moreover, 50% of its NCC accuracies are within the ranges of 33.87% and 67.14%. To further validate these statistical observations, we will analyze the NCC accuracies using the hypothesis tests discussed in the previous section. In the evaluation, we separated the hypothesis tests into two broad categories. First, we compare the NCC classification accuracies that we presume are obtained in the registered

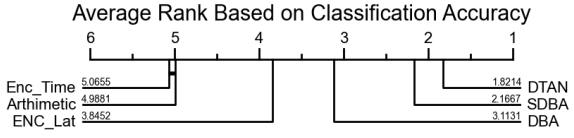
Table 3.9: Statistical parameters for the box-whisker plot shown in Figure 3.10

Technique	Top Whisker	Bottom Whisker	25% percentile	75% percentile	Median
Arithmetic	96.43	3.27	33.87	67.14	51.72
DBA	100	18.25	54.05	79.84	65.04
DTAN	**100**	**25.97**	**58.55**	**85.45**	**72.94**
Enc_Lat	**96.29**	**19.66**	**51.16**	**74.48**	**58.66**
Enc_Time	99.05	1.05	34.07	62.55	50.00
SDBA	**99.05**	**25.27**	**57.41**	**81.22**	**69.02**

space. In this regard, we take the classification accuracies of DTAN, DBA, SDBA and Enc_Lat as a registered space classification accuracies. This is because, in the NCC, DBA and SDBA estimates are paired with DTW which warps the classified series and the estimates into DTW space. On the contrary, DTAN transforms a test set into the morphed space before performing the classification. In this aspect, we considered the latent space of the autoencoder as registered space of our approach since we augment the time domains from this space. Following the evaluation of the registered space classification accuracies, we include the NCC accuracies of the arithmetic mean and Enc_Time to make our second assessment. In general, Figure 3.11 (a) demonstrates the statistical comparison of DTAN, DBA, SDBA and Enc_Lat using their registered space classification accuracies.



(a) Evaluation of NCC accuracies that are obtained by using averages in their registered spaces



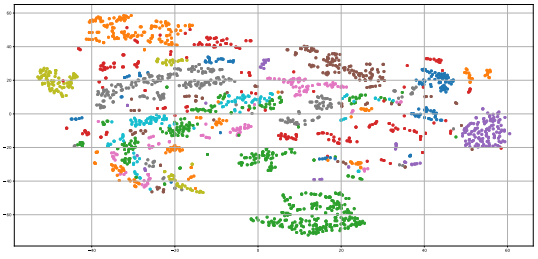
(b) Evaluation of NCC accuracies obtained by using averages in their registered and un-registered spaces

Figure 3.11: Hypotesis tests for averages estimated with the basic autoencoder and its counterparts

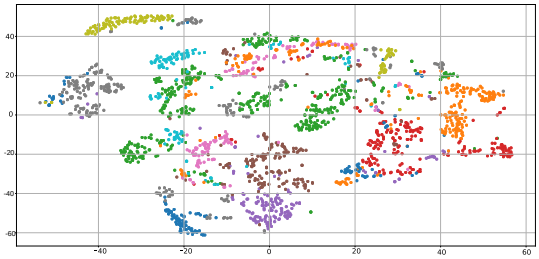
The Friedman and Wilcoxon hypothesis tests also reveal that DTAN outperforms all averaging techniques. However, unlike the box-whisker analysis, the Wilcoxon signed rank test identified that the performances of the arithmetic mean and Enc_Time to be statistically indifferent, i.e., as shown in Figure 3.11 (b). In Figure 3.11 (b), this equality is shown with the bold horizontal line connecting the lines indicating the average Friedman rank of Enc_Time and Arithmetic. In this context, the Wilcoxon signed rank test identified that the *p-value* for the two averaging techniques is 0.76. This is way above the critical *p-value* of 0.05 over which we reject the *null hypothesis*. In other words, at a dataset level, most of the classification accuracies of the arithmetic mean and Enc_Time got found to be not significantly different.

Practically, one possible technical reason behind this outcome could be the latent space representations are not compact enough for augmenting a time domain estimate. For instance, in Figure 3.12, we have given the t-SNE projection of: the time domain, DTAN morphed space and autoencoder latent space representation of the UCR's *FacesUCR* test datasets. In reality, we have extracted DTAN's morphed space t-SNE projection from [25]. In general, even though the autoencoder's latent space

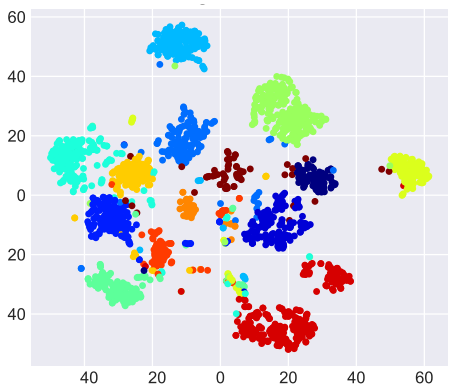
representations are relatively denser than the time domain, they are not comparable to DTAN's morphed space representation. This will have a major implication on the autoencoder's projected time-domain estimates. This is because, in a less dense latent space, we expect to have a lot of open



(a) *FacesUCR* in time domain



(b) *FacesUCR* in the latent space of the reduced VGG16

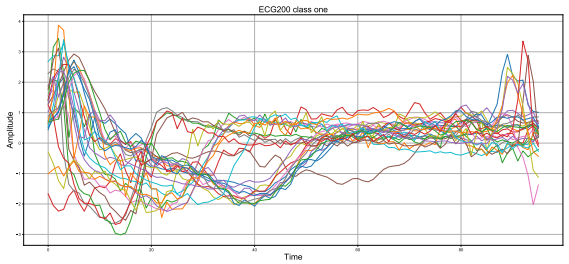


(c) *FacesUCR* in the morphed space of DTAN [25]

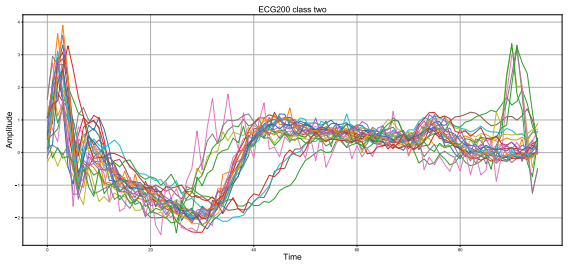
Figure 3.12: t-SNE projections for the UCR archive's *FacesUCR* test datasets: (a) in time domain, (b) in autoencoder's latent space and (c) in DTAN's morphed space

space between the latent space representations of the input dataset the decoder has no knowledge of. To this end, when we take an arithmetic mean of the latent space features, there is a higher chance that the latent mean will fall in one of these open spaces. Thus, the decoder will likely have difficulty re-projecting them into a more optimal time domain series, i.e., using the weights it has learned from training datasets. Conseqently, we expect the projected time-domain estimates to behave as a time domain arithmetic mean. This is because a sparse latent space representation implies the effect of temporal distortion is not minimized significantly.

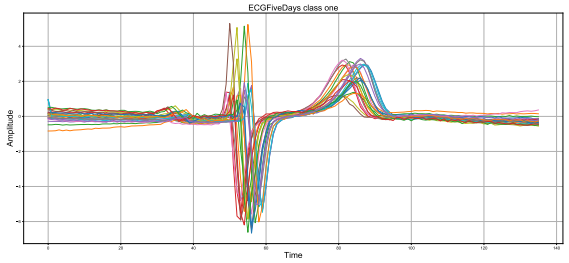
In order to visually demonstrate this observation, in Figure 3.13 we have plotted the test split of the UCR's *ECG200* and *ECGFiveDays* datasets as an example. Moreover, in Figure 3.14, we have plotted the averages that were estimated using arithmetic, autoencoder, DBA and SDBA. From Figures 3.14 (a) and 3.14 (b), we can see that there is a high degree of resemblance between a time domain arithmetic mean and its autoencoder estimated counterpart. This is in line with our initial argument of the autoencoder's latent space not being compact enough to overcome time domain temporal distortions. This fact is also evident in Table 3.10, where we have given the NCC accuracies for the *ECG200* and *ECGFiveDays* datasests. According to Table 3.10, DTAN obtained the best NCC classification accuracies, i.e., 79% on *ECG200* and 97.79% on *ECGFiveDays*. On the contrary, Enc_Time obtained a 65% and 52.15% NCC accuracies. This is very close to the NCC accuracies that are obtained using



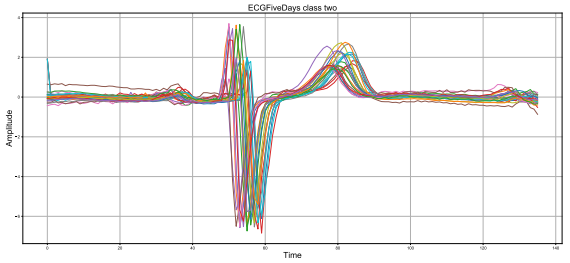
(a) Test dataset from *ECG200* class 1



(b) Test dataset from *ECG200* class 2

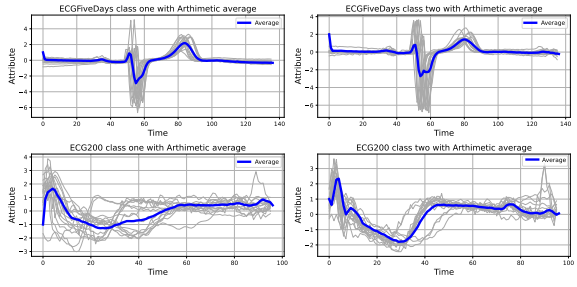


(c) Test dataset from *ECGFiveDays* class 1

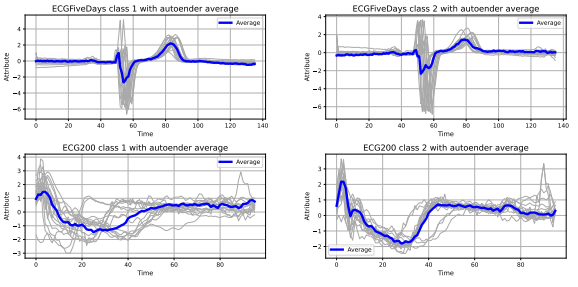


(d) Test dataset from *ECGFiveDays* class 2

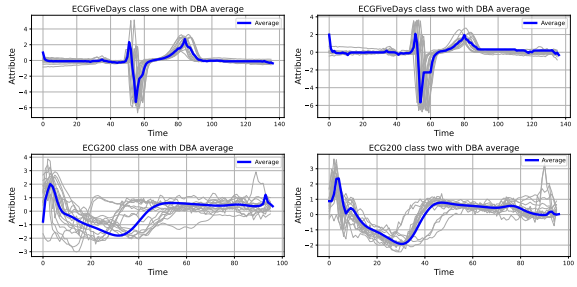
Figure 3.13: The UCR archive's *ECG200* and *ECGFiveDays* test datasets



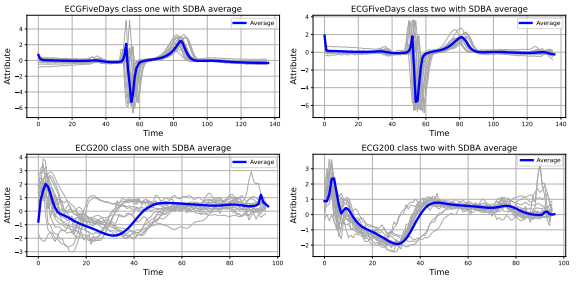
(a) Arithmetic averages



(b) Autoencoder augmented averages



(c) DBA estimated averages



(d) SDBA estimated averages

Figure 3.14: Averages estimated for the UCR archive's *ECG200* and *ECGFiveDays* datasets. The averages were estimated by using their training datasets and different averaging techniques: arithmetic (a), autoencoder (b), DBA (c) and SDBA (d)

an arithmetic mean, i.e., 67% and 52.96%. Thus, further validating the high resemblance between an arithmetic mean and those estimated by the basic autoencoder. With these observations in mind, in the next subsection, we aim to investigate factors affecting the quality of the autoencoder latent space features. In this regard, we first aim to investigate the impact of the autoencoder network architecture on the projected means. To this end, we propose to investigate autoencoders based on alternative architecture such as Inception and ResNet. Additionally, we noted some gaps in the proposed reduced

Table 3.10: NCC classification accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets

Averaging Technique	Accuracy on *ECG200* in %	Accuracy on *ECGFiveDays* in %
Arithmetic	67	52.96
DBA	65	52.15
DTAN	**79**	**97.79**
Enc_Time	65	52.15
SDBA	73	67.02

VGG16 architecture and the overall experimental setup. To this end, in addition to evaluating alternative setups, we aim to address these gas and re-assess the VGG16 architecture. In this regard, we noted that the filter arrangement at the encoder is sequentially increasing as we go down the network. This in turn significantly increases the dimension of the flattened features. To this end, the number of parameters at the encoder's fully connected *Dense* layer becomes significantly large compared to its *Convolutional* counterparts. Practically, the two *Dense* layers at the encoder and decoder are relatively susceptible to overfitting. Thus, in reality, having a large number of network parameters at these layers is not wise. Moreover, in autoencoders, we often desire to introduce a bottleneck at the encoder to force it to extract the most useful features. To this end, we strongly believe that reduced VGG16's encoder filter arrangement should also be modified to align with the observations*Convolutional* layers. In addition to this, at the reduced VGG16 decoder, we have used an unintelligent *UpSampling* layer to sequentially increase the dimension of the latent space representation by repeating coordinate values. However, in practice, we can possibly perform this task more intelligently using a transposed convolution that uses trainable weights to up-sample its inputs. Thus, we also propose to modify the decoder portion of the reduced VGG16 architecture by replacing the *UpSampling* layers with transposed convolution. Additionally, in the reduced VGG16, we have initialized layers using their default layer weight initialization technique, i.e., *Glorot Uniform*. However, in the network, we have two different activation functions, i.e., *Linear* and ReLU. In practice, these activations are known to give better overall network performance under different layer weight initialization techniques, i.e., Glorout uniform/normal and He uniform/normal [66], [67]. With these observations in mind, we also propose to use the proposed initialization layer weight techniques while assessing the modified reduced VGG16 based autoencoder and the ResNet and Inception architectures. Finally, in addition to the gaps we observed in the reduced VGG16, we have also observed some gaps in our training setup. In this regard, we only trained our proposed network for five regularization setups and later reported the best-performing outcome. However, due to the randomness of weights initialization, it is difficult to capture outlier performance, such as maximum accuracies, with a limited number of trials. To address this issue, in our extended evaluations we propose to run 25 repeated trials for each dataset, regularization setup, and network architectures; i.e., 100 training evaluations per a single dataset.

Even though we expect the minor and major architectural modifications to improve the quality of the estimates, in reality, we can not fully rely on minor modifications and architectural changes for significant improvements. This is because, in addition to network architecture, other factors, such as objective function, also play a role in the type of extracted latent features. In this regard, we asked

ourselves, can we only depend on reconstructable features to get compact latent space representations? The answer to this question is mostly no. This is because, in the UCR archive and most practical cases, shapes differentiating one class from another are mostly minor. For instance, from Figure 3.13 (c) and 3.13 (c), we can see that the shape difference between the two classes *ECGFiveDays* datasets is minor. Thus, if we extract the reconstructable feature of these datasets, the features will most likely have similar patterns. This is because an autoencoder has no prior information about the difference in their class labels. To this end, the multiclass latent space features will end up sharing similar regions of the latent space, i.e., as shown in Figure 3.12 (b). This, in turn, will confuse the decoder and prevents it from optimally projecting the estimated per-class averages. In addition to the mixing of the per-class latent features, the basic autoencoder performs a one-to-one mapping between input datasets and latent space representations. Thus, in reality, the latent space of the basic autoencoder is in a sense discrete. To this end, when we re-project the latent space estimated means, the decoder is asked to interpret something it has not seen before. In this regard, we have two possibilities for assisting the decoder. First, we could make the latent space representations dense and separable so that the latent mean lies in the near neighborhood of the input dataset's latent representations. This, in turn, will help the decoder to re-project the estimated means in a manner that has minimized shape distortion. Another alternative solution would be, utilizing an autoencoder setup with a relatively continuous latent space. One possibility in this regard would be utilizing Variational AutoEncoder (VAE) [38]. In reality, each choice has its own set of requirements and limitations. In this regard, the former approach requires a thorough analysis, the customization of the objective function, and the overall architectural setup. In reality, changing the previously proposed autoencoders into a VAE is relatively easy. However, a basic VAE tries to fit every per class latent feature into a normal Gaussian distribution which could make the latent representation to be indistinguishable. With these pros and cons in mind, we will first present our extended evaluation of the basic autoencoders by first making minor and major architectural modifications. Following these evaluations, we assess the possibility of utilizing the latent space of the variational variant of the evaluated basic autoencoders.

3.3 Extended Evaluation of Basic Autoencoders and their Variational Variants

In this subsection, we start our discussion with the minor architectural adjustments of the reduced VGG16 architecture, which we will further call the modified reduced VGG16. After presenting the configurations of the reduced VGG16, we then present the architectural configuration of the autoencoders that resemble the ResNet and Inception version two [58], [60]. In these architectures, we also aim to keep the number of trainable parameters under watch to minimize the chances of overfitting. With this said, we will proceed with the discussion of the basic autoencoders.

3.3.1 Proposed Modified Reduced VGG16 Autoencoder

In this setup, we modify the reduced VGG16 autoencoder shown in Figure 3.9 to address some limitations we have observed. In this aspect, in the reduced VGG16, the three *Convolutional* stacks have

a filter size of {32, 64, 128}. However, when the output of the last *Convolutional* gets flattened, we will end up with a $(Batch, \langle \frac{M}{4} \times 128 \rangle)$ features as the inputs of a fully connected *Dense* layers. This,

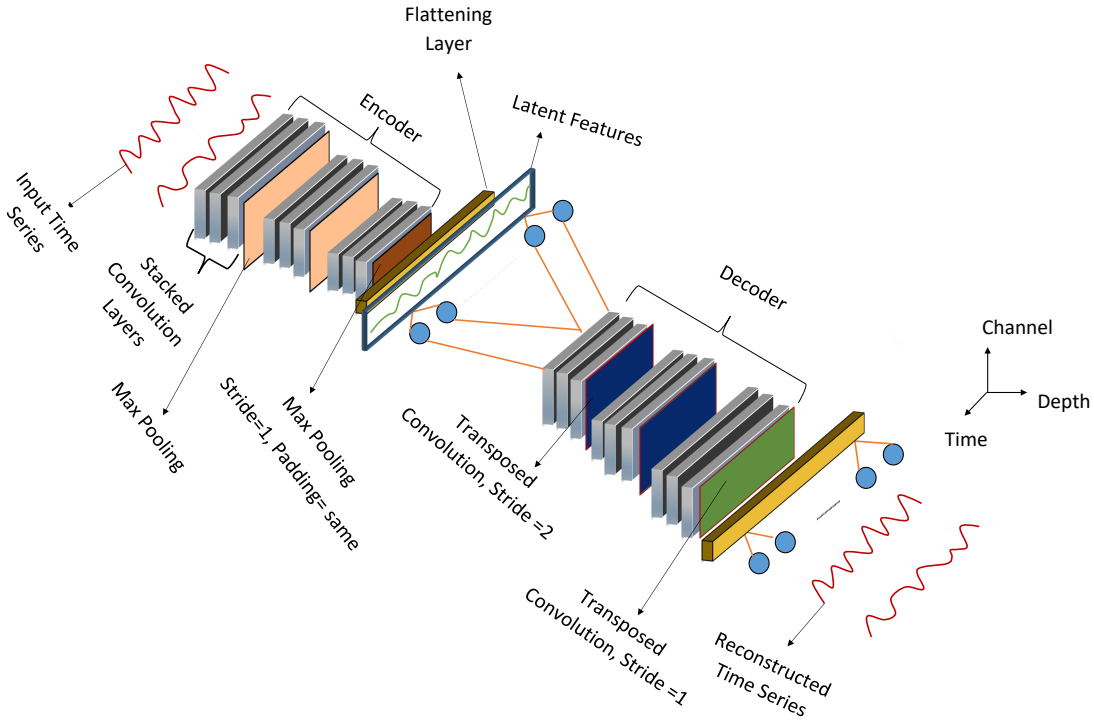


Figure 3.15: Proposed modified reduced VGG16 autoencoder

in turn, will prohibit us from introducing a bottleneck for the *Convolutional* layer features that are often desired in autoencoders [59]. Moreover, due to the 128 *Convolutional* channels, the flattened features will incur a significant amount of trainable weights at the encoder's *Dense* layer. To this end, we propose to reconfigure the encoder's filter size to {128, 64, 32} to reduce the dimension of the flattened latent features and to introduce a bottleneck for the outputs of the *Convolutional* stacks. Moreover, we propose to reconfigure the stride and padding of the third encoder *MaxPooling* layer to 1 and same. We propose this modification to remove the requirement of changing the *Convolutional* kernels for datasets with smaller dimensions (length). In addition to these improvements, we also changed the first two *UpSampling* layers of the decoder to two transposed *Convolutional* layers that have a stride of 2. Furthermore, we also append an additional transposed *Convolutional* layer at the last *Convolutional* stack of the decoder. We append this layer to make the operations at the encoder and decoder opposite but symmetrical. Thus, we set the stride of the last *Convolutinoal* layer to 1 to match the stride of the encoder's last *MaxPooling* layer. However, for all the *Convolutional* and *MaxPooling* layers, we set their kernel size to 3. In general, the overall layer configuration for this modification is shown in Table 3.11.

According to Table 3.11, we now have more trainable parameters at the top *Convolutional* stacks rather than than the *Dense* layers. With this reconfiguration, we expect to gain two main advantages. First, in *Convolutional* layers, we have kernels rather than connection weights. Thus, more *Convolutional* parameters imply more channels than weighted connections, i.e., as in the case of *Dense* layers.

Additionally, by using more filters, i.e., at the input layers of the encoder, we will be able to analyze more input features. In another aspect, as we decrease the filter size down the encoder, we create a feature bottleneck expected to force the encoder to be more selective. Additionally, the two decoder's transposed *Convolutional* layers now scale the dimension of the latent features by a factor of 2 rather than 3. This helps to reduce the number of connections at the decoder's *Dense* layer. Finally, we respectively used the Glorot uniform and He normal weight initialization for the *Linear* and ReLU activated layers. On the contrary, all of the remaining *Convolutional* layers that are ReLU activated get initialized with He normal.

Table 3.11: Layer configurations for the modified reduced VGG16 autoencoder

Module	Layer (s)	Input dim.	Output dim.	# parameters
Encoder	Reshape	(Batch, M)	(Batch, M,1)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \end{bmatrix}$	(Batch, 1, M)	(Batch, M, 128)	99,072
	MaxPooling	(Batch, M, 128)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 128)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	49,344
	MaxPooling	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 64)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{9} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	12,384
	MaxPooling	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	0
	Flattening	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor \times 32$)	0
Latent	*Dense*	(Batch, $\lfloor \frac{M}{9} \rfloor \times 32$)	(Batch, $\lfloor \frac{M}{4} \rfloor$)	$(\lfloor \frac{M}{9} \rfloor \times 32) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$
Decoder	Reshape	(Batch, $\lfloor \frac{M}{4} \rfloor$)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 128)	99,072
	Transp. Cnv.	(Batch, $\lfloor \frac{M}{4} \rfloor$, 128)	(Batch, $\lfloor \frac{M}{2} \rfloor$, 128)	49,280
	$\begin{bmatrix} Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{2} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{2} \rfloor$, 64)	49,344
	Transp. Conv.	(Batch, $\lfloor \frac{M}{2} \rfloor$, 64)	(Batch, M, 64)	12,352
	$\begin{bmatrix} Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, M, 64)	(Batch, M, 32)	12,384
	Transp. Conv.	(Batch, M, 32)	(Batch, M, 32)	6,208
	Flattening	(Batch, M, 32)	(Batch, $M \times 32$)	0
Time Domain	*Dense*	(Batch, $M \times 32$)	(Batch, M)	$(M^2 \times 32) + M$

3.3.2 Proposed Reduced Residual Network Architecture

Even though the modified reduced VGG16 is relatively shallow, i.e., compared to its original counterpart, we found it not wise not to investigate the possibility of mixing the features at different stages of the neural network. To this end, in this extended evaluation, we also propose to evaluate a reduced ResNet version of the basic autoencoder. In this regard, Figure 3.16 shows how we propose to modify the reduced VGG16 in order to accommodate the reduced ResNet setup. Moreover, Table 3.12 summa-

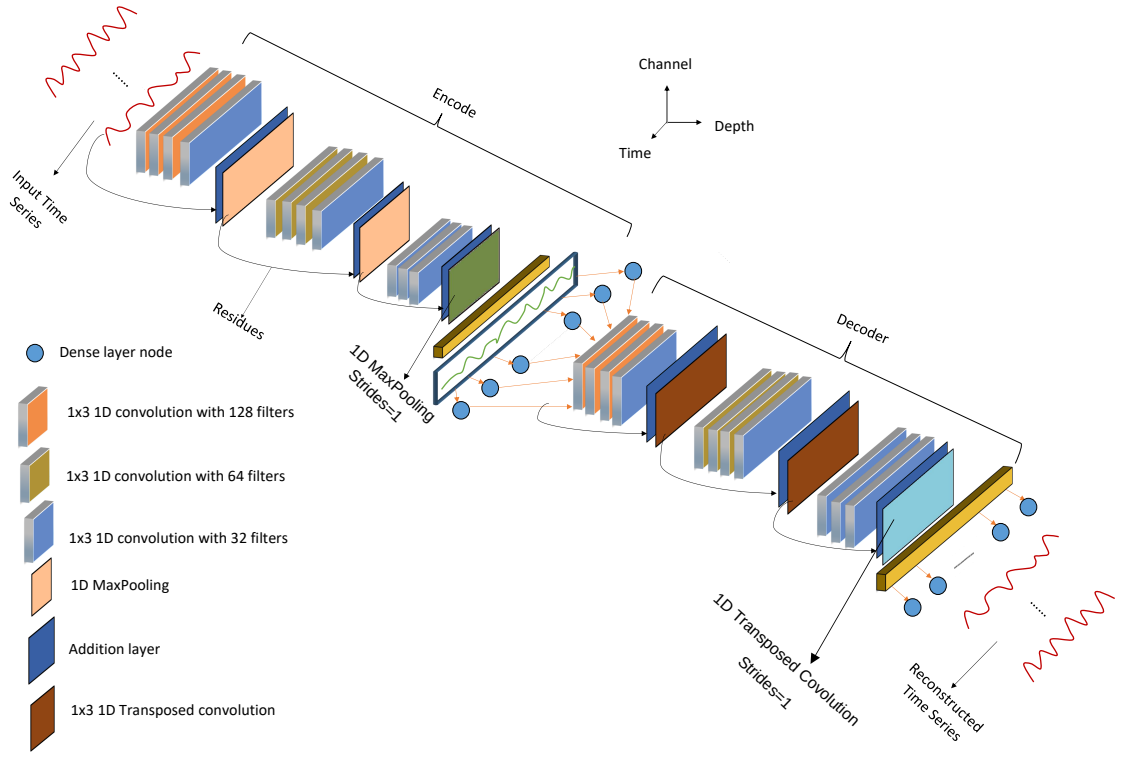


Figure 3.16: Proposed reduced ResNet autoencoder architecture

rizes the overall layer configuration of the proposed reduced ResNet architecture. As compared to its modified reduced VGG16 counterpart, the proposed reduced ResNet has two key differences. First, we have six skip connections (memory links) that are interconnecting *Convolutional* stacks. Moreover, at the encoder and decoder, we added a fourth *Convolutional* layer on the first two *Convolutional* stacks. This additional layer gets used to match the dimensions outputted by the *Convolutional* stacks to the dimensions of the memory links. In practice, ResNet needs such dimension matching since it utilizes an *Addition* layer to combine features [58]. Thus, since we have 32 channels as the outputs of the encoder's and decoder's last *Convolutional* stacks, we have set the channels of the additional *Convolutional* layers to 32. Despite this additional *Convolutional* layer and the skip connections, we have kept the configurations of the remaining layers similar to the configurations used in the modified reduced VGG16. In this aspect, we have set the kernel size of the *MaxPooling* layers to 3 and their strides to 2, i.e., except for the last pooling layer. For this layer, we have respectively set the padding and the stride to same and 1. Furthermore, as in the case of the modified VGG16, we have also set the stride and kernel size of the decoder's first two transposed *Convolutional* layers to 2 and 3. However, we have set the stride of the last transposed *Convolutional* layer to 1 in order to keep the operations

performed at the encoder and decoder opposite but symmetrical. Finally, we have initialized layers

Table 3.12: Layer configurations for the reduced ResNet autoencoder

Module	Layer (s)	Input dim.	Output dim.	# parameters
Encoder	Reshape	(Batch, M)	(Batch, M,1)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, 1, M)	(Batch, M, 32)	111,392
	MaxPooling	(Batch, M, 32)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 32)	0
	ADD	(Batch, M, 32 & 1)	(Batch, M, 32)	0
	MaxPooling	(Batch, M, 32)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 32)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64, \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	37,160
	ADD	(Batch, $\lfloor \frac{M}{3} \rfloor$, 32) (Batch, $\lfloor \frac{M}{3} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 32)	0
	MaxPooling	(Batch, $\lfloor \frac{M}{3} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	9,312
	ADD	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	0
	MaxPooling	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	0
	Flattening	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor \times 32$)	0
Latent	*Dense*	(Batch, $\lfloor \frac{M}{9} \rfloor \times 32$)	(Batch, $\lfloor \frac{M}{4} \rfloor$)	$(\lfloor \frac{M}{9} \rfloor \times 32) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$
Decoder	Reshape	(Batch, $\lfloor \frac{M}{4} \rfloor$)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	0
	$\begin{bmatrix} Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128 \\ Conv.\ 1\times 3,\ 128, \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 32)	111,392
	ADD	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 32)	0
	Transp. Conv.	(Batch, $\lfloor \frac{M}{4} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{2} \rfloor$, 32)	3,104
	$\begin{bmatrix} Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 64 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, $\lfloor \frac{M}{2} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{2} \rfloor$, 32)	37,088
	ADD	(Batch, $\lfloor \frac{M}{2} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{2} \rfloor$, 32)	0
	Transp. Conv.	(Batch, $\lfloor \frac{M}{2} \rfloor$, 32)	(Batch, M, 32)	3,104
	$\begin{bmatrix} Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \\ Conv.\ 1\times 3,\ 32 \end{bmatrix}$	(Batch, M, 32)	(Batch, M, 32)	9,312
	Transp. Conv.	(Batch, M, 32)	(Batch, M, 32)	3,104
	ADD	(Batch, M, 32)	(Batch, M, 32)	0
	Flattening	(Batch, M, 32)	(Batch, $M \times 32$)	0
Time Domain	*Dense*	(Batch, $M \times 32$)	(Batch, M)	$(M^2 \times 32) + M$

with ReLU activation functions with a He normal initialization [67]. On the contrary, we initialized the encoder's first *Convolutional* layer and the decoder's *Dense* layers with Glorot uniform [66]. In conclusion, we have summarized the overall parameters of the proposed reduced ResNet architecture as shown in Table 3.12.

3.3.3 Proposed Reduced Inception Version Two Autoencoder

We base our final basic autoencoder architecture on the Inception version two [60]. In this proposal, instead of concatenating stacked *Convolutional* layers, we used Inception modules as shown in Figure 3.17. In Figure 3.17, the basic Inception module is composed of a $\{1 \times 1,\ 1 \times 2,\ 1 \times 3\ and 1 \times 5\}$ *Convolutional* layers. Moreover, these layers get fed with similar inputs or concatenated in parallel. In addition to this, each *Convolutional* layer with a kernel size of $\{1 \times 2\ 1 \times 3\ ,\ and\ 1 \times 5\}$ get their input via *Convolutional* layers that have a $\{1 \times 1\}$ kernel size. We have used this $\{1 \times 1\}$

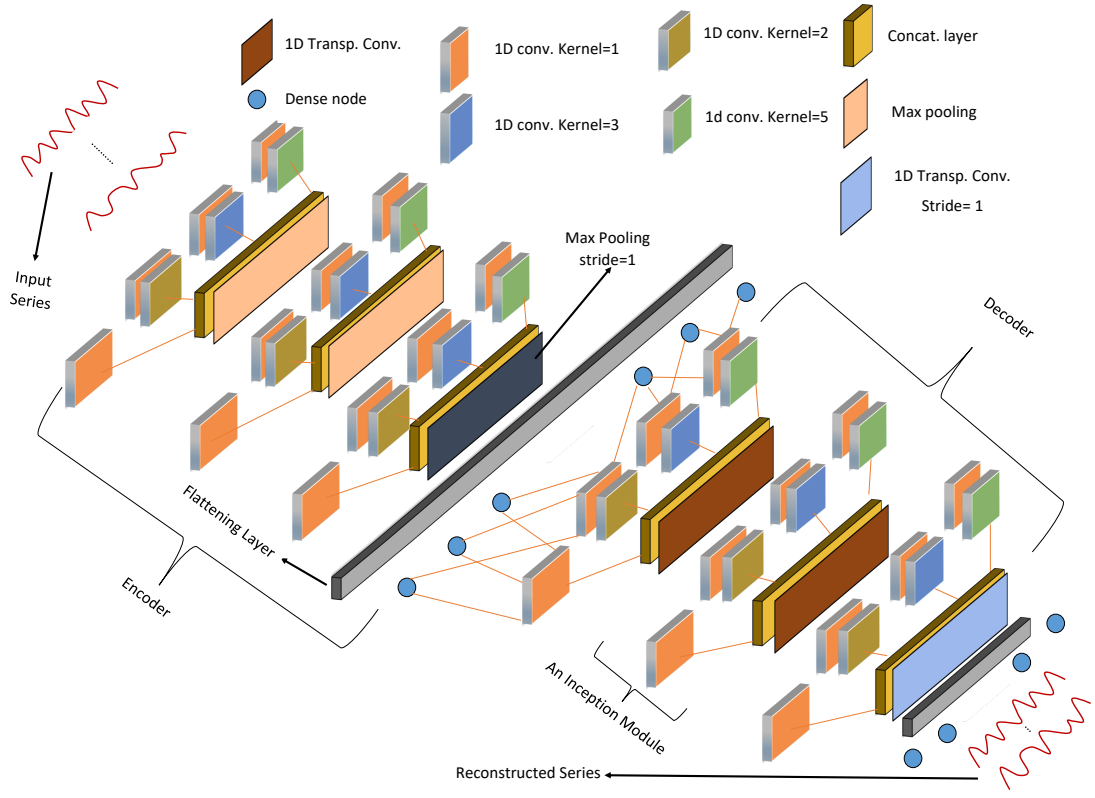


Figure 3.17: Proposed reduced Inception version two autoencoder architecture

Convolutional layers to reshape input features. For instance, the $\{1 \times 1\}$ *Convolutional* kernel at the second inception module of the encoder reshapes a 128 channel input feature map into a 16 channel feature map. This feature reshaping is useful when we concatenate the channels of each *Convolutional* layer within an Inception module. In this aspect, we have arranged each Inception module to output channel sizes equivalent to the reduced VGG16 and ResNet architectures. For instance, at the encoder, we have set each *Convolutional* building block of the first Inception module to output a channel size of 32. Thus, when concatenated, they give a total channel size of $4 \times 32 = 128$. By following this trend, we have set the encoder and decoder Inception modules to output a concatenated channels of

{128, 64, 32}, i.e., as in the case of the modified reduced VGG16 and ResNet architectures as shown in Table 3.13. In reality, we expect this channel matching to help us evaluate the impact of the neural

Table 3.13: Layer configurations for the reduced Inception version two autoencoder

Module	Layer (s)	Input dim.	Output dim.	# parameters
Encoder	Reshape	(Batch, M)	(Batch, M,1)	0
	Inception Module	(Batch, M,1)	-	10,592
	Concatenate	$4 \times (Batch, M, 32)$	(Batch, M, 128)	0
	MaxPooling	(Batch, M, 128)	(Batch, $\lfloor \frac{M}{3} \rfloor$, 128)	0
	Inception Module	(Batch,$\lfloor \frac{M}{3} \rfloor$, 128)	-	10,864
	Concatenate	$4 \times (Batch, M, 16)$	(Batch, M, 64)	0
	MaxPooling	(Batch, $\lfloor \frac{M}{3} \rfloor$, 64)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 64)	0
	Inception Module	(Batch, $\lfloor \frac{M}{9} \rfloor$, 64)	-	2,744
	Concatenate	$4 \times (Batch, M, 8)$	(Batch, M, 32)	0
	MaxPooling	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	0
	Flattening	(Batch, $\lfloor \frac{M}{9} \rfloor$, 32)	(Batch, $\lfloor \frac{M}{9} \rfloor \times 32$)	0
Latent	*Dense*	(Batch, $\lfloor \frac{M}{9} \rfloor \times 32$)	(Batch, $\lfloor \frac{M}{4} \rfloor$)	$(\lfloor \frac{M}{9} \rfloor \times 32) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$
Decoder	Reshape	(Batch, $\lfloor \frac{M}{4} \rfloor$)	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	0
	Inception Module	(Batch, $\lfloor \frac{M}{4} \rfloor$, 1)	-	8,288
	Concatenate	$4 \times (Batch, \lfloor \frac{M}{4} \rfloor, 32)$	(Batch,$\lfloor \frac{M}{4} \rfloor$, 128)	0
	Transp. Conv.	(Batch, $\lfloor \frac{M}{4} \rfloor$, 128)	(Batch, $\lfloor \frac{M}{2} \rfloor$, 128)	49,280
	Inception Module	(Batch, $\lfloor \frac{M}{2} \rfloor$, 128)	-	10,864
	Concatenate	$4 \times (Batch, \lfloor \frac{M}{2} \rfloor, 16)$	(Batch,$\lfloor \frac{M}{2} \rfloor$, 64)	0
	Transp. Conv.	(Batch, $\lfloor \frac{M}{2} \rfloor$, 64)	(Batch, M, 64)	12,352
	Inception Module	(Batch, $\lfloor \frac{M}{2} \rfloor$, 64)	-	10,864
	Concatenate	$4 \times (Batch, M, 8)$	(Batch, M, 32)	0
	Transp. Conv.	(Batch, M, 32)	(Batch, M, 32)	3,104
	Flattening	(Batch, M, 32)	(Batch, $M \times 32$)	0
Time Domain	*Dense*	(Batch, $M \times 32$)	(Batch, M)	$(M^2 \times 32) + M$

architecture on the estimated mean in a more balanced manner. With this understanding, we have also kept the remaining common configurations similar to the VGG16 and ResNet. In this regard, we have set the kernel and stride of the *MaxPooling* layers to 3 and 2. However, similar to the cases of the VGG16 and ResNet, the last *MaxPooling* layer has a stride of 1. Furthermore, the first two and the last transposed *Convolutional* have a stride of two and one respectively. Finally, in the proposed Inception architecture, we have a *Linear* activation functions at the encoder's first $\{(1 \times 1, 1 \times 5)\}$ Inception building block and the decoder's *Dense* layers. For these layers, we have used a *Glorot uniform* layer weight initialization technique [66]. On the other hand, a He normal initialization was used for the remaining ReLU activated layers [67].

3.3.4 Variational Variant of the Basic Autoencoders

In reality, before settling for the final latent representation of an input series, we expect the decoder portion of an autoencoder to see different sets of latent representations for a given input series. Moreover, as the training progresses, on one hand, we expect the encoder to start learning latent representations of an input that are near neighborhoods of its finally allocated latent space representation. On the other hand, for a properly trained autoencoder, we expect the decoder's reconstruction loss to be relatively low for such neighborhood latent representations. In reality, we are relying on this neighborhood interpretation capability of the decoder while projecting the latent space averages to the time domain. This is because the decoder has no prior knowledge of the averaged set's latent space arithmetic mean. However, in the basic autoencoder, we do not either have a way of keeping track of neighborhood latent space representations or a means of influencing neighborhood and finally allocated latent space representations. To this end, we presume the latent space of the basic autoencoder to be discrete, i.e., one-to-one mapping. In practice, there are different versions of the basic autoencoder which could address this limitation, i.e., the Variational AutoEncoder (VAE).

To evaluate this variant of the basic autoencoder, we propose to make minor modifications to the architectures presented in the previous section. In this regard, unlike their basic counterparts, variational autoencoders aim to model the latent space using multivariate Gaussian distribution. To realize this practically, we get expected to reconfigure the encoders to output the mean and the logarithmic variances of a multivariate Gaussian distribution [38]. In reality, the outputted mean and variances are then to be used to generate a sample from the estimated distribution using (3.4), where μ_x, σ_x^2 are the estimated mean and the variance of an input time series X. Moreover, $Z_{\mathcal{N}(0,\ 1)}$ is a random sample taken from a multivariate normal Gaussian distribution ($\mathcal{N}(0,\ 1)$). In practice, this sampling process is commonly called the reparameterization trick since the normal Gaussian distribution is reparametrized by a trainable mean and logarithmic variance. In practice, implementing a VAE without the reparameterization trick would have been infeasible. This is because, we expect the encoder of a VAE to generate outputs that correspond to μ_x and σ_x as shown in Figure 3.18. However, in practice, the decoder is capable of interpreting samples rather than distribution parameters. Thus, with the help of the reparametrization trick, we can introduce a custom sampling layer that takes μ_x and σ_x^2 as an input and generates a sample from a Gaussian distribution using (3.4). Thus, this way, the autoencoder can tune the distribution parameters through backpropagation.

$$Z = \mu_x \ + \ \exp^{(0.5 \times \sigma_x^2)} \ \times Z_{\mathcal{N}(0,\ 1)} \tag{3.4}$$

In reality, the introduction of the normal Gaussian distribution in (3.4) is not there by accident. In this regard, a VAE is intended to be used as a generative autoencoder by modeling the underlying latent space distribution of its inputs. In other words, after a proper training of a VAE autoencoder, its decoder is expected to interpret samples from an underlying latent space distribution in a meaningful way [38]. However, before this point, we are expected to guide the estimated distribution parameters (μ_x and σ_x^2) into a well-formulated one. This is because we can not expect an autoencoder to generalize for distribution parameters estimated from individual datasets. To address this issue, a VAE proposes

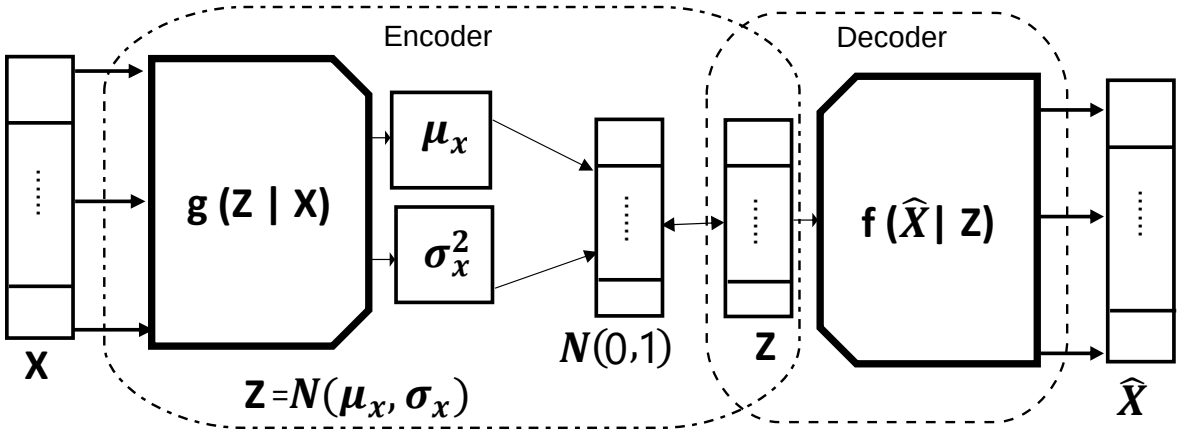


Figure 3.18: Block diagram of a basic varational autoencoder

to utilize Kullback–Leibler (KL) divergence that aims to guide estimated distribution parameters to collectively follow the parameters of a prior distribution often set to the normal Gaussian distribution ($\mathcal{N}(0, 1)$). In practice, to compute the KL divergence, a VAE assumes the individual coordinates of latent space embedding are independent. Furthermore, it also assumes the latent space embedding is mutually independent (they have a diagonal covariance matrix). In other words, a VAE treats a multivariate latent space embedding as a set of mutually independent univariate Gaussian variables. In reality, this assumption gets taken since a VAE encoder can not guarantee the generation of a nonsingular covariance matrix which is useful in the computation of the KL divergence between two multivariate distributions. With this understanding, a VAE propose to computes the KL divergence using (3.5), where $P(Z|X)$ is the distribution learned by a VAE given an input datasets X and its latent space representations Z. To this end, given a set of N time series, the objective functions of a basic VAE becomes (3.6), where $X_i \, \epsilon \, \mathbb{R}^M$ and $Z_i \, \epsilon \, \mathbb{R}^\tau \; : \tau \; < \; M$.

$$KL(P(Z|X)||\mathcal{N}(0,\ 1)) = -0.5 \times (1 + \log^{(\sigma_x^2)} - \mu_x^2 - \exp^{\log(\sigma_x^2)}) \tag{3.5}$$

$$L(X, \hat{X}) = \frac{1}{N}\sum_{i=1}^{N}||X - \hat{X}||_{l2} + \frac{1}{N}\sum_{i=1}^{N} KL(P(Z_i|X_i)||\mathcal{N}(0,\ 1)) \tag{3.6}$$

Architecture wise, we have modified the three architectures given in Figures 3.15, 3.17 and 3.16 in order to account for μ_x and σ_x^2. In this aspect, for the variational variant of these architectures, the output of the encoder's *Flattening* layer is fed to two *Dense* layers representing μ_x and σ_x. Since we now have two *Dense* layers at the encoder, the variational version of the basic autoencoders will have an additional $(\lfloor \frac{M}{9} \rfloor \times 32) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$ trainable parameters. In addition to this change, we have also set the activation function of the two *Dense* layers to *Linear* which was *ReLu* in the non-variational form. We introduce this change since μ_x and σ_x^2 are continuous distribution parameters. Despite these changes, we keep the other layer configurations similar to the one shown in Tables 3.11, 3.12, and 3.13. Overall, with VAE, we are modeling the latent space with a continuous normal Gaussian distribution. Thus, we expect the latent space of VAE to be continuous. To this end, we expect a minor shape change on a reconstructed series due to a minor perturbance in its latent space representation. Hence, we expect the VAE projection of a latent space mean to be better than its basic autoencoder counterpart.

However, in reality, VAE's latent space is cramped into the space of a single multivariate normal Gaussian distribution. To this end, we believe this rigid constraint will have a negative implication on the resolvability of the latent space features, latent averages, and their re-projected estimates. With this concern in mind, we train the variation and non-variational autoencoders with the experimental setups discussed in the next subsection.

3.3.5 Experimental Setup

We have trained the proposed variational and non-variational autoencoders using four $L2$ regularization setups, i.e., $L2 = [0.0, 0.0001, 0.001, 0.01]$ that are uniformly distributed over each autoencoder layer. Moreover, we trained the networks for 600 epochs when they have zero $L2$ regularization. On the contrary, we have used 1500 training epochs when the network gets initialized with the remaining $L2$ regularization setups. However, for each regularization setup, we have used a similar learning rate of 10^{-4}. Furthermore, we have used 80 % of the training datasets for training and the remaining 20% for validation. Moreover, while training, we allow the networks to update their weights after a mini-batch size of $\frac{K}{4}$, where K is the size of a training split. Finally, in the context of repeated trials, we have trained the proposed autoencoders for 25 repeated trials. Thus, overall, a given autoencoder is trained 100 times for each evaluated UCR archive dataset. We then report our experimental evaluations using median, mean, maximum, and minimum NCC accuracies of the 25 repeated trials.

3.3.6 Experimental Results and Interpretation

3.3.6.1 Evaluations for the Basic Autoencoders

We have divided our extended evaluation of the basic autoencoders into two subsections. In this section, we present the experimental outcomes corresponding to the non-variational autoencoders. In this aspect, Table 3.14 presents the wins/ties/losses associated with the proposed VGG16, ResNet and Inception architectures. To evaluate the wins/ties/losses, we have taken the maximum NCC accuracies associated with each averaging technique. However, since we have utilized four L_2 regularization setups while training the proposed autoencoders, we assess the maximum classification accuracies associated with each regularization separately. To this end, in Table 3.14, VGG_Regx_Lat (TD)_Max, Inc_Regx_Lat (TD)_Max, and ResNet_Regx_Lat (TD)_Max corresponds to the wins/ties/losses obtained using the respective autoencoder's latent space (Lat) and time domain (TD) maximum NCC accuracies. Moreover, we indicate the utilized $L2$ regularization using the labels Regx, where $x = \{0,\ 1,\ 2,\ 3\}$ corresponding to $L2 = \{0, 0.0001, 0.001, 0.01\}$. Additionally, the labels VGG, Inc, and ResNet corresponds to the autoencoders based on the modified reduced VGG16, reduced Inception, and reduced ResNet architectures. Table 3.14 further confirms that latent embedding obtained from reconstructable features is not sufficient to obtain good time-domain projections. In this regard, in the latent space, the proposed architectures that were based on VGG16, Inception, and ResNet architectures aggregately won on 12 datasets. However, in the time domain, none of the autoencoders were able to generate estimates that could win a NCC. However, if we compare the latent space NCC outcomes of modified reduced VGG16 that is shown in Figure 3.15 and its counterpart shown in Figure 3.9, the former was able to win on 6 out of the 74 UCR archive dataset while the latter won on 5. On the contrary,

Table 3.14: Win/tie/losses analysis of NCC classification accuracies obtained from the extended evaluation of basic autoencoders.The analysis was performed using 74 UCR archive datasets, averages estimated via autoencoder and different averaging techniques, and NCC accuracies. Moreover, the outcomes reported for the autoencoders are generated using maximum NCC accuracies obtained from 25 repeated trials and four L2 regularization setups.

Techniques	*L2* Reg. (x)	Wins	Ties	Losses
Arithmetic	-	0	0	74
DBA		3	4	67
DTAN		35	3	36
SDBA		17	4	53
VGG_Regx_Lat_Max	{0, 1, 2, 3}	{2, 1, 3, 0}	{2, 2, 0, 0}	{70, 71, 71, 74}
VGG_Regx_TD_Max		{0, 0, 0, 0}	{0, 0, 0, 0}	{74, 74, 74, 74}
Inc_Regx_Lat_Max		{2, 0, 0, 0}	{1, 1, 0, 0}	{71, 73, 74, 74}
Inc_Regx_TD_Max		{0, 0, 0, 0}	{0, 1, 0, 0}	{74, 73, 74, 74}
ResNet_Regx_Lat_Max		{1, 0, 1, 2}	{2, 2, 1, 0}	{71, 72, 72, 72}
ResNet_Regx_TD_Max		{0, 0, 0, 0}	{0, 0, 0, 0}	{74, 74, 74, 74}

the latter won on 1 of the 74 UCR datasets. However, the former could not win on any of them. This, in turn, implies the significance the minor architectural changes have on the quality of the latent space embedding and re-projected latent estimates. To further validate this remark, we will consecutively evaluate the statistics of the NCC using box-whisker plots and hypothesis tests. In this aspect, Table 3.15 summarizes the statistics for the NCC accuracies obtained with the estimates of proposed autoencoders and alternative averaging techniques.

According to Table 3.15, DTAN achieved the best overall statistics, i.e., in terms of bottom whisker, Top Whisker, 25% and 75% quantiles, and median accuracies. In the context of the autoencoders, the reduced ResNet autoencoder achieved better latent space statistics compared to its counterparts. In this regard, in the worst case, the best NCC accuracy the reduced ResNet obtained is 21.79%. On the contrary, in the best case, it obtained a 100% NCC accuracy. In reality, this performance of the reduced ResNet gets seconded by the modified reduced VGG16. In general, when we compare the time domain performance of this architecture with the one shown in Figure 3.9, the modified reduced VGG16 obtained a 12.78% worst case classification accuracy. On the contrary, on the 74 UCR archive datasets, the unmodified version of the architecture obtained a 10.50% worst-case NCC accuracy. Moreover, on the remaining statistical parameters, the modified reduced VGG16 obtained the following best NCC accuracies: top whisker 96.72%, median 54.70%, and finally 50% of its classification accuracies are within 44.11% and 64.58%. On the contrary, in the time domain, the unmodified version obtained the following best NCC classification accuracies: top whisker 95%, median 50%, and finally 50% of its classification accuracies are within 40.01% and 62.58%. Contrary to the wins/ties/losses analysis, the underlying statistics for the NCC accuracies reveals that the architectural improvement has a positive implication on the projected latent means. In general, we have graphically summarized the statistical parameters shown in Table 3.15 in Figure 3.19. The plot also shows that architectures based on the modified reduced VGG16 and ResNet for the autoencoder-based estimations are better suited for the task at hand. In this context, the *Convolutional* layer stacking in the modified reduced VGG16 and

Table 3.15: Statistics assessment of the NCC accuracies that are obtained with the modified VGG16, reduced Inception, and reduced ResNet architectures. These assessments were conducted using the maximum NCC accuracies associated with each averaging technique.

Techniques	Bot. whisker	Top whisker	25% Quant.	75% Quant.	Median
Arithmetic	7.47	96.43	40.08	68.09	53.09
DBA	28.94	100	55.13	79.09	65.49
DTAN	33.31	100	59.31	85.62	74.30
SDBA	32.83	99.05	58.72	81.07	69.79
VGG_Regx_Lat_Max x={0, 1, 2, 3}	{20.86, 14.58 16.30, 8.29 }	{100, 100 99.05, 95.24}	{50.87, 50.04 46.86, 38.94}	{74.64, 75.99 74.82, 66.85 }	{62.29, 62.58 58.42, 51.63}
VGG_Regx_TD_Max x={0, 1, 2, 3}	{17.99, 17.78 10.24, 8.08}	{90.07, 83.96 96.72, 90.36 }	{43.63, 44.11 39.90, 31.65 }	{63.99, 64.58 63.61, 56.95}	{54.70, 52.04 50.19, 47.79}
Inc_Regx_Lat_Max x={0, 1, 2, 3}	{16.30, 17.87 16.61, 8.29 }	{100, 99.05 99.04, 92.38 }	{50.16, 50.27 48.95, 37.93}	{75.13, 74.64 71.28, 65.03}	{62.02, 60.06 59.05, 51.47}
Inc_Regx_TD_Max x={0, 1, 2, 3}	{16.87, 18.33 9.55, 7.55}	{98.74, 83.45 96.14, 87.50}	{44.44, 43.18 41.07, 31.85}	{66.45, 64.05 63.57, 56.05}	{52.61, 52.44 50.63, 47.04}
ResNet_Regx_Lat_Max x={0, 1, 2, 3}	{21.79, 18.97 16.93, 18.13}	{100, 100 100, 94.29}	{51.19, 50.46 49.52, 43.50}	{74.61, 73.83 73.11, 65.75}	{62.51, 61.81 59.93, 57.24}
ResNet_Regx_TD_Max x={0, 1, 2, 3}	{12.78, 18.15 10.39, 7.34}	{96.14, 88.20 96.43, 91.20}	{42.50, 43.99 41.77, 36.27}	{63.97, 63.84 64.61, 60.39}	{51.42, 50.71 50.99, 49.39}

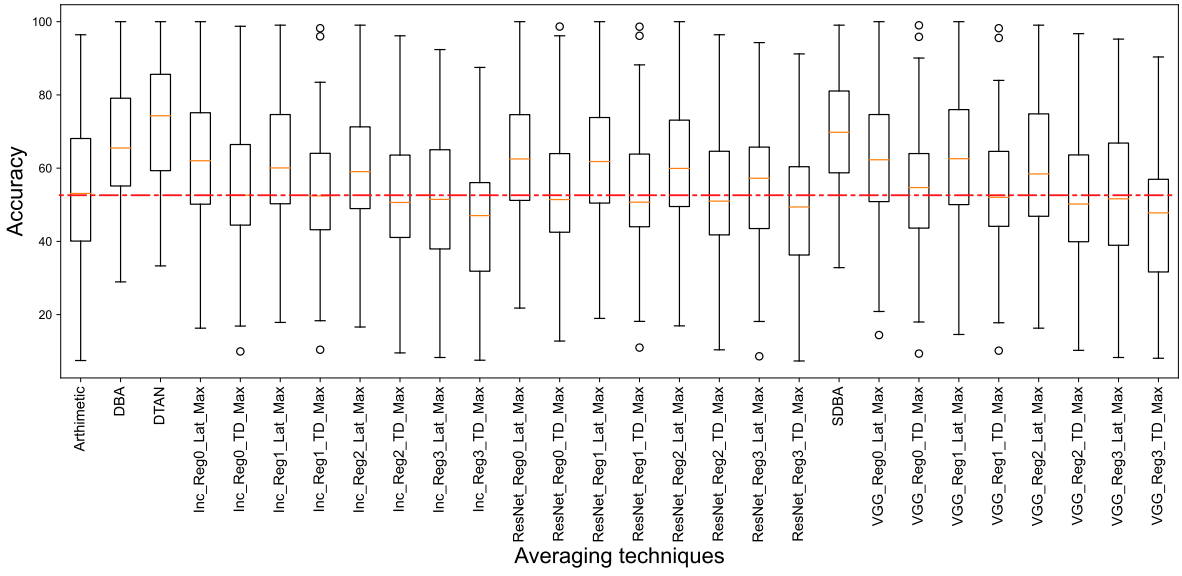
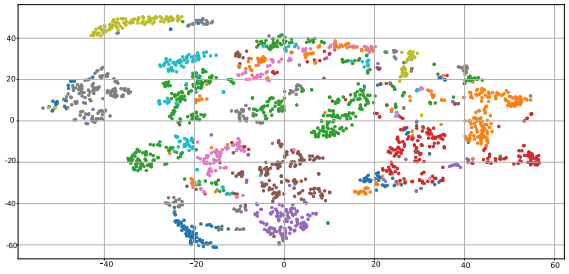


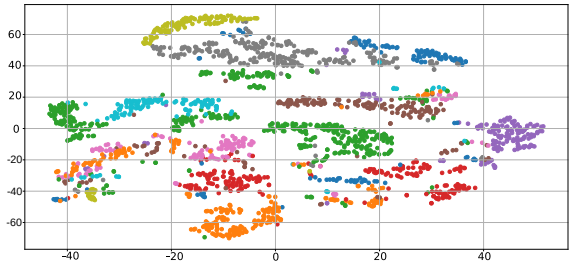
Figure 3.19: Box-whisker plot analysis of the NCC accuracies obtained with the modified reduced VGG16, reduced Inception, and reduced ResNet architectures

ResNet has a positive impact on the separability and the compactness of the latent embedding as shown in Figure 3.20. This,in turn, helps the decoder to better re-project latent means which in such cases are most likely in the neighborhood of the latent embedding of the training datasets. In order to plot the t-SNE projections shown in Figure 3.20, we identified and selected the case where the autoencoders performed better in the latent space NCC classification. In general, when we compare Figure 3.20 (a)

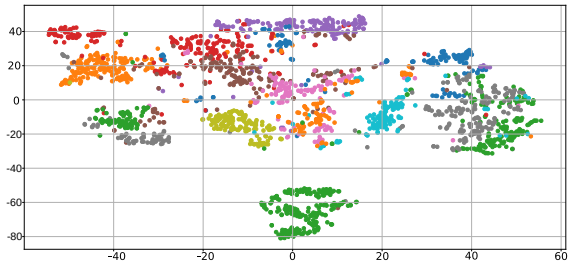
and 3.20 (b), we can see that the modified reduced VGG16 has better density as compared to its predecessor. The density of the latent features further increases in the reduced ResNet and Inception architectures. However, in the case of the reduced Inception autoencoder, the latent embeddings are overlapping. This, in turn, has a negative implication on the distinguishability of the projected latent mean. We associate this behavior of the reduced Inception architecture with the limited number of *Convolutional* layers that input gets passed through before generating the latent embedding. In this aspect, the reduced Inception passes input through 6 *Convolutional* layers before generating its latent embedding. On the contrary, the reduced and modified reduced VGG16 architectures passes input through 9 and 11 *Convolutional* layers. The addition *Convolutional* layers are in turn expected to help the two architectures further refine the latent features. However, despite this advantage, it



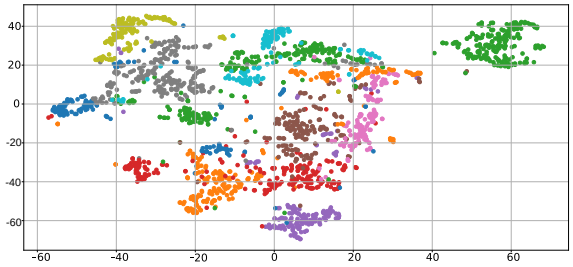
(a) *FacesUCR* in the latent space of the reduced VGG16 autoencoder



(b) *FacesUCR* in the latent space of the modified reduced VGG16 autoencoder



(c) *FacesUCR* in the latent space of the reduced ResNet autoencoder



(d) *FacesUCR* in the latent space of the reduced Inception autoencoder

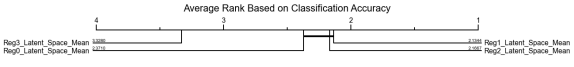
Figure 3.20: t-SNE projections for the UCR archive's *FacesUCR* test datasets in the latent spaces of: (a) reduced VGG16, (b) modified reduced VGG16, (c) reduced ResNet, and (d) reduced Inception

should also get noted that the reduced Inception obtains the presented performance with a smaller number of training parameters. To this end, we aim to reuse this architecture in our subsequent investigations. With this said, we next evaluate the stability of the proposed architectures. In this regard, we assess the average standard deviation (σ) of the NCC accuracies obtained from 25 repeated training trials conducted on 89 UCR archive datasets. The summary for the standard deviation is shown in Table 3.16. According to Table 3.16, the modified reduced VGG16's time domain minimum average standard deviation (VGG_TD) is 3.44%. In this regard, the reduced Inception (Inc_TD) and ResNet (ResNet_TD) obtained 3.67% and 3.43%. Overall, the average standard deviation for the time domain and latent space NCC accuracies are well below 7% and 6%. These results tells us that we could have trained the network for a limited number of trials and could have observed relatively similar outcomes. In other words, variations arising due to the randomness of the optimization setup are relatively low. This, in turn, indicates the reproducibility of our experimental evaluations.

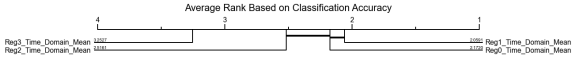
Table 3.16: Standard deviation across the NCC accuracies that are obtained using: modified reduced VGG16, reduced Inception, and reduced ResNet autoencoders.

Techniques	$\pm\sigma$ in % L2 Reg0	$\pm\sigma$ in % L2 Reg1	$\pm\sigma$ in % L2 Reg2	$\pm\sigma$ in % L2 Reg3
VGG_Lat	3.86	4.00	3.44	4.52
Inc_Lat	3.50	3.75	3.67	4.74
ResNet_Lat	5.17	4.26	4.35	3.43
VGG_TD	5.39	5.75	5.19	4.91
Inc_TD	5.49	5.47	4.97	5.09
ResNet_TD	6.36	6.00	5.78	5.58

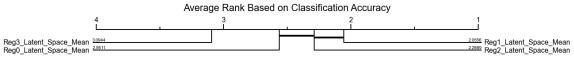
With this said, we will assess which of the $L2$ regularization is better. To make the assessment, we conduct a hypothesis test on the mean latent space and time domain NCC accuracies. In this regard, Figures 3.21 (a) and 3.21 (b) shows that for the modified reduced VGG16 autoencoder, the first three $L2$ regularization setups ($L2 = [0, 0.001, 0.001]$) resulted with estimates that generated a statistically indifferent NCC accuracies. In reality, this is also valid for the reduced Inception and



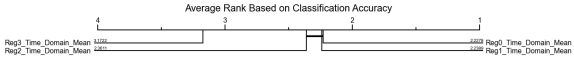
(a) Evaluation using NCC mean accuracies obtained in the latent space of the modified reduced VGG16 autoencoder



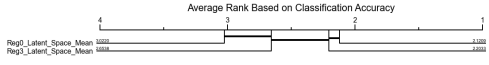
(b) Evaluation using NCC mean accuracies obtained with time domain estimates of the modified reduced VGG16 autoencoder



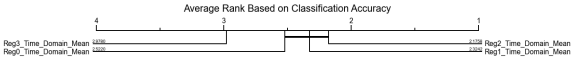
(c) Evaluation using NCC mean accuracies obtained in the latent space of the reduced Inception autoencoder



(d) Evaluation using NCC mean accuracies obtained with time domain estimates of the reduced Inception autoencoder



(e) Evaluation using NCC mean accuracies obtained in the latent space of the reduced ResNet autoencoder

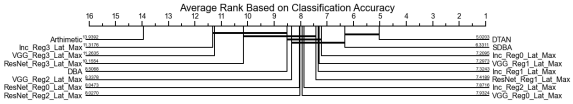


(f) Evaluation using NCC mean accuracies obtained with time domain estimates of the reduced ResNet autoencoder

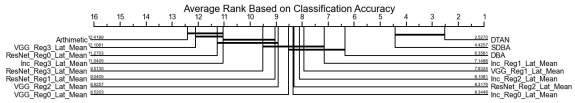
Figure 3.21: Evaluation of the impact of L2 regularization on the quality of means estimated with basic autoencoders

ResNet autoencoders. In general, we can safely assume that for all proposed autoencoders, the first three regularization setups are statistically indifferent. However, if the slightest improvement gets desired, we suggest the utilization of the second $L2$ regularization (L2=0.001). We will conclude our extended analysis of the autoencoders with their hypothesis tests divided into two categories. First, we compare the performance of the proposals among themselves and their counterparts using 74 UCR archive datasets. In this comparison, we include the experimental outcomes of DTAN. However, in the second comparison, we exclude DTAN and compare the remaining techniques using 89 UCR archive datasets. In this context, Figure 3.22 shows the hypothesis test for the NCC accuracies obtained in the latent spaces of the autoencoders. Overall, in the latent space, the Inception architecture obtains a better Friedman average rank compared to the alternative autoencoder proposals. Moreover, if we observe the evaluations based on mean and median NCC accuracies, i.e., Figures 3.22 (a) and 3.22 (b), we can see that the estimates of the modified reduced VGG16 and reduced ResNet autoencoders are at

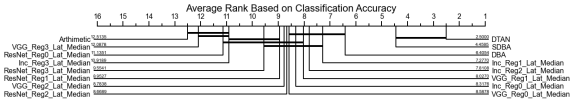
time behaving as an arithmetic mean. Even though the spaces from which the arithmetic and latent means get estimated are different, this observation further supports our previous argument that the latent space of a basic autoencoder can not overcome the effects of temporal distortion. Moreover, this further presents evidence as to why the time domain projections of the reduced VGG16 autoencoder behaves as arithmetic mean.



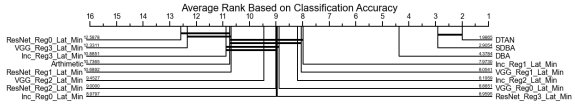
(a) Evaluation of latent space maximum NCC as compared to alternatives



(b) Evaluation of latent space mean NCC as compared to alternatives



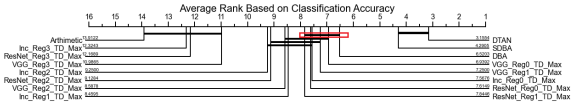
(c) Evaluation of latent space median NCC as compared to alternatives



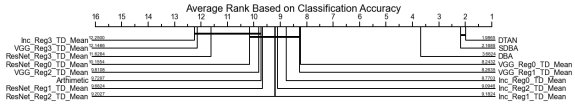
(d) Evaluation of latent space minimum NCC as compared to alternatives.

Figure 3.22: Evaluation of latent space NCC accuracies that are obtained using the modified reduced VGG16, reduced Inception, and reduced ResNet. These evaluations were conducted using 74 UCR archive datasets.

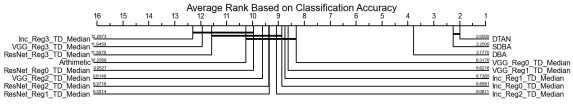
In reality, the performance similarity between the arithmetic means and their autoencoder counterparts is also evident in the time domain. In this aspect, Figure 3.23 (b) shows that an arithmetic mean is in average better than the autoencoder's estimate on some training configurations. Moreover, Figure 3.23 (c) shows that the Inception autoencoder's estimates are behaving as an arithmetic mean when the network is regularized with $L2 = [0.01]$. Furthermore, in the worst case, the estimates of the autoencoders are worst than an arithmetic mean, i.e., Figure 3.23 (d). However, one interesting point



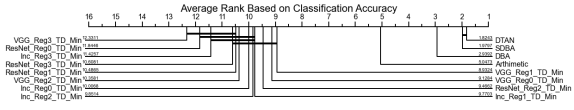
(a) Evaluation of latent space and time domain maximum NCC as compared to alternatives



(b) Evaluation of latent space and time domain mean NCC as compared to alternatives



(c) Evaluation of latent space and time domain median NCC as compared to alternatives

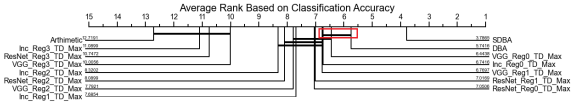


(d) Evaluation of latent space and time domain minimum NCC as compared to alternatives.

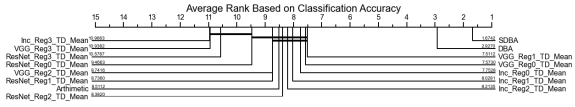
Figure 3.23: Evaluation of latent space and time domain NCC accuracies that are obtained using the modified reduced VGG16, reduced Inception, and reduced ResNet. These evaluations were conducted using 74 UCR archive datasets.

to note here is that, in Figure 3.23 (a), the reduced ResNet architecture is performing in a statistically indifferent manner as compared to DBA. This happens when the architecture is regularized with $L2 = [0.001]$. Given the fact that the figure compares maximum NCC accuracies of the different averaging techniques, we find it to be quite impressive for two main reasons. First, the time domain classification is performed in DTW space where DBA is relatively favored as compared to the estimates of the autoencoders. This is because the estimates of DBA are generated through DTW warping. Thus, while performing the NCC classification using DTW, we are basically transforming the estimates

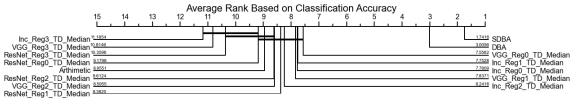
of DBA into their registered space. On the contrary, for the estimates of the autoencoders, this is not valid. In addition to this, the reduced ResNet is able to achieve this performance by only using reconstruction loss. This further motivates us to dig deeper into ways which could improve the time domain projections. However, before proceeding to those discussions, we will present the evaluation of the time domain classifications which includes additional 25 UCR archive datasets. According to Figure 3.24 (a), the time domain NCC accuracies obtained with the estimates of the modified reduced VGG16 shows a statistical indifference to DBA's outcomes in the post hypothesis test. This happens when the architecture is trained with zero $L2$ regularization. In general, the extended evaluation reveals that minor modification on the arrangement of network layers and the architecture of the network have a positive implication on the time domain estimates. In this aspect, the VGG16 based autoencoder appears to be providing the best outcome.



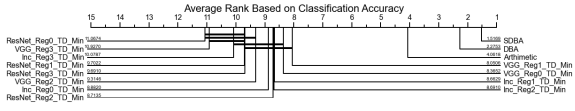
(a) Evaluation of time domain maximum NCC as compared to alternatives



(b) Evaluation of time domain mean NCC as compared to alternatives



(c) Evaluation of time domain median NCC as compared to alternatives



(d) Evaluation of time domain minimum NCC as compared to alternatives.

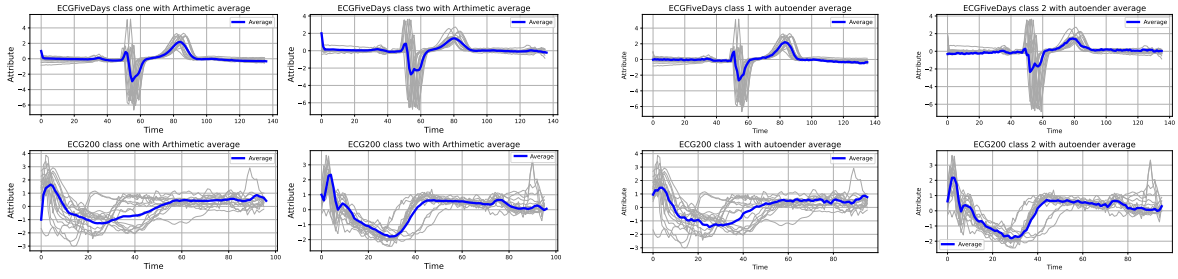
Figure 3.24: Evaluation of time domain NCC accuracies that are obtained using: the modified reduced VGG16, reduced Inception, and reduced ResNet. These evaluations were conducted using 89 UCR archive datasets.

With these observations in mind, we will finalize discussion in this section by presenting the time domain estimates of the UCR archive's *ECG200* and *ECGFiveDays* datasets as a visual demonstration. Figure 3.25 demonstrates the estimates generated by the various averaging techniques and the proposed autoencoders. Moreover, Table 3.17 shows the NCC accuracies obtained with the estimated

Table 3.17: NCC accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets.

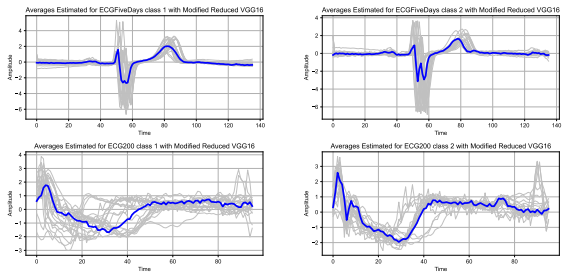
Techniques	NCC for *ECG200* in %	NCC for *ECGFiveDays* in%
Arithmetic	67	52.96
DBA	65	52.15
SDBA	73	67.02
DTAN	**79**	**97.79**
Enc_Time	65	52.15
VGG_TD	**76**	**75.61**
Inc_TD	76	71.54
ResNet_TD	**76**	**72.24**

averages. In the table, we have marked the top three NCC classification accuracies. In this regard, DTAN, the modified reduced VGG16 and reduced ResNet provided estimates that achieved higher NCC accuracies. This in turn implies these estimates were able to capture the dominant descriptive

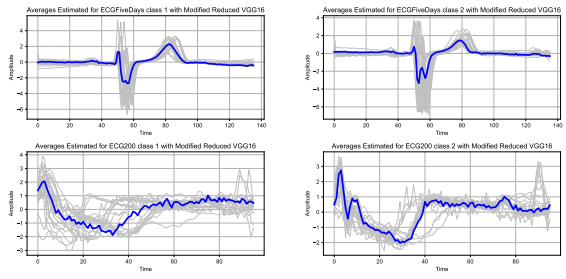


(a) Arithmetic means

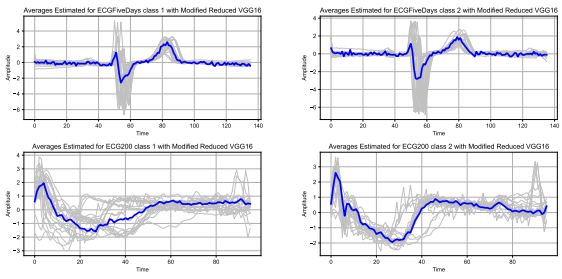
(b) Reduced VGG16 estimates



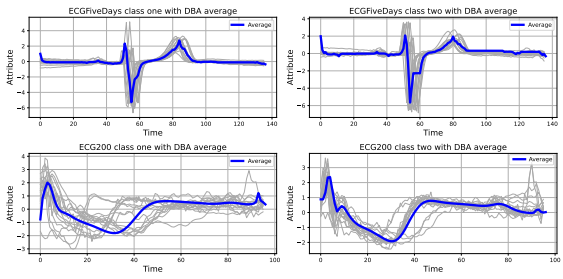
(c) Modified reduced VGG16 estimates



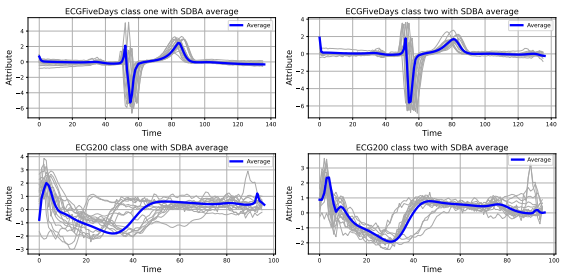
(d) Reduced Inception estimates



(e) Reduced ResNet estimates



(f) DBA estimates



(g) SDBA estimates

Figure 3.25: Averages estimated for the UCR archives *ECG200* AND *ECGFiveDays* datasets using: the modified reduced VGG16, reduced Inception, reduced ResNet, and alternative averaging techniques

shapes observed in the averaged set. In general, the minor and major modifications we introduced on the architecture of the reduced VGG16 has contributed positively to the overall estimation process. However, we believe there are still some remaining issues that needed to be addressed. For instance, for the *ECGFiveDays* dataset, we can observe that the estimates of the modified reduced VGG16 and reduced Inception has introduced two negative peaks on class 2. In reality, this has helped the estimates to obtain better NCC accuracies as compared to their DTW based counterparts. This is because, DTW can now pair one of the two peaks to different shifted versions in the test datasets while conducting the NCC classification. In reality, for the class 2 of the *ECGFiveDays*, DBA's estimate

is also affected in a similar manner. However, in this case, one of the peaks is clipped and the other one is relatively large. In general, despite the positive implication of the two peaks on the NCC accuracy of the autoencoder's estimates, we consider this to be a significant shape distortion. With this observation in mind, we will next proceed to observe if the variational version of the autoencoders address this issue.

3.3.6.2 Evaluations for the Variational Versions of the Basic Autoencoders

Following the same trend, we start our evaluation of the variational autoencoders using wins, ties, and losses analysis. However, before proceeding to the analysis, we would like to mention that out of the 88 UCR archive datasets used for the evaluation of the variational versions of the autoencoders, the overall training process failed to converge on 5 of them. We have summarized these datasets in Table 3.18. The main reason behind the convergence problem lies in the nature of the datasets. In this aspect, unlike most of the UCR datasets, the amplitude values of the datasets are often well above 500. However, this is contrary to the underlying assumption of variational autoencoders that try to model a dataset using a normal Gaussian distribution. Under such an assumption, we expect amplitude values to be between zero and one. However, when this is not the case, the KL divergence loss of the variational autoencoders explodes and fails to converge. This is because the variance of the input datasets becomes significantly large. In reality, we could deploy batch normalization layers that could significantly reduce the data variance. However, we avoided this approach for two main reasons. First, introducing batch normalization layers will further constrain the latent space we could explore. In other words, it will crunch everything into a smaller region. Secondly, we have not utilized batch normalization layers in the proposed basic autoencoders. Thus, the utilization of batch normalization will make the comparison unfair.

Table 3.18: List of UCR archive datasets on which the variational autoencoders failed to converge

Datasets	Classes	# of training sets	# of test sets	Length
ChinaTown	2	20	343	24
GunPointAgeSpan	2	135	316	150
GunPointMaleVersusFemale	2	135	316	150
GunPointOldVersusYoung	2	136	315	150
MealbournPedesterian	10	1194	2439	24

With this in mind, we put aside the comparison for the mentioned datasets and proceed to present the evaluations for the variational autoencoders using the remaining 82 UCR archive datasets. In this regard, Table 3.19 summarizes the wins/ties/losses associated with the variational modified reduced VGG16, reduced Inception and reduced ResNet autoencoders. In general, we observe minor improvements in the performances of the reduced Inception and ResNet architectures compared to the performances of their non-variational counterparts. In this context, in the latent space, the variational versions of both architectures can win on one additional dataset. On the contrary, the performance of the modified reduced VGG16 significantly dropped from winning on six datasets to one in its variational form. However, since an estimate might lose or win by a smaller margin, we

further analyzed the statistics of the NCC accuracies if the performance improvement or degradation has a significant statistical meaning.

Table 3.19: Win/tie/losses analysis of NCC classification accuracies obtained from the extended evaluation of basic autoencoders.The analysis was performed using 73 UCR archive datasets, averages estimated via autoencoder and different averaging techniques, and NCC accuracies. Moreover, the outcomes reported for the autoencoders are generated using maximum NCC accuracies obtained from 25 repeated trials and four L2 regularization setups.

Techniques	*L2* Reg. (x)	Wins	Ties	Losses
Arithmetic	-	0	0	73
DBA		2	5	66
DTAN		33	3	37
SDBA		16	4	53
Var_VGG_Regx_Lat_Max	{0, 1, 2, 3}	{1, 0, 0, 0}	{3, 1, 2, 0}	{69, 72, 71, 73}
Var_VGG_Regx_TD_Max		{0, 0, 0, 0}	{0, 0, 0, 0}	{73, 73, 73, 73}
Var_Inc_Regx_Lat_Max		{1, 1, 0, 2}	{3, 1, 1, 1}	{69, 71, 72, 70}
Var_Inc_Regx_TD_Max		{0, 0, 0, 0}	{0, 0, 0, 0}	{73, 73, 73, 73}
Var_ResNet_Regx_Lat_Max		{3, 3, 0, 0}	{3, 4, 2, 0}	{67, 66, 71, 73}
Var_ResNet_Regx_TD_Max		{0, 0, 0, 0}	{0, 0, 0, 0}	{73, 73, 73, 73}

We first base our statistical analysis on a box-whisker plot. In this regard, Figure 3.26 shows the box-whisker plot, whereas Table 3.20 shows the statistics of the plot. Based on the results reported on Tables 3.15 and 3.20, we observe slight improvements on the NCC accuracies of the variational autoencoders. For instance, in the latent space of the variational modified reduced VGG16 autoencoder, we obtained a best case worst NCC classification accuracy (best case bottom whisker) of 21.16%. Moreover, in the best cases, 50% of the NCC classification accuracies are between 51.15% and 73.33%. In this regard, the non-variational autoencoder's NCC accuracies are between 50.87% and 74.64%. Moreover, its best case worst NCC classification accuracy is 20.86%. Thus, in this case, modeling the

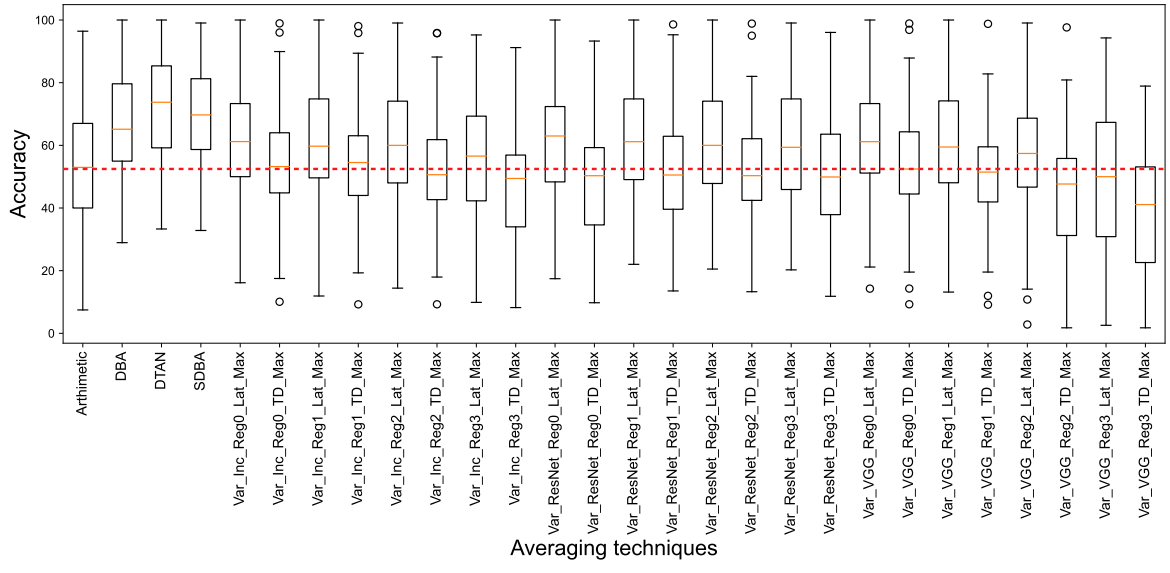


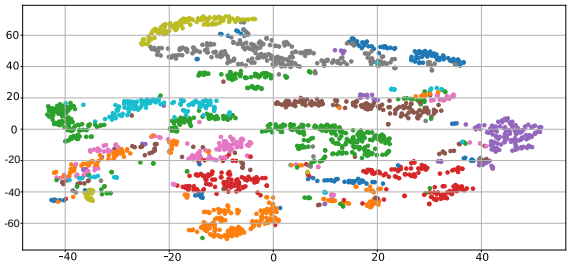
Figure 3.26: Box-whisker plot analysis of the NCC accuracies obtained with the variational: modified reduced VGG16, reduced Inception, and reduced ResNet architectures

latent space via a continuous distribution introduced minor improvements. In reality, such slight performance improvements are also evident in the time domain. Specifically, for the worst case scenario (bottom whiskers), the variational autoencoder is better than its basic counterpart.

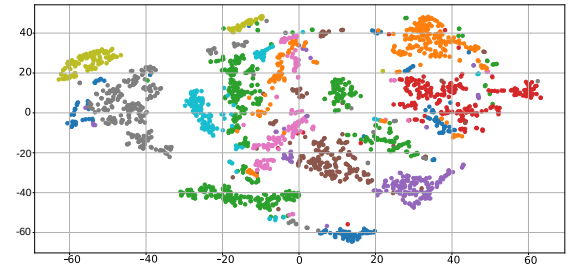
Table 3.20: Statistics assessment of the NCC accuracies that are obtained with the modified VGG16, reduced Inception, and reduced ResNet architectures. These assessments were conducted using the maximum NCC accuracies associated with each averaging techniques.

Techniques	Bot. whisker	Top whisker	25% Quant.	75% Quant.	Median
Arithmetic	7.47	96.43	40.00	67.00	52.96
DBA	28.94	100	54.96	79.65	65.14
DTAN	33.31	100	59.20	85.37	73.75
SDBA	32.83	99.05	58.67	81.27	69.79
VGG_Regx_Lat_Max x={0, 1, 2, 3}	{21.16, 13.17 14.11, 2.56 }	{100, 100 99.05, 94.29}	{51.15, 48.05 46.67, 30.86}	{73.33, 74.19 68.67, 67.33 }	{61.17, 59.46 57.41, 47.66}
VGG_Regx_TD_Max x={0, 1, 2, 3}	{19.55, 19.55 1.76, 1.76}	{87.87, 82.76 80.87, 78.91 }	{44.47, 41.93 31.22, 22.59 }	{64.32, 59.53 55.82, 53.12}	{52.45, 51.47 47.66, 41.09}
Inc_Regx_Lat_Max x={0, 1, 2, 3}	{16.14, 11.91 14.42, 9.87 }	{100, 100 99.05, 95.23 }	{50.00, 49.61 48.00, 42.29}	{73.33, 74.82 74.10, 69.32}	{61.20, 59.73 60.00, 56.57}
Inc_Regx_TD_Max x={0, 1, 2, 3}	{17.50, 19.29 17.96, 8.22}	{89.93, 89.40 88.19, 91.20}	{44.82, 44.00 42.67, 33.98}	{64.01, 63.07 61.82, 56.89}	{53.23, 54.52 50.62, 49.43}
ResNet_Regx_Lat_Max x={0, 1, 2, 3}	{17.43, 22.02 20.51, 20.26}	{100, 100 100, 99.05}	{48.33, 49.07 47.83, 45.90}	{72.38, 74.82 74.10, 74.82}	{62.98, 61.17 60.00, 59.38}
ResNet_Regx_TD_Max x={0, 1, 2, 3}	{19.55, 19.55 1.76, 1.76}	{87.86, 82.79 80.87, 78.91}	{44.47, 41.93 31.22, 22.59}	{64.32, 59.53 55.81, 53.12}	{50.30, 50.54 50.32, 49.90}

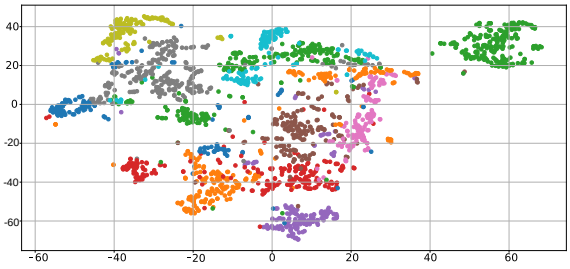
Even though the improvements are not significantly large, it is in line with our initial expectation of variational autoencoders. With this in mind, we next compare the t-SNE projections of the UCR archive's *FacesUCR* dataset. In Figure 3.27, the three subfigures in the left column correspond to the t-SNE projections of the latent embedding obtained with the non variational autoencoders. In reality, we do not observe a significant difference in the separability and compactness of the latent embedding. However, when we compare the latent space NCC accuracies that correspond to the presented embedding, the non-variational versions of the autoencoder performed better. In this regard, in the latent space, the non variational VGG16, Inception and ResNet based autoencoders respectively obtained 65.02%, 71.12% , and 67.90% NCC accuracies. On the contrary, their variational version obtained 61.22%, 68.43%, and 68.68% NCC accuracies. On the contrary, in the time domain, estimates generated by the variational versions of the basic autoencoders performed better than those generated by their basic counterparts. In this regard, the variational VGG16, Inception, and ResNet respectively obtained 60.98%, 59.07%, and 67.90% NCC accuracies. However, their non variational counterparts obtained 48.53%, 49.56%, and 50.49% NCC accuracies. We can correlate the significant improvement in the time domain accuracies to two design factors. First, in the variational setup, the latent space is confined to a relatively smaller region that meets the requirements of a normal Gaussian distribution. This, in turn, reduces the burden on the decoder since it is now expected to interpret a relatively smaller region of the latent space. In addition to this, in each training epoch, the decoder



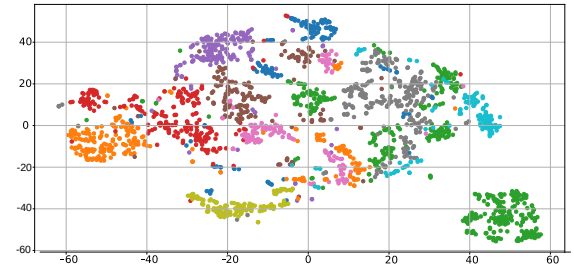
(a) *FacesUCR* in the latent space of the reduced VGG16 autoencoder



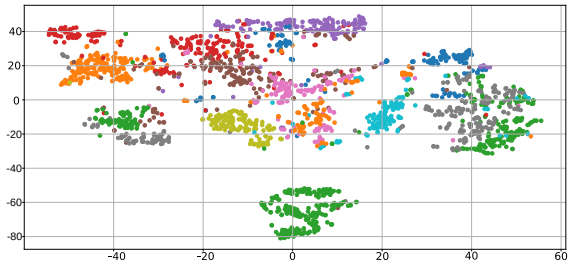
(b) *FacesUCR* in the latent space of the variational modified reduced VGG16 autoencoder



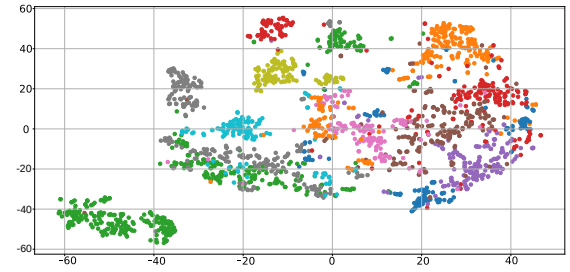
(c) *FacesUCR* in the latent space of the reduced Inception autoencoder



(d) *FacesUCR* in the latent space of the variational reduced Inception autoencoder



(e) *FacesUCR* in the latent space of the reduced ResNet autoencoder

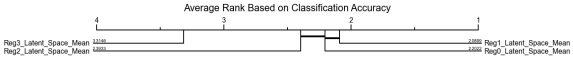


(f) *FacesUCR* in the latent space of the variational reduced ResNet autoencoder

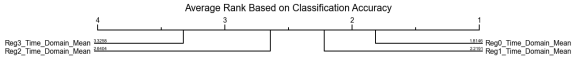
Figure 3.27: Comparison of the latent embedding obtained with the variational and non variational autoencoders for the UCR archive's *FacesUCR* test dataset

gets the chance to see a different interpretation of the input dataset due to the reparametrization trick. Moreover, at the end of the overall training process, we expect the latent space to converge to a normal Gaussian distribution. Thus, in the final few epochs, the decoder will start to see different representations of the input dataset that are samples taken from a normal Gaussian distribution. Thus, the decoder of the variational autoencoders is expected to better reconstruct the latent means that are by themselves samples taken from a normal Gaussian distribution. In reality, while training, the decoders of the basic autoencoder also see different latent space representations of input datasets. However, the overall scenario is different from the variational versions. In this aspect, there is a higher chance that the different representations are from different regions of the latent space. Moreover, at the end of the training, a latent space region that once was occupied by a certain group of input datasets could finally represent different groups of the input. To this end, the decoder could end up with blind spots due to the continuously shifting location of the latent embedding that is by no means constrained to a certain region. This in turn is expected to harm the quality of the re-projected means.

However, in the latent space, the relatively relaxed constraints of the non-variational autoencoder have helped them to obtain better performance. In general, since we can not make conclusive remarks based on the outcomes of a single experiment, we next place our focus on the hypothesis test to further assess this claim. In this aspect, we first evaluate the impact of $L2$ regularization on the latent space and time domain performance, i.e., $L2 = [0, 0.0001, 0.001, 0.01]$. In this regard, Figure 3.28 shows the first two $L2$ regularizations often gave better time domain results when used with the variational modified reduced VGG16 and Inception autoencoders. However, for the ResNet architecture, the two middle $L2$ regularizations performed better. These outcomes suggest that the variational autoencoders give better time domain re-projections while regularized with $L2 = [0.0001]$. Moreover, since there is often no clear statistical demarcation among the first three $L2$ regularizations, i.e., in the latent space, we propose the utilization of $L2 = [0.0001]$.

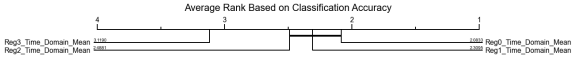


(a) Evaluation using NCC mean accuracies obtained in the latent space of the variational modified reduced VGG16 autoencoder

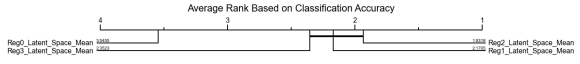


(b) Evaluation using NCC mean accuracies obtained with time domain estimates of the variational modified reduced VGG16 autoencoder

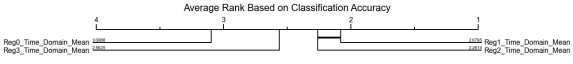
(c) Evaluation using NCC mean accuracies obtained in the latent space of the variational reduced Inception autoencoder



(d) Evaluation using NCC mean accuracies obtained with time domain estimates of the reduced Inception autoencoder



(e) Evaluation using NCC mean accuracies obtained in the latent space of the reduced ResNet autoencoder



(f) Evaluation using NCC mean accuracies obtained with time domain estimates of the reduced ResNet autoencoder

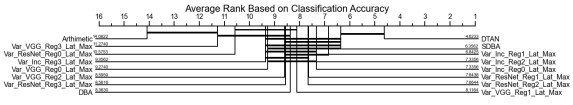
Figure 3.28: Evaluation of the impact of L2 regularization on the quality of means estimated with variational autoencoder

With this at hand, we will next assess the NCC accuracies in the context of reproducibility. To make this assessment, we evaluate the average standard deviation (σ) of the NCC accuracies corresponding to the outcomes of the 25 training iterations conducted on 89 UCR archive datasets. If we compare the results reported on Tables 3.16 and 3.21, we can observe that the standard deviation for the variational autoencoders is relatively large. We found this to be logical since we introduced an additional source of randomness in the latent space of the autoencoder, i.e., we take random samples from normal

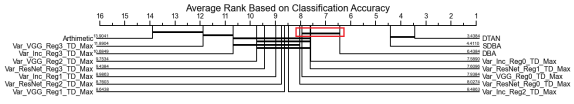
Table 3.21: Standard deviation of NCC accuracies that are obtained using: modified reduced VGG16, reduced Inception, and reduced ResNet autoencoders.

Techniques	$\pm\sigma$ in % L2 Reg0	$\pm\sigma$ in % L2 Reg1	$\pm\sigma$ in % L2 Reg2	$\pm\sigma$ in % L2 Reg3
Var_VGG_Lat	3.54	7.21	5.61	10.32
Var_Inc_Lat	4.39	4.61	3.49	4.29
Var_ResNet_Lat	9.78	7.28	7.49	7.31
Var_VGG_TD	5.75	9.20	8.19	9.26
Var_Inc_TD	5.24	6.26	6.45	7.89
Var_ResNet_TD	12.13	9.90	9.89	9.25

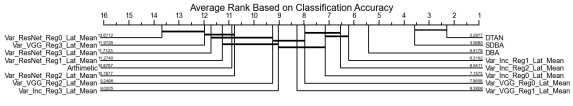
Gaussian distribution to define the latent embedding. However, even with this additional source of randomness, the latent space standard deviation is below 5% for most of the L2 regularization and architectural configurations. Moreover, in the time domain, it is below 8%. However, for the variational reduced ResNet architecture, the overall standard deviation is grater than its counterparts. We speculate that this slightly higher standard deviation is associated with the *Addition* layer within the ResNet architecture. In the architecture, we add the outputs of *Convolutional* layers with the outputs of predecessor layers. In the context of the variational autoencoders, the addition operation continuously introduces a constant offset on the means of the latent features. However, since the overall architecture is trying to pull down the mean to zero, it is forced to look for different alternatives on different initialization. This in turn contributes to the slightly larger variance across estimates.



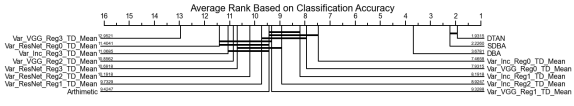
(a) Evaluation of latent space maximum NCC accuracies as compared to alternatives



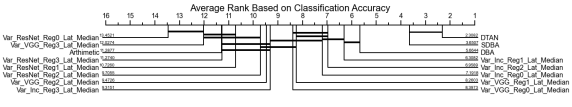
(b) Evaluation of latent space and time domain maximum NCC accuracies as compared to alternatives



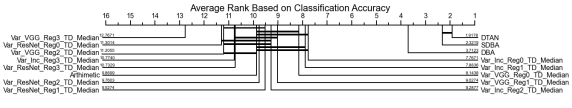
(c) Evaluation of latent mean NCC accuracies as compared to alternatives



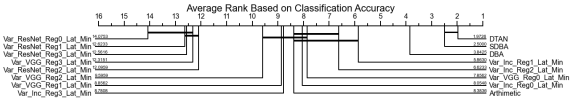
(d) Evaluation of latent space and time domain mean NCC accuracies as compared to alternative



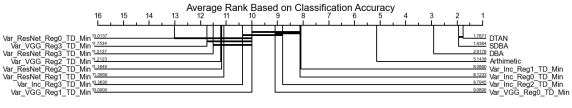
(e) Evaluation of latent space median NCC accuracies as compared to alternative



(f) Evaluation of latent space and time domain median NCC accuracies as compared to alternative



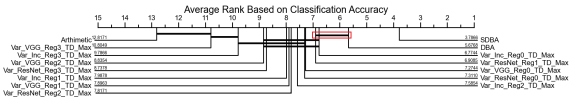
(g) Evaluation of latent space minimum NCC accuracies as compared to alternative



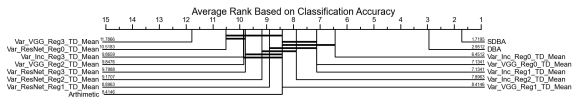
(h) Evaluation of latent space and time domain minimum NCC accuracies as compared to alternative

Figure 3.29: Comparison of NCC accuracies that are obtained with the estimates of variational autoencoders and their counter parts. These comparison are based on the NCC accuracies that are obtained from 73 UCR archive datasets.

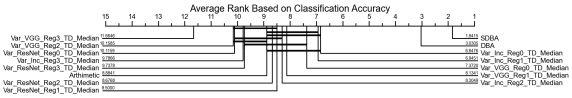
In general, like their basic counterparts, architectures based on the VGG16 and Inception appears to be relatively stable. With this in mind, we next compare the NCC accuracies across averaging techniques to assess which of the proposed variational architectures performs better. In this aspect, Figure 3.30 show the comparison of latent space and time domain NCC accuracies across averaging techniques. Overall, we found the basic autoencoders to generate better separable and dense latent space representation. However, due to the relatively lower latent space constraint, their embedding could span over a wider area of the latent space. To this end, their decoder is expected to generalize to a wider area of the latent space. This in turn has a negative implication on the re-projection of the latent means. In this aspect, we found the variational autoencoders to be better. We have summarized this observation in Figure 3.31 (b). In the figure, we have compared the median latent space and time domain NCC accuracies associated with the variational and non-variational autoencoders. To make the comparison, we took the outcomes corresponding to 83 UCR archive datasets and zero $L2$



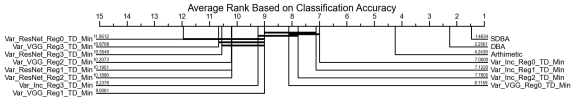
(a) Evaluation of time domain maximum NCC accuracies as compared to alternatives



(b) Evaluation of time domain mean NCC accuracies as compared to alternatives



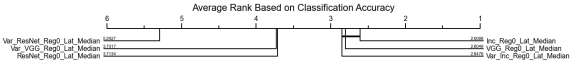
(c) Evaluation of time domain median NCC accuracies as compared to alternatives



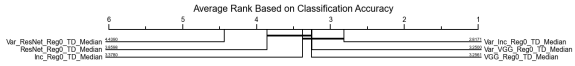
(d) Evaluation of time domain minimum NCC accuracies as compared to alternatives

Figure 3.30: Comparison of NCC accuracies that are obtained with the estimates of variational autoencoders and their counter parts. These comparison are based on the NCC accuracies that are obtained from 83 UCR archive datasets and the time domain estimates of arithmetic mean, DBA, SDBA, and variational auotencoders.

regularization. According to Figure 3.31, in the latent space, the variational and the non-variational Inception autoencoders behaved similarly. However, the variational version performed lower than the non-variational VGG16 based autoencoder. We can also observe the same performance among the time domain accuracies of the Inception architectures. However, in this case, the variational Inception performed better than its non-variational VGG16. In general, if the non-variational autoencoders get used, we suggest the architecture gets based on VGG16. On the contrary, if the variational versions gets used, we suggest the utilization of the Inception architecture.



(a) Comparison of latent space median NCC accuracies



(b) Comparison of time domain median NCC accuracies

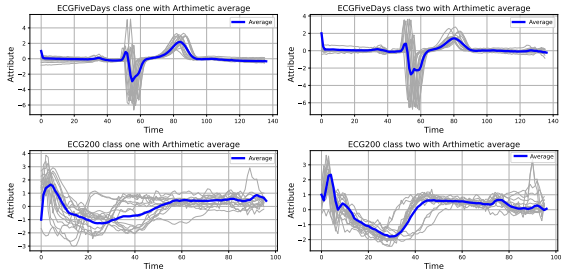
Figure 3.31: Comparison of median latent space and time domain NCC accuracies that are obtained with the estimates of variational and non variational autoencoders. These comparisons are based on the outcomes of NCC accuracies conducted on 83 UCR archive datasets

We will conclude this chapter by presenting the estimates generated by the variational autoencoders for the UCR archive's *ECG200* and *ECGFiveDays* datasets, i.e., as shown in Figure 3.32. In Figure 3.32,

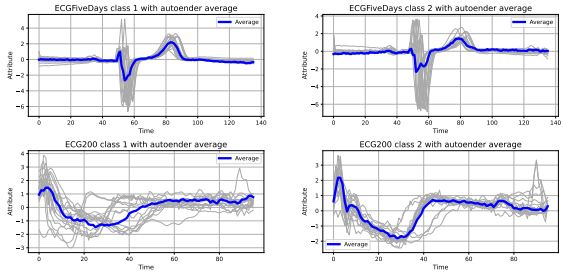
Table 3.22: NCC accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets.

Techniques	NCC for *ECG200* in %	NCC for *ECGFiveDays* in%
Arithmetic	67	52.96
DBA	65	52.15
SDBA	73	67.02
DTAN	**79**	**97.79**
Enc_Time	65	52.15
VGG_TD	**76**	**75.61**
Inc_TD	76	71.54
ResNet_TD	**76**	**72.24**
Var_VGG_TD	72	67.25
Var_Inc_TD	74	68.41
Var_ResNet_TD	76	70.84

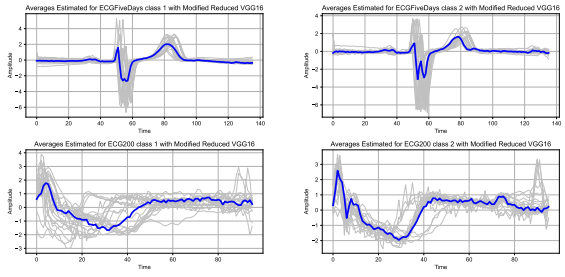
it is clear that the estimates minimized the shape distortion previously evident in the estimates of the



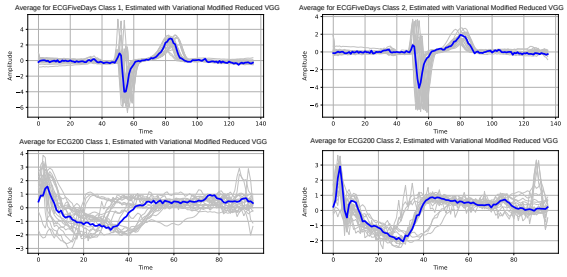
(a) Arithmetic means



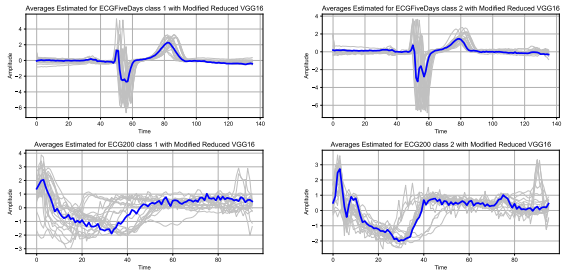
(b) Reduced VGG16 estimates



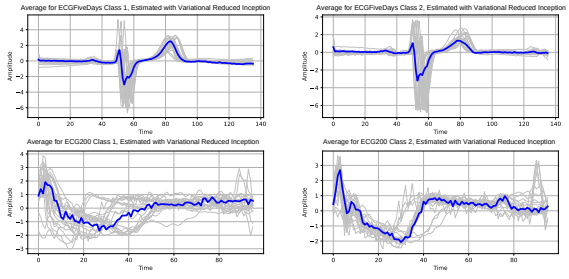
(c) Modified reduced VGG16 estimates



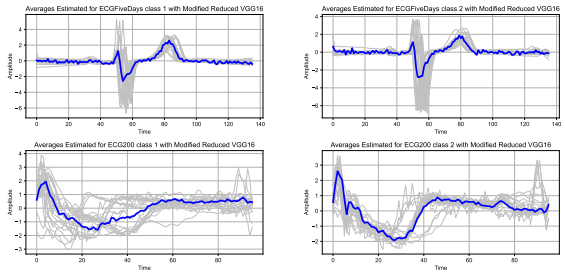
(d) Variational modified reduced VGG16 estimates



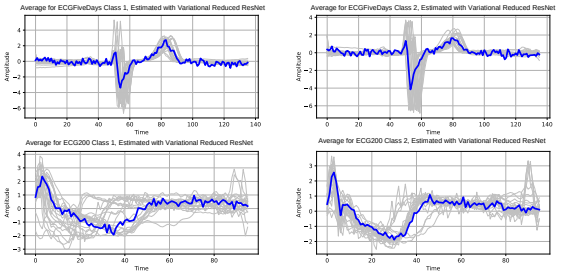
(e) Reduced Inception estimates



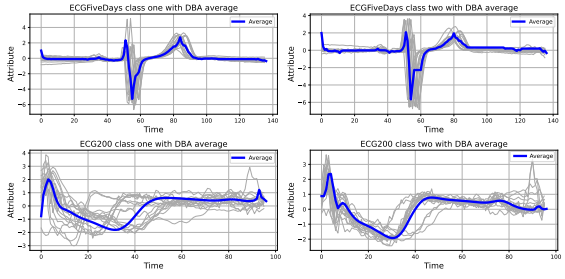
(f) Variational reduced Inception estimates



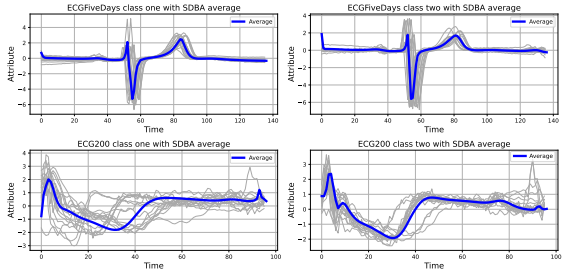
(g) Reduced ResNet estimates



(h) Variational reduced ResNet estimates



(i) DBA estimates



(j) SDBA estimates

Figure 3.32: Visual comparison of estimated averages for the UCR archive's *ECG200* and *ECGFiveDays* datasets.

ECGFiveDays dataset. Moreover, the variational ResNet autoencoder's estimates resembled the estimates of DBA. However, the autoencoder's estimates contained ripples in response to minor spikes in the dataset. In summary, we have given the NCC accuracies associated with the different estimates in Table 3.22. Even though the variational autoencoder's estimates visually appear to be better than their basic counterparts, the NCC accuracies state the contrary. One contributing factor could be the higher ripply nature of the variational autoencoder estimates. In reality, since the variational autoencoder's latent space is relatively confined to a smaller region, we expect the decoder to be sensitive to slight changes in the latent space. This, in turn, makes it sensitive to spikes that are outliers at a time stamp level. This in turn could have a negative implication on the DTW warping process utilized in NCC. With this said, we conclude this chapter and proceed to address the limitations observed with the autoencoder-based average estimation process.

4 Time Series Averages from the Latent Space of Multi-Tasking Neural Networks

The main limitation of the variational and non-variational autoencoders is that they only guarantee the extraction of reconstructable latent embedding. However, given the multi-class nature of the UCR datasets, we desire the extracted latent features to be separable and compact. In reality, previous proposals guarantee dense transformation either through DTW or CPA fields. Moreover, if there are multiple classes (clusters), they directly or indirectly utilize class label information in the estimation process [16], [25]. In this regard, in [20], [25], DBA's and SDBA's averages were estimated on a per-class basis. Additionally, from (2.39), we can see that DTAN utilizes class labels while minimizing the WGSS loss between the morphed series and their respective centroids. Thus, given the availability of class (cluster) labels, a logical step to take would be to utilize the label information in the feature extraction and augmentation process. With this underlying assumption, we propose to estimate the time domain per-class averages from the latent space of a multi-tasking autoencoder. In this regard, we propose our multi-tasking network to perform multi-class classification and reconstruction. In other words, we propose the multi-tasking autoencoder to optimize the objective function given in (4.1), where $Cat_{i,j}$ is the label given for an input dataset out of the C categories and $p_{i,j}$ is the *Softmax* activation value assigned to it by a classifier. Moreover, X_i, $\hat{X}_i \; \epsilon \; \mathbb{R}^M$ are an input time series and its reconstruction. With this said, we preset how we modified the autoencoder shown in Figure 3.9 into its multi-tasking version and the corresponding experimental evaluation.

$$L_{Multi}(X_i, \hat{X}_i, Cat, p_{cat}) = \frac{1}{N}\sum_{i=1}^{N}||X_i - \hat{X}_i||_{l2} - \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} Cat_{i,j} \ln^{p_{i,j}} \tag{4.1}$$

4.1 Time Series Averaging Using a Multi-tasking Autoencoder

Architecture-wise, the proposed multi-tasking autoencoder has a classifier attached to the encoder portion of the autoencoder. We construct this classifier from three fully connected *Dense* layers. Moreover, for the first two *Dense* layers, we use ReLU activation function. However, we set the last classifier's *Dense* layer activation function to *Softmax*. Hence, the outputs of the classifier can be interpreted as the probability of occurrence of the categories. To accommodate this concept, we set the number of neurons at the last *Dense* layer to be equal to the number of categories. On the contrary, the first two classifier's *Dense* layers are configured with $\frac{M}{8}$ and $\frac{M}{16}$ neurons. Thus, the change in the number of trainable parameters is insignificant compared to the parameters given in 3.7. In this regard, the additional three classifier's *Dense* layers incur: $(\lfloor\frac{M}{4}\rfloor \times \lfloor\frac{M}{8}\rfloor) + \lfloor\frac{M}{8}\rfloor$, $(\lfloor\frac{M}{8}\rfloor \times \lfloor\frac{M}{16}\rfloor) + \lfloor\frac{M}{16}\rfloor$, and $\lfloor\frac{M}{16}\rfloor \times Cat + Cat$ additional trainable parameters as compared to the parameters given in Table 3.7. However, despite these changes, we reused the configuration for the encoder and decoder shown in Figure 3.9 and Table 3.7. Overall, the architectural configuration of the multi-tasking autoencoder is shown in Figure 4.1. With this said, we will proceed with the experimental setups.

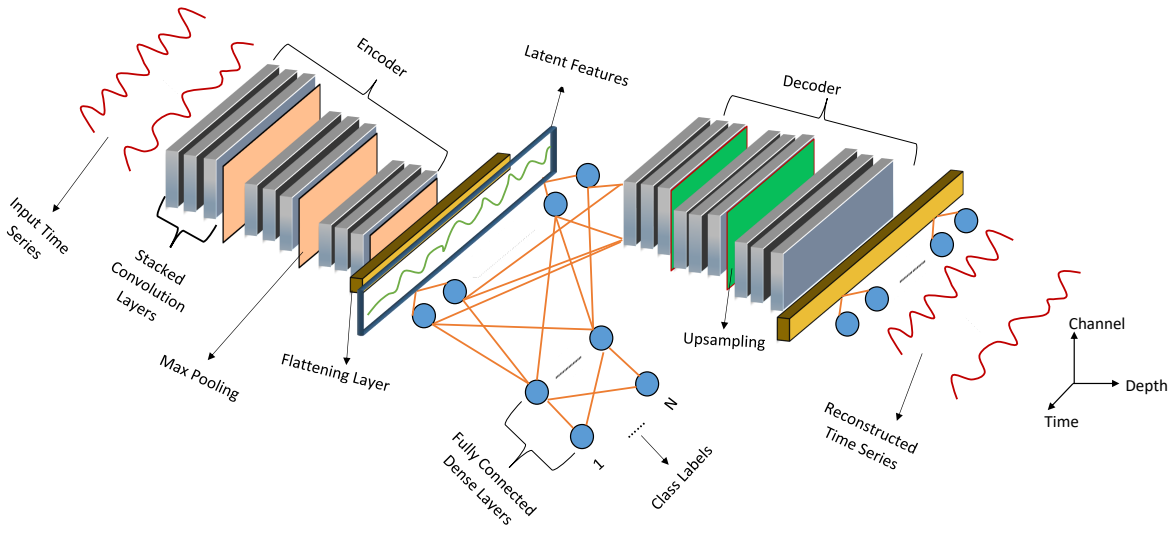


Figure 4.1: Proposed reduced VGG16 multi-tasking autoencoder

4.1.1 Experimental Setup

We have also trained the multi-tasking autoencoder using 80% of the training set for training and 20% for validation. However, unlike the setups utilized for the basic autoencoder, we have used two sets of $L2$ regularization. In this aspect, for the encoder and decoders we utilized $L2_{encoder/decoder} = [0,\ 0.0001,\ 0.001,\ 0.001, 0.01,]$. On the contrary, for the classifier we used $L2_{classifier} = [0,\ 0.001,\ 0.001,\ 0.01,\ 0.01]$. To this end, at a given training, the encoder and decoder used similar $L2$ regularization factors. In the country, in most cases, the classifier is regularized by a factor that is 10 times higher. For instance, $\{L2_{Encoder},\ L2_{Deccoder},\ L2_{Classifier}\} = (0.0001, 0.0001, 0.001)$. We chose this regularization approach since the classifier is more susceptible to overfitting due to its fully connected *Dense* layers. Moreover, since we proposed the classifier to aid the overall arrangement of the extraction of separable and dense latent features, we desire the classifier to generalize well. To this end, we place a relatively higher $L2$ regularization penalty on the weights of the classifier. Despite this major difference, we have kept the remaining training configuration relatively similar to the one used on the basic autoencoder. In this aspect, we set the learning rate and the number of training epochs in the case of zero $L2$ regularization to 10^{-4} and $epochs = 600$. However, for non-zero $L2$ regularization, we used 2500 training epochs to ensure the convergence of the network. Finally, we have utilized the trained network and the train split to estimate the latent and time domain average. In this regard, the per-class latent averages get estimated by taking the arithmetic mean of the latent space representation of the training set. Moreover, we used the decoder to project the latent space estimates into the time domain. In addition to this, we also used the trained network to project the test datasets into the latent space. We then performed a latent space and time domain NCC using 84 UCR datasets.

4.1.2 Experimental Results and Interpretation

To evaluate the performance of the multi-tasking autoencoder, we re-utilized the assessment techniques used for the basic autoencoder. To this end, we will make the first assessment using the Win

Ties loss analysis shown in Table 4.1, where Enc_Lat (Time) and MT_ Enc_ Lat (Time) correspond to the latent space (time domain) NCC accuracies of the basic and multi-tasking autoencoders.

4.1.3 Experimental Results and Interpretation

In order to evaluate the performance of the multi-tasking autoencoder, we re-utilized the assessment techniques that we used for the basic autoencoder. To this end, we will make the first assessment using the Win Ties loss analysis shown in Table 4.1, where Enc_Lat (Time) and MT_ Enc_ Lat (Time) correspond to the latent space (time domain) NCC accuracies of the basic and multi-tasking autoencoders.

Table 4.1: Analysis of wins/ties/losses of the NCC accuracies that are obtained using the estimates of the multi-tasking autoencoder and alternative averaging techniques.

Technique	Wins	Ties	Losses
Arithmetic	1	0	83
DBA	1	1	82
DTAN	**30**	**4**	**50**
Enc_Lat	2	0	82
Enc_Time	0	0	83
MT_Enc_Lat	**27**	**1**	**56**
MT_Enc_Time	1	1	82
SDBA	**16**	**2**	**66**

In Table 4.1, we have marked the top three performing average estimation techniques using bold-faced letters. According to these results, the multi-tasking autoencoder is performing better than DBA and SDBA in the latent space. This fact gets further validated by the box-whisker plot shown in Figure 4.2. According to the statics of the plot, i.e., Table 4.2, the multi-tasking autoencoder achieved a latent space median NCC accuracy of 75.03%. This is a 16.37% improvement compared to its basic autoencoder counterpart (Enc_Lat). Furthermore, the worst and best latent space NCC accuracies now

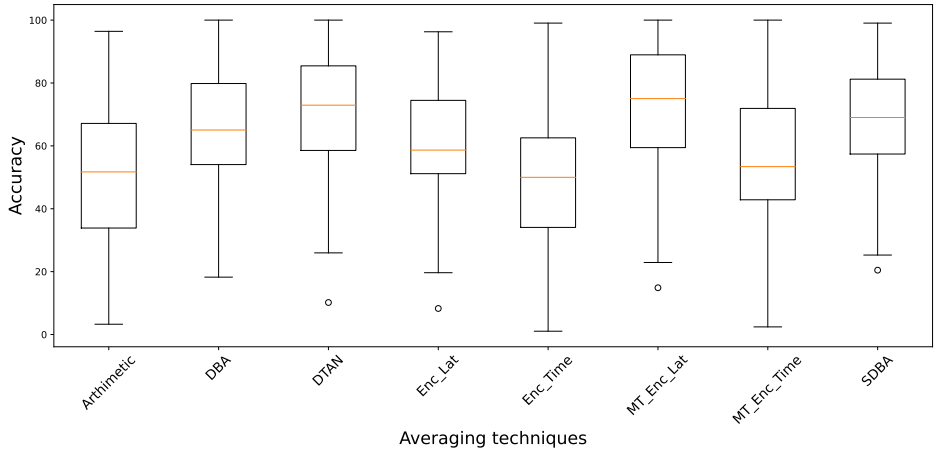


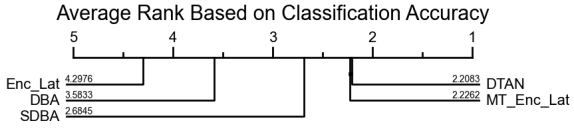
Figure 4.2: Box-whisker plot comparisons of the NCC accuracies that are obtained using averages estimated with the multi-tasking auroencoder and its counterparts.

Table 4.2: Statistical parameters for the box-whisker plot shown in Figure 4.2

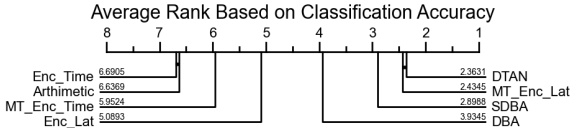
Technique	Top Whisker	Bottom Whisker	25% percentile	75% percentile	Median
Arithmetic	96.43	3.27	33.87	67.14	51.72
DBA	100	18.25	54.05	79.84	65.04
DTAN	**100**	**25.97**	**58.55**	**85.45**	**72.94**
Enc_Lat	96.29	19.66	51.16	74.48	58.66
Enc_Time	99.05	1.05	34.07	62.55	50.00
MT_Enc_Lat	**100**	**22.91**	**59.44**	**88.95**	**75.03**
MT_Enc_Time	100	2.43	42.85	71.92	53.40
SDBA	**99.05**	**25.27**	**57.41**	**81.22**	**69.02**

became 100% and 22.91%. This is also better than its basic autoencoder part standing at 99.29% and 19.66%. Additionally, 50% of the multi-tasking latent space classification accuracies are between 59.44% and 88.95%. This is also significantly better than the 51.16%-74.40% range of the basic autoencoder. In practice, we expect the multi-tasking autoencoder to perform better in the latent space. In reality, for the multi-tasking autoencoder, we have a classifier that forces the encoder to extract reconstructable and class-specific latent features. Thus, the latent space representation of the multi-tasking setup gets expected to be relatively dense and separable. The question now becomes, how do the separability and compactness of the latent space features manifest themselves in the time domain? In this aspect, in the time domain (MT_Enc_Time), the multi-tasking autoencoder showed a 3.40% increase in its median accuracy, i.e., from 50% accuracy in the basic autoencoder (Enc_Time) to 53.40%. Moreover, the worst and best case scenarios now improved from 1.05% and 99.05% to 2.43% and 100%. In addition to this, 50% of its classification accuracies are now in between 42.85% to 71.92%, i.e., as compared to the 34.07% to 62.55% of the basic autoencoder. We find these short 1ncc statistics as a significant improvements compared to their basic autoencoder counterpart. This conclusive remark is also validated using the hypothesis tests shown in Figure 4.3.

In the hypothesis evaluation, we have followed the same approach we used for the basic autoencoders and we separated the hypothesis test into two categories. First, we compare the NCC accuracies that are obtained in the registered space of the averaging techniques. We then include the time domain classification results obtained using an arithmetic mean, multi-tasking and basic autoencoders. In reality, these NCC results are the only ones evaluated in a neutral space. In this aspect, the arithmetic and time domain re-projection of the autoencoders gets warped into a DTW space to which they have no



(a) Evaluation based on NCC accuracies obtained in the registered space of the averaging techniques

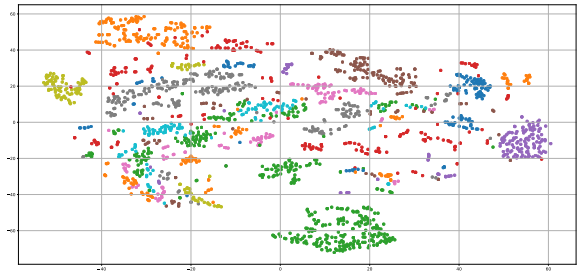


(b) Evaluation including time domain and latent space NCC accuracies

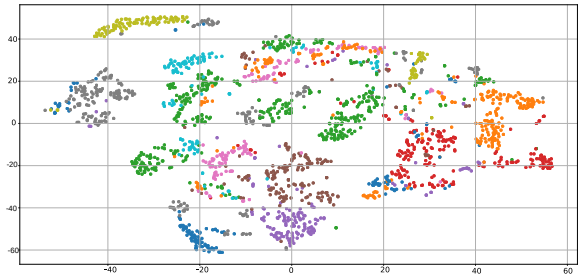
Figure 4.3: Hypothesis test based on the NCC accuracies obtained with the estimates of the multi-tasking autoencoder and its counterparts.

prior knowledge. According to Figures 4.3 (a) & (b), the multi-tasking autoencoder outperforms DBA and SDBA using its latent space estimates. Moreover, in the latent space, it is statistically indifferent to the state of the art (DTAN). Additionally, in the time domain, the multi-tasking autoencoder estimates perform far better than the arithmetic mean. In this aspect, statistically, 50% of the arithmetic mean classification results are between 33.86% to 67.14%. This is relatively lower than the multi-tasking autoencoder range of 42.85% to 71.92%.

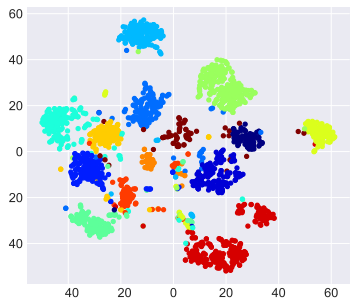
In addition to the numerical improvements, a t-SNE projection of the multi-tasking's latent space features reveals their capability of mimicking the effects of multiple alignments observed in predecessor techniques such as DTAN. In this regard, in Figure 4.4, we have revisited the *FacesUCR* test dataset and plotted their t-SNE projections when seen from: the time domain, latent spaces (multi-tasking and basic autoencoders) and DTAN's morphed space. According to Figures 4.4 (b) and (d), the multi-tasking autoencoder's latent space representation of the test dataset is separable and dense as compared to the basic autoencoder. Additionally, the latent space of the multi-tasking autoencoder is mimicking the effects of multiple alignments which is evident in DTAN's morphed space as shown in Figure 4.4 (c).



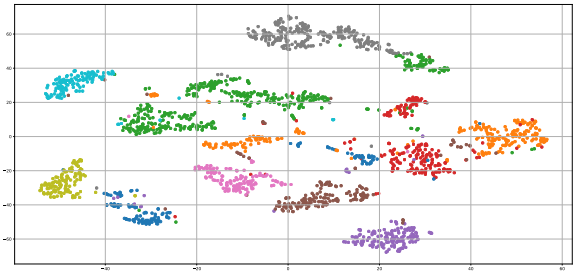
(a) *FacesUCR* in the time domain



(b) *FacesUCR* in the latent space of the basic autoencoder



(c) *FacesUCR* in the morphed space of DTAN [25]

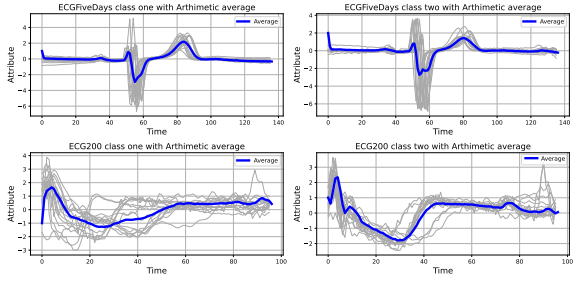


(d) *FacesUCR* in the latent space of the multi-tasking autoencoder

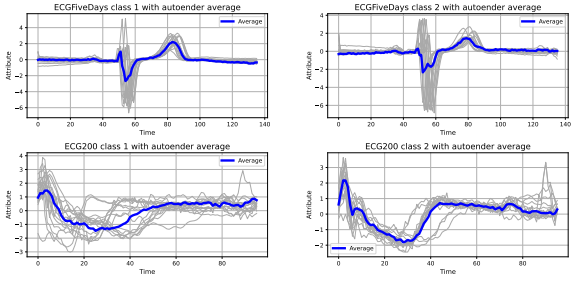
Figure 4.4: t-SNE projections for the UCR archive's *FacesUCR* test datasets: in time domain (a), autoencoder's latent space (b), DTAN's morphed space (c) and multi-tasking autoencoder's latent space (c)

In addition to the improvements observed in the compactness of the latent features, the improvement is also evident in the shapes of the time domain re-projected averages. In order to demonstrate this fact, we revisit the UCR's *ECG200* and *ECGFiveDays* datasets and we show the time domain estimate of the different averaging techniques in Figure 4.5. In this regard, if we carefully compare the estimates shown in Figure 4.5 (b) and (e), we can observe that the multi-tasking autoencoder estimates have better captured the negative peaks of the *ECGFiveDays* datasets. Moreover, compared

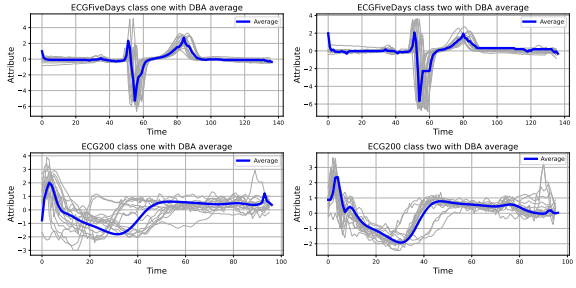
to the estimates of the basic autoencoder, the multi-tasking autoencoder generated a better estimate for the first class of the *ECG200*. However, despite such improvements, the time-domain estimates of the multi-tasking autoencoder are still performing well below the performances of the alternative averaging techniques. This fact is reflected on the NCC accuracies of the multi-tasking autoencoder shown in Table 4.3. In this regard, we have also identified three contributing factors that can be improved. First, the filter arrangements of the encoder and the *UpSampling* layers at the decoder are similar to the basic autoencoder. Thus,we propose to first take corrective measures in this regard. Moreover, we have also not assessed different architectural setups of the multi-tasking autoencoder.



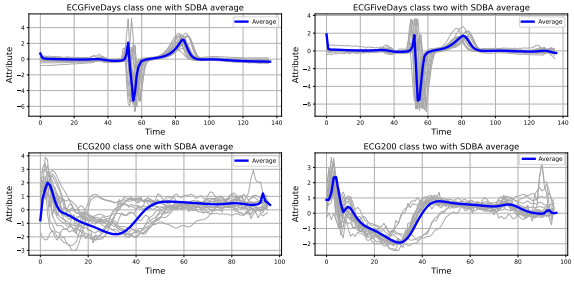
(a) Arithmetic means



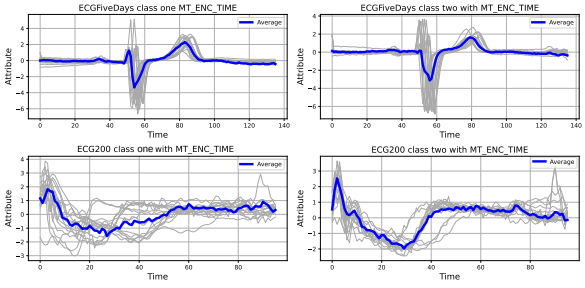
(b) Autoencoder's estimates



(c) DBA's estimates



(d) SDBA's estimates



(e) Multi-tasking autoencoder's estimates

Figure 4.5: Averages that are estimated for the UCR archive's *ECG200* and *ECGFiveDays* datasets

Finally, we strongly believe that we can not solely rely on minor modifications to obtain a significant improvement. To this end, in addition to these modifications, we propose to further analyze additional parameters, such as the objective functions of the network. In general, we observe two key points that could be improved. First, if we further zoom in and look at the estimations for the *ECGFiveDays*'s second class, the positive going peaks of the estimates got significantly distorted (clipped). One reason behind this distortion could be that we fully relied on the classifier to identify separable and compact latent features. However, in reality, the classifier can not fully guarantee dense latent features. For instance, a classifier could identify similar per class features that occupy a specific region of the latent

Table 4.3: NCC classification accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets

Averaging Technique	accuracy on *ECG200* in %	accuracy on *ECGFiveDays* in %
Arithmetic	67	52.96
DBA	65	52.15
DTAN	**79**	**97.79**
Enc_Time	65	52.15
MT_Enc_Time	72	58.65
SDBA	73	67.02

space. However, the features could have different magnitude levels. Under such situations, latent space representation of the multiple classes could become separable but not dense. Additionally, the non-variational multi-tasking autoencoder setup still has a discontinuous latent space. To this end, we can safely assume that the decoder highly relies on the latent space mappings of the training datasets to estimate the projections of neighborhood points, such as a latent mean. Thus, if the latent space features are not dense enough, the decoder will still have difficulties identifying exemplary neighborhood latent space representations for the projection of the latent means. Thus, if we desire to mitigate this challenge, the objective function should have a part that explicitly accounts for this factor. Finally, we should also ask ourselves, does having a rigid reconstruction criterion favors a better latent mean re-projection? In reality, we are using the decoder as a generative unit while re-projecting latent estimates. On the contrary, when we utilize a reconstruction loss (mean squared error), we wish that the decoder learns a perfect fit for the reconstruction of the training datasets. However, we later expect it to project a latent mean that it has never seen before (has no prior ground truth). To this end, we believe the decoder's objective function must further get relaxed if better time domain estimations get desired. With these observations in mind, in the next section, we will first re-evaluate the change in the quality of the time domain estimates by only changing the architectural setup. On the contrary, in the following section, we will address the limitations observed in the objective functions of the multi-tasking autoencoder.

4.2 Extended Evaluation of Multi-tasking autoencoders

In this section, we propose to follow the same approach as the extended re-evaluations of the basic autoencoder. To this end, we modify the autoencoder portion of the multi-tasking autencoder using layer arrangements that resemble the modified: reduced VGG16, ResNet and Inception version two. We propose to evaluate the multi-tasking version of these architectures using their variational and non-variational form. With this said, we will start our further discussion on the layer arrangements of the multi-tasking autoencoder that are based on the non-variational form of the mentioned autoencoders.

4.2.1 Modified Reduced VGG16 Based Multi-tasking Autoencoder

In this setup, we have re-used the autoencoder architecture shown in Figure 3.15. To this end, the *UpSampling* layers in the previous basic multi-tasking autoencoder gets substituted with transposed

Convolutional layers. Moreover, we have also set the *stride* of the last transposed *Convolutional* layer to one. On the contrary, we set the *stride* of the remaining two transposed *Convolutional* layers to two. At the encoder, we have also set the stride of the last *MaxPooling* layer to one, i.e., to accommodate datasets with a smaller dimension. Finally, we have kept the classifier layer arrangement similar to the basic multi-tasking autoencoder, i.e, three *Dense* layers. However, unlike the basic multi-tasking autoencoder, we have set the neuron size of the first two classifier's *Dense* layers to $\lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor$ and $\lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor$, where M is the length of the averaged series. To this end, parameter wise, the modified reduced VGG16 multi-tasking autoencoder has an additional $\lfloor \frac{M}{4} \rfloor \times \lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor + \lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor$, $\lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor \times \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor + \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor$ and $C \times \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor + C$ trainable parameters as compared to the once shown in Table 3.11. However, when we deploy the variational version of this multi-tasking setup, there is an additional $(\lfloor \frac{M}{4} \rfloor \times 32) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$ trainable parameter accounting for the additional *Dense* layer at the encoder.

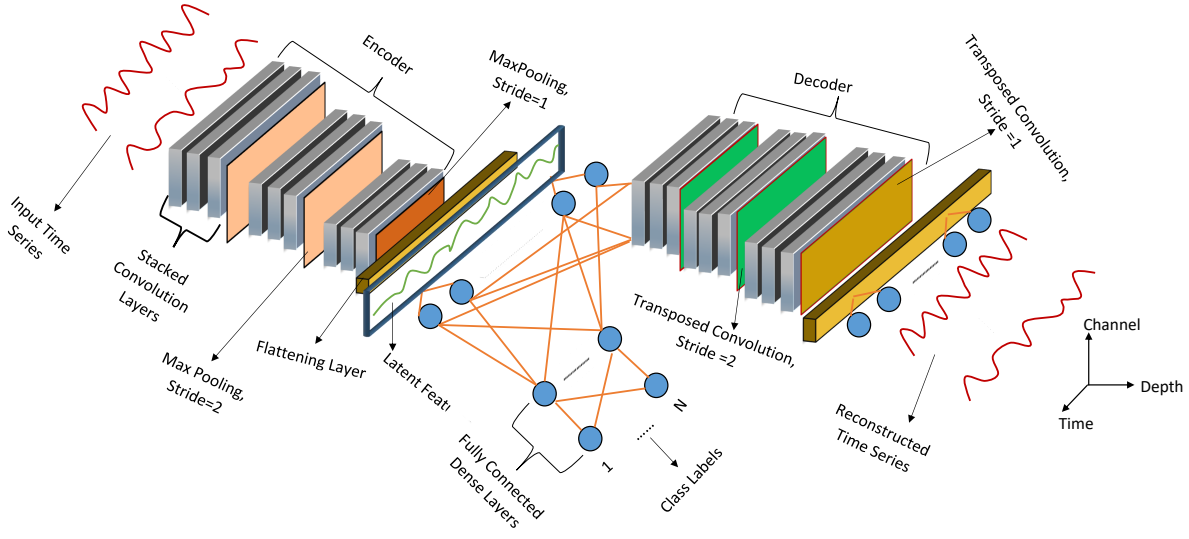


Figure 4.6: Proposed modified reduced VGG16 multi-tasking autoencoder

4.2.2 Proposed Reduced ResNet Multi-tasking Autoencoder

In this setup, we replace the modified reduced VGG16 autoencoder with the Inception-based autoencoder shown in Figure 3.17. However, we have kept the classifier architecture similar to the previous multi-tasking proposals, i.e., three *Dense* layers. Moreover, we have also set the number of neurons at the classifier's *Dense* layers to $\lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor$, $\lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor$ and C, where M, C are the length of the averaged series and the number of classes (categories). To this end, as compared to the parameters shown in Table 2.5, this multi-tasking setup has an additional $\lfloor \frac{M}{4} \rfloor \times \lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor + \lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor$, $\lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor \times \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor + \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor$ and $C \times \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor + C$ trainable weights. Moreover, when the variational version of this multi-tasking autoencoder is implemented, there will be an additional $(\lfloor \frac{M}{4} \rfloor \times 32) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$ trainable parameters in order to account for the additional *Dense* layer at the encoder. In general, we have shown the overall architecture of the reduced ResNet based multi-tasking autoencoder in Figure 4.7.

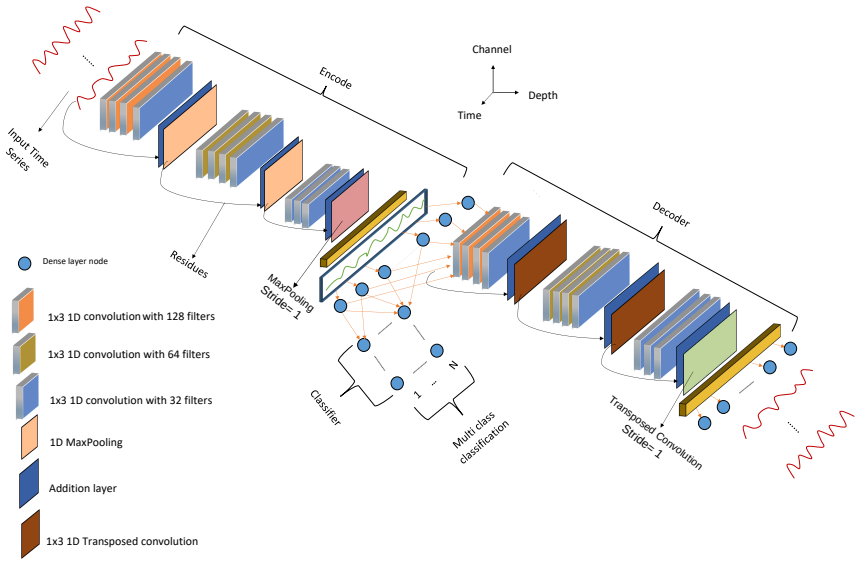


Figure 4.7: Proposed reduced ResNet multi-tasking autoencoder

4.2.3 Inception Version Two Based Multi-tasking Autoencoder

Following the same trend, for this setup, we have also changed the encoder-decoder portion of the multi-tasking configuration with the Inception-based autoencoder shown in Figure 3.17. To this end, the major parameter difference between the Inception multi-tasking setup and the one shown in Table 3.13 would also be $\lfloor \frac{M}{4} \rfloor \times \lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor + \lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor$, $\lfloor \frac{M}{4} - \frac{0.1 \times M}{4} \rfloor \times \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor + \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor$ and $C \times \lfloor \frac{M}{4} - \frac{0.2 \times M}{4} \rfloor + C$. Similarly, when the variational version of this multi-tasking autoencoder is implemented, there will also be an additional $(\lfloor \frac{M}{4} \rfloor \times 32) \times \lfloor \frac{M}{4} \rfloor + \lfloor \frac{M}{4} \rfloor$ trainable parameters in order to account for the additional *Dense* layer at the encoder In conclusion, the layer arrangement for this architecture is shown in Figure 4.8.

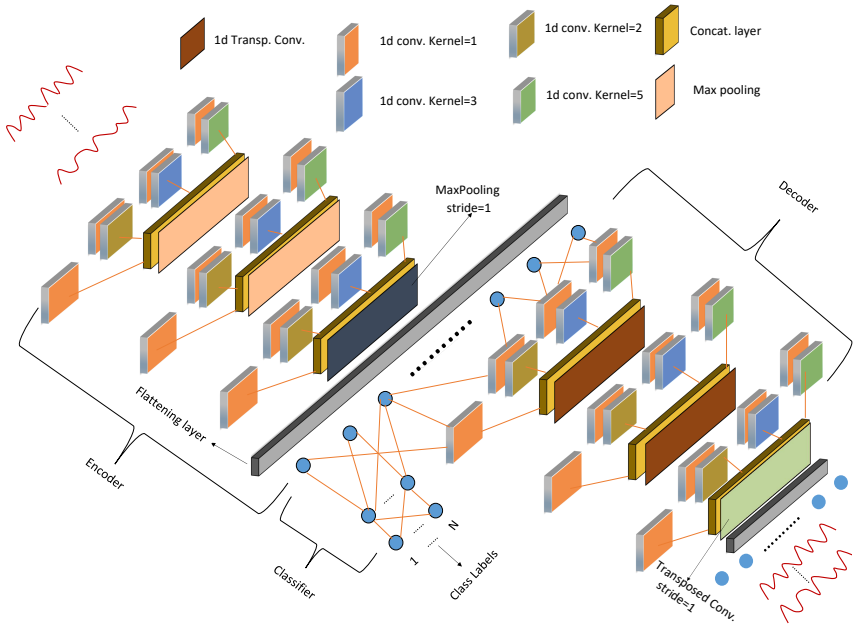


Figure 4.8: Proposed reduced Inception version two multi-tasking autoencoder

4.2.4 Experimental Setup

In our extended evaluation of the basic multi-tasking autoencoder, we have also used 80% of the training dataset for training and 20% for validation. However, in this evaluation, we have trained the variational and non-variational proposals using only four different $L2$ regularization setups, i.e., $L2 = [0,\ 0.0001,\ 0.001,\ 0.01]$. Moreover, unlike the training setups used for the reduced VGG16 multi-tasking autoencoder, we propose to regularize the encoder, decoder and classifier with the same set of $L2$ regularization values say for instance, $[L2_{encoder}, L2_{decoder}, L2_{classifier}] = [0.001,\ 0.001,\ 0.001]$. In addition to this training setup modification, we propose to change the number of training epochs for non-zero $L2$ regularization from 2500 to 1500. Additionally, in this case, we aim to train the multi-tasking networks for 25 repeated trials on each regularization setup. We update the gradients of each training trial after passing through $\lfloor \frac{K}{4} \rfloor$ mini-batches, where K is the number of training datasets. Finally, we access the performances of the proposals using minimum, maximum, median, and mean NCC accuracies. With this said, we will next proceed to present our experimental evaluations.

4.2.5 Experimental Results and Interpretation

4.2.5.1 Evaluation of Averages Estimated with Basic Multi-tasking Autoencoders

We start the extended evaluation of the non-variational multi tasking autoencoders with a wins ties losses analysis. In this aspect, Table 4.8 shows the wins/ties/losses associated with the NCC accuracies that are obtained using the estimates of the multi-tasking autoencoders and their counterparts. In the table, MMT_ VGG_ Regx_ Lat (TD)_ Max, MT_ Inc_Regx_ Lat(TD)_ Max, and MT_ ResNet_ Regx_ Lat(TD)_Max corresponds to the latent space (time domain) outcomes of the multi-tasking autoencoders based on: the modified reduced VGG16, Inception, and ResNet architectures. Moreover,

Table 4.4: Win/tie/losses analysis of NCC classification accuracies obtained from the extended evaluation of basic multi-tasking autoencoders.The analysis was performed using 74 UCR archive datasets, averages estimated via autoencoder and different averaging techniques, and NCC accuracies. Moreover, the outcomes reported for the autoencoders are generated using maximum NCC accuracies obtained from 25 repeated trials and four L2 regularization setups.

Techniques	$L2$ Reg. (x)	Wins	Ties	Losses
Arithmetic	-	0	0	74
DBA		1	1	72
DTAN		10	4	60
SDBA		10	1	63
MMT_VGG_Regx_Lat	{0, 1, 2, 3}	{5, 9, 1, 2}	{5, 5, 5, 2}	{64, 60, 68, 70}
MMT_VGG_Regx_TD		{0, 0, 0, 1}	{2, 0, 1, 1}	{72, 74, 73, 72}
MT_Inc_Regx_Lat		{2, 4, 1, 4}	{2, 4, 5, 5}	{70, 66, 68, 65}
MT_Inc_Regx_TD		{1, 0, 0, 0}	{1, 3, 0, 1}	{72, 71, 74, 73}
MT_ResNet_Regx_Lat		{2, 4, 1, 4}	{2, 4, 5, 5}	{70, 66, 68, 65}
MT_ResNet_Regx_TD		{0, 0, 0, 0}	{2, 1, 2, 2}	{72, 73, 72, 72}

the $x = 0,\ 1\ 2,\ 3$ in Regx stands for the type of regularization used to train the network, where $\{0,\ 1,\ 2,\ 3\}$ corresponds to $L2 = \{0, 0.0001, 0.001, 0.01\}$. Based on the results reported in Table 4.8, the latent space of the multi-tasking autoencoders is performing better than the ones obtained with the

plain autoencoder. In reality, we expect this to be evident since we are now targeting reconstructable per class latent features that are better separable. However, an interesting point here is that we now observe ties with the time domain estimates of the multi-tasking autoencoders. This is not evident with the estimates generated from the variational and non-variational autoencoders. However, as stated earlier, wins/ties/losses analysis does not provide concise information about the performances of the estimates. This is because an averaging technique can either be winning or losing with a small margin that has no significant practical implication. To this end, we next palace our focus on the box-whisker plot analysis of the NCC accuracies. In this aspect, Table 4.5 summarizes the statistics of the box-whisker plot shown in Figure 4.9.

Table 4.5: Statistics assessment of the NCC accuracies that are obtained with the multi-tasking modified VGG16, reduced Inception, and reduced ResNet architectures. These assessments were conducted using the maximum NCC accuracies associated with each averaging techniques.

Techniques	Bot. whisker	Top whisker	25% Quant.	75% Quant.	Median
Arithmetic	7.47	96.43	40.08	68.09	53.09
DBA	28.94	100	55.13	79.09	65.49
DTAN	33.31	100	59.31	85.62	74.30
SDBA	32.83	99.05	58.72	81.07	69.79
MMT_VGG_Regx_Lat x={0, 1, 2, 3}	{42.32, 42.63 41.07, 35.42 }	{100, 100 100, 100}	{50.87, 50.04 46.86, 38.94}	{90.63, 90.26 90.46, 89.52 }	{79.61, 79.54 77.23, 75.83}
MMT_VGG_Regx_TD x={0, 1, 2, 3}	{17.72, 19.43 16.77, 16.92}	{100, 99.05 100, 100 }	{51.83, 52.15 49.79, 46.82 }	{76.73, 76.63 75.40, 70.75}	{63.17, 61.64 60.82, 58.89}
MT_Inc_Regx_Lat x={0, 1, 2, 3}	{43.26, 45.04 42.48, 35.89}	{100, 100 100, 100}	{65.26, 62.54 63.59, 61.29}	{89.98, 89.85 89.04, 88.24}	{78.44, 78.48 78.13, 75.00}
MT_Inc_Regx_TD x={0, 1, 2, 3}	{20.53, 19.44 17.71, 13.79}	{99.05, 100 99.05, 96.19}	{52.44, 51.58 49.58, 46.33}	{76.99, 76.38 75.68, 71.26}	{63.38, 61.83 61.68, 58.22}
MT_ResNet_Regx_Lat x={0, 1, 2, 3}	{44.87, 40.91 42.79, 36.21}	{100, 100 100, 100}	{65.26, 62.54 63.59, 61.29}	{90.42, 90.81 90.45, 87.70}	{80.00, 79.56 79.52, 74.22}
MT_ResNet_Regx_TD x={0, 1, 2, 3}	{20.15, 17.08 20.06, 16.30}	{100, 100 100, 100}	{50.81, 51.99 50.00, 46.71}	{74.82, 76.83 75.70, 73.33}	{62.05, 63.74 60.43, 60.52}

In overall, Figure 4.9 shows that the performance of the time domain estimates of the multi-tasking autoencoders is far better than the arithmetic mean. This was not evident in the estimates of the basic and variational autoencoders. This observation supports our initial argument that dense and separable latent features have a positive implication on the time domain projection. In this regard, comparatively, the time domain median NCC accuracies of the multi-tasking autoencoders are better than their counterparts. For instance, the modified reduced VGG16 autoencoder obtained a best case time domain median accuracy of 54.70%. Moreover, its variational version obtained 52.45%. On the contrary, in the best case, the multi-tasking setup obtained a 63.17% median accuracy. In addition to this, the time domain lower whiskers of the multi-tasking setup are now relatively closer to DBA's lower whisker. In this aspect, the difference between the best case multi-tasking lower whisker and that of DBA is 8.41%. This gets achieved with the outcomes of the multi-tasking autoencoder that is based on the Inception architecture. In this regard, the variational and non-variational autoencoder

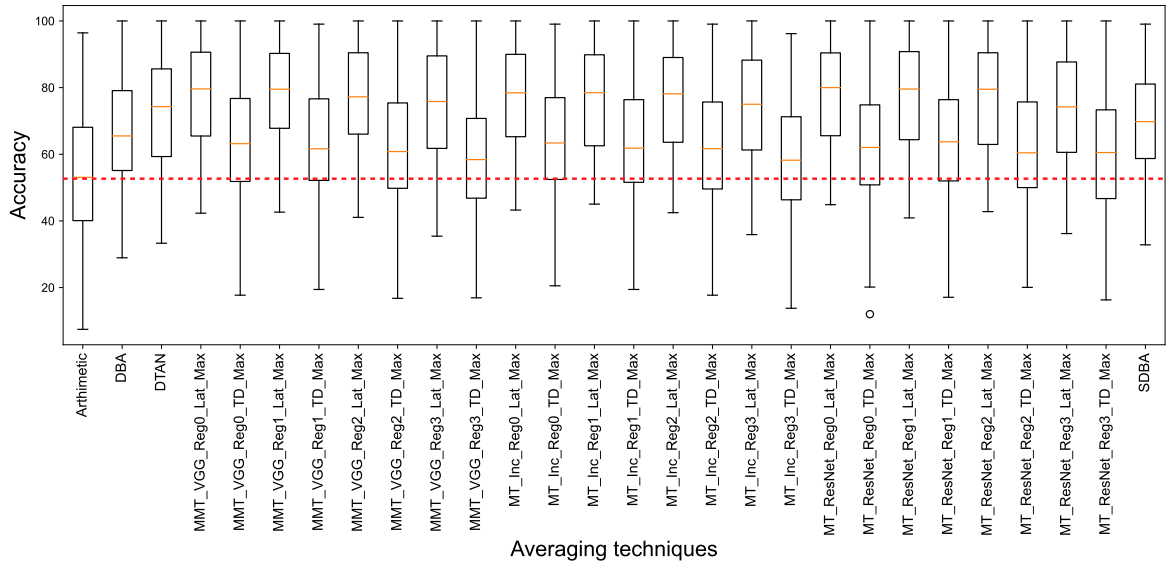
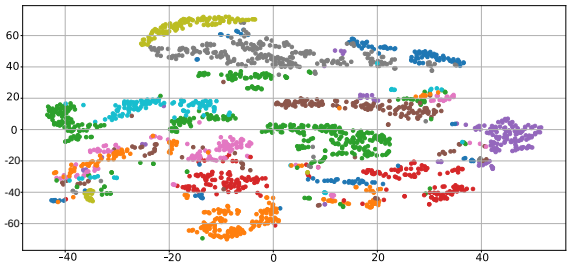


Figure 4.9: Box-whiker plot comparison of the NCC accuracies obtained with the multi-tasking autoencoders and their counterparts.

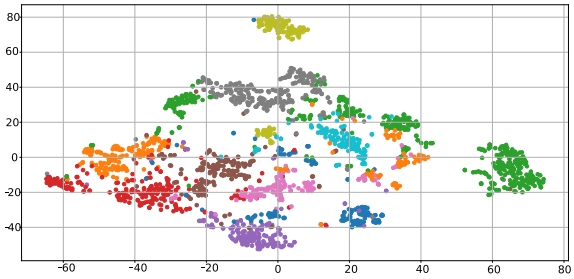
versions of the multi-tasking setups obtained a lower whisker difference of 9.39% and 10.95%. In general, we attribute this significant difference to the separability and compactness of the extracted latent embedding. To visually demonstrate this argument, we revisit the UCR archive's *FacesUCR* dataset and present their latent embedding in Figure 4.10. In the figure, we have included the latent embedding obtained by the autoencoders and their multi-tasking version for a better comparison. Among the embedding of the two approaches, we can see that the embedding of the multi-tasking setups (shown on the right column) is relatively dense. Moreover, if we see the embedding obtained with the multi-tasking Inception network, it is relatively separable compared to its counterpart. In reality, we expect this to have a positive implication on the re-projection since the multi-tasking setup reduces the chances of overlapping latent embedding that could lead to a relatively close latent means which the decoder often finds difficult to differentiate. In this aspect, the VGG16, Inception, and ResNet multi-tasking setups respectively obtained a 61.25%, 60.82%, 60.34% time domain NCC accuracies. On the contrary, their autoencoder counterparts respectively obtained a 48.54%, 49.56%, 44.39% NCC accuracy over the 14 classes of the *FacesUCR* dataset.

In reality, we obtained almost a 20% improvement in the performances of the time domain estimates by introducing class information. However, the next question now becomes, how well are such improvements distributed along the evaluation datasets? Even though the box-whisker plot gives a relatively better insight, i.e., as compared to wins/ties/losses analysis, it still does not tell us if there is a statistically significant difference among the averaging techniques. To this end, we next place our focus on hypothesis tests. In this regard, we first first compare the outcomes of the multi-tasking setups with their counterparts, i.e., arithmetic, DBA,SDBA, and DTAN using 74 UCR archive datasets. We then compare: arithmetic, DBA,SDBA, and the multi-tasking setups on 89 UCR archive datasets since the outcomes of DTAN was not reported for the 25 additional datasets. We then conclude, this subsection by presetting the estimates for *ECG200* and *ECGFiveDays* as a visual

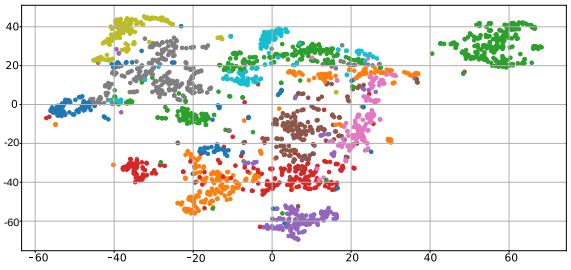
demonstration. However, before proceeding to these evaluations, first, we analyze which of the $L2$ regularization gives better performance. Moreover, we also assess the standard deviation (σ) of the NCC accuracies to check for the reproducibility of the experimental outcomes.



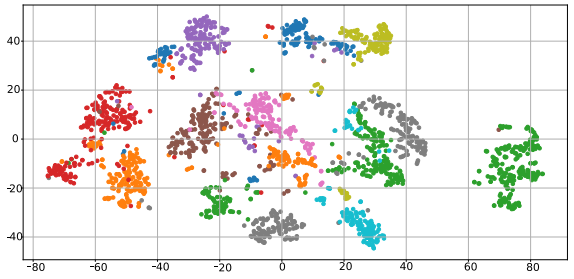
(a) *FacesUCR* in the latent space of the modified reduced VGG16 autoencoder



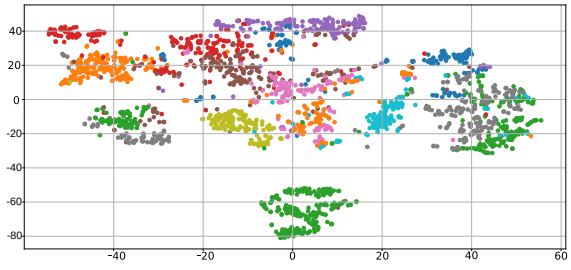
(b) *FacesUCR* in the latent space of the modified reduced multi-tasking VGG16 autoencoder



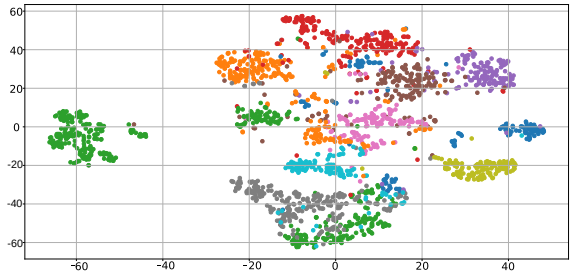
(c) *FacesUCR* in the latent space of the reduced Inception autoencoder



(d) *FacesUCR* in the latent space of the reduced multi-tasking Inception autoencoder



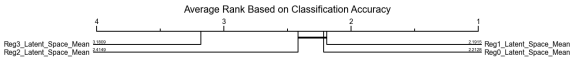
(e) *FacesUCR* in the latent space of the reduced ResNet autoencoder



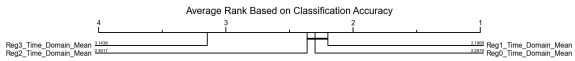
(f) *FacesUCR* in the latent space of the reduced multi-tasking ResNet autoencoder

Figure 4.10: acrshorttsne projections for the UCR archive's *FacesUCR* test datasets in the latent space of multi-tasking and basic autoencoder architectures

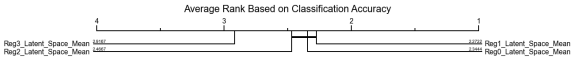
In Figure 4.11, we have presented the performance difference among the four $L2$ regularization setups, i.e., $L2 = [0, 0.0001, 0.001, 0.01]$. In the figure, the left column corresponds to the performance comparison based on latent space classification. In this context, the first three $L2$ regularization setups provided better performance. Even though the post hypothesis identified the second and the third $L2$ regularization to be statistically indifferent, the first $L2$ regularization setup obtained a better Friedman average rank. In reality, this is also evident in the time domain NCC accuracies. Thus, if the need for $L2$ regularization arises, we propose the utilization of $L2 = 0.0001$. With this in mind, in Table 4.6, we compare the standard deviation among the NCC accuracies obtained with the estimates of the different multi-tasking autoencoders. Overall, the average standard deviation of the latent space accuracies is below 5%. Moreover, in the time domain, the standard deviation of the NCC accuracies is below 7%. In addition to this, we also observe that the first $L2$ regularization has



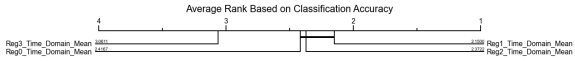
(a) Evaluation using NCC mean accuracies obtained in the latent space of the multi-tasing modified reduced VGG16 autoencoder



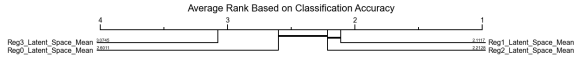
(b) Evaluation using NCC mean accuracies obtained with time domain estimates of the multi-tasking modified reduced VGG16 autoencoder



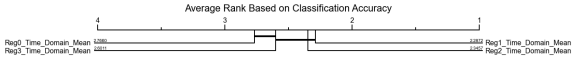
(c) Evaluation using NCC mean accuracies obtained in the latent space of the multi-tasking reduced Inception autoencoder



(d) Evaluation using NCC mean accuracies obtained with time domain estimates of the multi-tasking Inception autoencoder



(e) Evaluation using NCC mean accuracies obtained in the latent space of the multi-tasking reduced ResNet autoencoder



(f) Evaluation using NCC mean accuracies obtained with time domain estimates of the multi-tasking reduced ResNet autoencoder

Figure 4.11: Evaluation of the impact of L2 regularization on the quality of means estimated with multi-tasking autoencoder

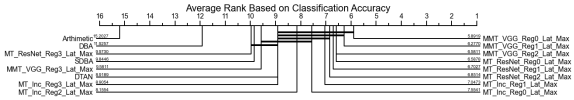
the lowest standard deviation on both latent space and time domain NCC accuracies. This further justifies our proposal of training the multi-tasking with this *L2* regularization. In general, we take the standard deviations to be acceptable since the overall network is optimizing for two kinds of losses. With this said, we next place our focus on the hypotheses tests. In this regard, Figure 4.12

Table 4.6: Standard deviation of NCC accuracies that are obtained using the multi-tasking: modified reduced VGG16, reduced Inception, and reduced ResNet autoencoders.

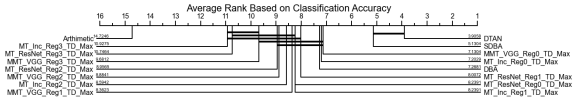
Techniques	$\pm\sigma$ in % L2 Reg0	$\pm\sigma$ in % L2 Reg1	$\pm\sigma$ in % L2 Reg2	$\pm\sigma$ in % L2 Reg3
MMT_VGG_Lat	4.39	3.67	4.36	5.13
MT_Inc_Lat	4.21	3.73	4.04	4.71
MT_ResNet_Lat	4.79	4.35	4.49	4.27
MMT_VGG_TD	5.87	5.46	5.69	6.77
MT_Inc_TD	5.77	5.36	5.47	6.32
MT_ResNet_TD	6.85	6.16	5.83	5.70

demonstrates the hypothesis evaluation conducted using the Nearest Centroid Classification (NCC) accuracies obtained on 74 UCR archive datasets. In the context of the multi-tasking autoencoders, the left column of the figure corresponds to outcomes obtained in the latent space of the autoencoders. On the contrary, the right column corresponds to outcomes obtained using the time domain estimates. We start our analysis of the hypotheses test from the comparison of the maximum NCC accuracies. In reality, for the alternatives, we used reported maximum NCC accuracies. In general, with the estimates of the variational and non-variational autoencoders, we could not beat the performances of most of the alternatives. However, in Figure 4.12 (b), the time domain estimates obtained with the multi-tasking modified reduced VGG16 and reduced Inception architectures were able to beat DBA. The architectures obtained this performance when trained with zero *L2* regularization. As we stated previously, we find this to be quite impressive since the estimates of the multi-tasking autoencoders are evaluated in DTW space which favors DBA. In addition to this, according to Figure 4.12 (a), we are

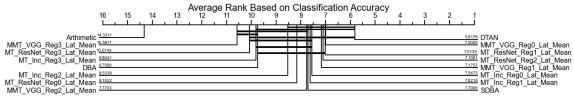
now able to obtain latent space registrations that are either better than or comparative to the state of the art, i.e., DTAN. In other words, we are in a sense mimicking the effects of multiple alignments in the latent space of the multi-tasking autoencoders. In this regard, the multi-tasking modified reduced VGG16 and reduced ResNet architectures are doing a good job.



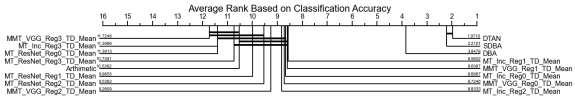
(a) Evaluation of maximum NCC accuracies obtained in the latent space of the multi-tasking autoencoders



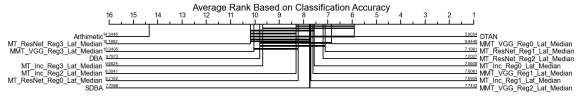
(b) Evaluation of maximum NCC accuracies obtained with time domain estimates of the multi-tasking autoencoders



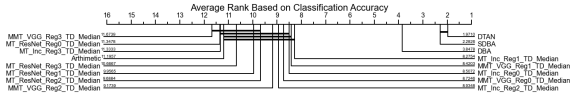
(c) Evaluation of mean NCC mean accuracies obtained in the latent space of the multi-tasking autoencoders



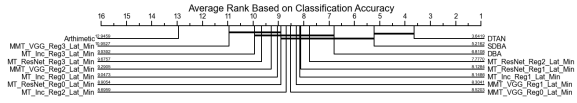
(d) Evaluation of mean NCC accuracies obtained with time domain estimates of the multi-tasking autoencoder



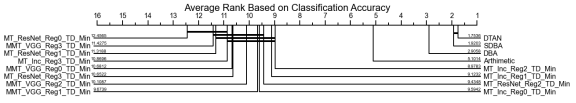
(e) Evaluation of median NCC accuracies obtained in the latent space of the multi-tasking autoencoders



(f) Evaluation of median NCC accuracies obtained with time domain estimates of the multi-tasking autoencoders



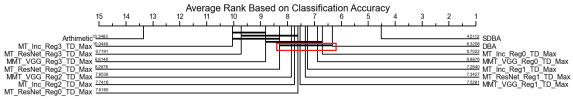
(g) Evaluation of minimum NCC accuracies obtained in the latent space of the multi-tasking autoencoders



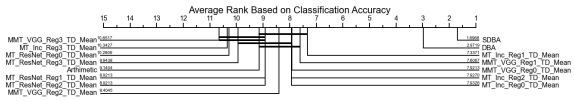
(h) Evaluation of minimum NCC accuracies obtained with time domain estimates of the multi-tasking autoencoders

Figure 4.12: CD diagram comparison of NCC accuracies obtained from the extended evaluation of multi-tasking autoencoders. These comparison is performed using NCC conducted on 74 UCR datasets using 25 repeated training trials.

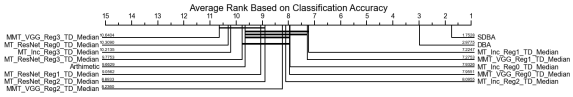
If we now focus on the mean and median accuracies, the multi-tasking estimates performed better than the arithmetic mean. However, in the worst case or in cases where the network is not generalizing well for various reasons, the time domain estimates are performing lower than an arithmetic mean. With these observations in mind, we next assess if the outcomes obtained with the maximum time domain NCC accuracies hold with the introduction of additional datasets. In order to make this assessment, we conducted 25 additional NCC tasks using 25 additional UCR archive datasets. In this regard, Figure 4.13 shows the comparison based on the multi-tasking autoencoder's time domain estimates and its counterparts. With the introduction of the additional 25 datasets, most of the performance of the multi-tasking autoencoder time-domain estimates are lower than DBA. However, some of the multi-tasking modified reduced VGG16 estimates performed similarly to DBA, i.e., when the network gets trained with the third $L2$ regularization. Besides these changes, the rest of the comparisons more or less remained the same. In general, the extended evaluation reveals that the multi-tasking setup could generate estimates close to DBA. Moreover, it also showed that by making proper adjustments to the layer arrangements of the architecture, the latent space could generate embedding that significantly mimics multiple alignments. With this said, we conclude this subsection by presenting the time domain estimates for the UCR archive's *ECG200* and *ECGFiveDays* datasets. Finally, in Figure 4.14, we have presented the estimates generated by the multi-tasking autoencoders, DBA, SDBA. Comparatively, the estimates generated by the multi-tasking VGG16 architectures are



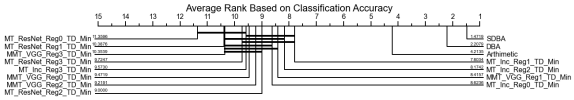
(a) Evaluation of maximum NCC accuracies obtained with time domain estimates of the multi-tasking autoencoders



(b) Evaluation of mean NCC accuracies obtained with time domain estimates of the multi-tasking autoencoders



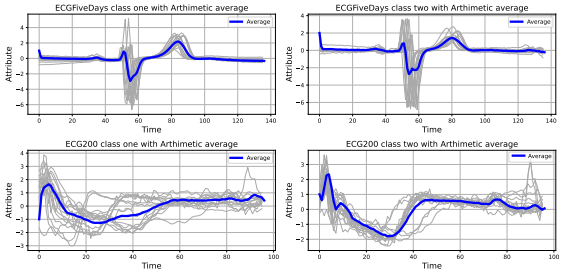
(c) Evaluation of median NCC accuracies obtained with time domain estimates of the multi-tasking autoencoders



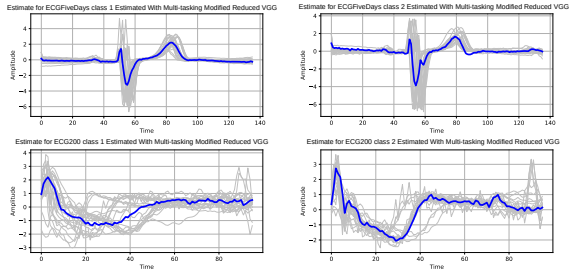
(d) Evaluation of minimum NCC accuracies obtained with time domain estimates of the multi-tasking autoencoders

Figure 4.13: CD diagram comparison of NCC accuracies obtained from the extended evaluation of multi-tasking autoencoders. These comparison is performed using NCC conducted on 89 UCR datasets using 25 repeated training trials.

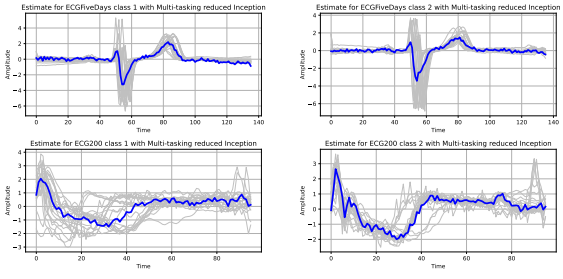
relatively better than its counterpart. Moreover, in some of its estimates, it gave shapes that resemble the estimates of DBA. For instance, the estimates of the multi-tasking autoencoder and DBA are relatively similar for the second class of the *ECGFiveDays*. In general, the NCC accuracies for the estimates are summarized in Table 4.7. The NCC accuracies presented in Table 4.7 shows that, the



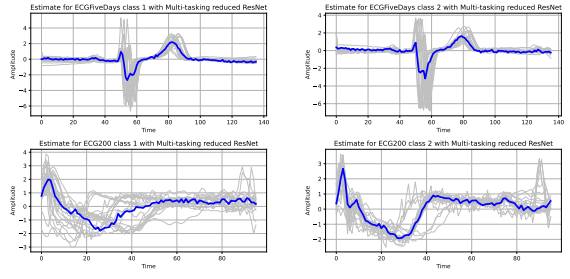
(a) Arithmetic means



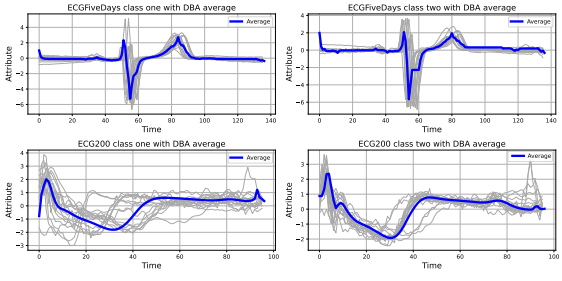
(b) Multi-tasking Modified Reduced VGG16 estimates



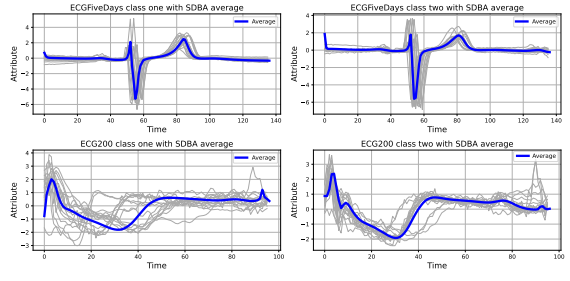
(c) Multi-tasking reduced Inception estimates



(d) Multi-tasking reduced ResNet estimates



(e) DBA estimates



(f) SDBA estimates

Figure 4.14: Averages estimated for the UCR archives *ECG200* and *ECGFiveDays* datasets using multi-tasking: modified reduced VGG16, reduced Inception, reduced ResNet, and alternative averaging techniques

estimates generated by the multi-tasking autoencoder have better representatives than their counterparts. With this said, we conclude the discussion in this subsection and proceed with the evaluation of the variational versions of the multi-tasking setups.

Table 4.7: NCC accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets that are obtained with multi-tasking autoencoders

Techniques	NCC for *ECG200* in %	NCC for *ECGFiveDays* in%
Arithmetic	67	52.96
DBA	65	52.15
SDBA	73	67.02
DTAN	**79**	**97.79**
MMT_VGG_TD	77	70.27
MT_Inception_TD	**78**	**72.36**
MT_ResNet_TD	**78**	**72.71**

4.2.5.2 Evaluation of Averages Estimated with Variational Multi-tasking Autoencoders

We start our assessment of the variational multi-tasking autoencoder with the wins/ties/losses analysis. However, the variational multi-tasking autoencoders also failed to converge on the datasets presented in Table 3.18 due to the high amplitude values of the datasets. Even though there is a chance we could mitigate this by introducing batch normalization layers, we refrained from doing so since we want to have a fair comparison among the multi-tasking setups, i.e., the variational and non-variational versions. However, we also found the multi-tasking setup based on the ResNet architecture also failed to converge for the *Fungi* dataset. In reality, the maximum amplitude value within the dataset ranges up to 80. Even though this is comparatively lower than the amplitude values observed for the datasets mentioned in Table 3.18, the ResNet was unable to converge. In this regard, as we stated earlier, the ResNet utilizes an *Addition* layer to combine skip connection features with the outputs of the *Convolutional* stacks. To this end, the ResNet continuously adds a constant offset to the means

Table 4.8: Comparison of wins/ties/losses obtained with the estimates of variational multi-tasking autoencoders and their counterparts. These comparisons are obtained using the best outcomes of NCC experiments that were conducted on 66 UCR archive datasets using 25 repeated training trials and four $L2$ regularization ($L2 = [0, 0.0001, 0.001, 0.01]$).

Techniques	$L2$ Reg. (x)	Wins	Ties	Losses
Arithmetic	-	0	0	66
DBA		0	1	65
DTAN		11	3	52
SDBA		8	1	57
Var_MMT_VGG_Regx_Lat	{0, 1, 2, 3}	{7, 7, 5, 3}	{5, 2, 4, 2}	{54, 57, 57, 61}
Var_MMT_VGG_Regx_TD		{2, 0, 0, 0}	{2, 0, 1, 1}	{62, 66, 65, 65}
Var_MT_Inc_Regx_Lat		{2, 0, 0, 3}	{3, 2, 3, 2}	{61, 64, 63, 61}
Var_MT_Inc_Regx_TD		{0, 0, 0, 0}	{1, 2, 0, 1}	{65, 64, 66, 65}
Var_MT_ResNet_Regx_Lat		{1, 1, 2, 3}	{2, 4, 5, 3}	{63, 61, 59, 60}
Var_MT_ResNet_Regx_TD		{0, 1, 0, 0}	{2, 3, 0, 1}	{64, 62, 66, 65}

of the latent features. This is contrary to the objective of variational autoencoders which aims to bring down the mean to zero. In addition to this, since we have now added a classification objective, it will further add to the difficulty of training the multi-tasking ResNet autoencoder. In general, given the circumstances, we find it not to be surprising that the multi-tasking ResNet failed to converge for the *Fungi* dataset. With this in mind, in Table 4.8, we have summarized the wins/ties/losses obtained with the variational multi-tasking autoencoders. Given the performances shown in Table 4.16 and the challenge associated with the network convergence, we found the estimates of the variational multi-tasking autoencoder to be relatively not encouraging. However, since we can not make conclusive remarks based on speculations, we next perform the statistical evaluations we have performed on previous setups. In this regard, we first analyze the overall NCC using a box-whisker plot.

Table 4.9 shows the statistics of the NCC accuracies corresponding to the box-whisker plot given in Figure 4.15. In reality, we observed no significant statistical improvement in the NCC accuracies obtained with the estimates of the variational and non-variational multi-tasking autoencoders. For instance, over the same 66 UCR archive datasets, the VGG16 based non-variational and variational autoencoders obtained a best case median accuracy of 79.21% and 79.31%. Moreover, for the Inception architecture, the accuracies were 78.21% and 77.92%. Finally, for the ResNet it is 79.26% and 76.82%. In general, we can associate the slight decrements in the performances of the variational autoencoder.

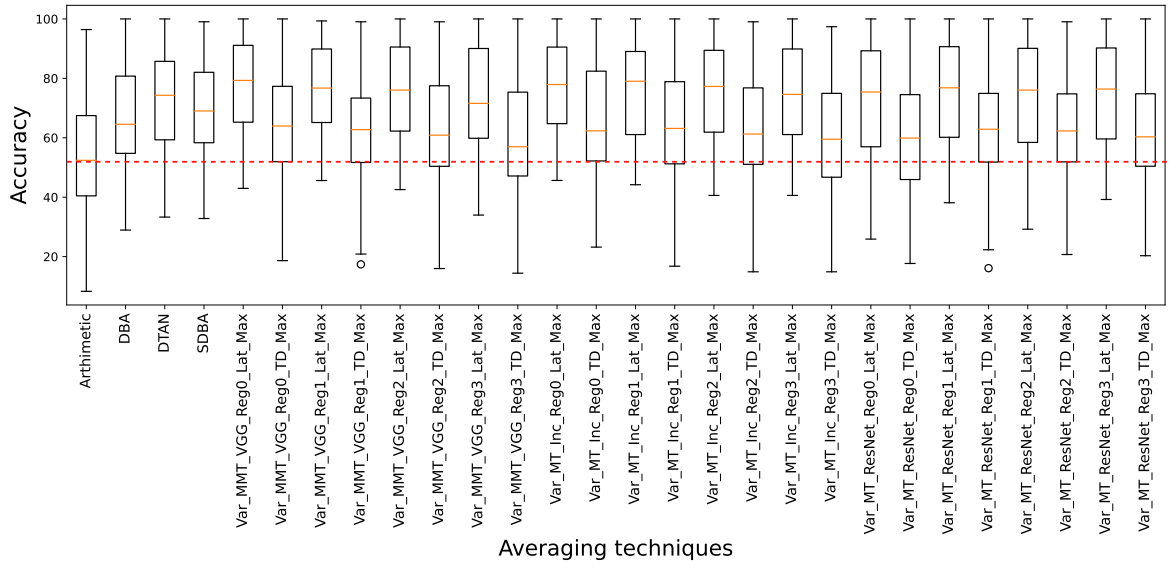


Figure 4.15: Box-whisker plot comparison of the NCC accuracies obtained with the variational multi-tasking autoencoders and their counterparts.

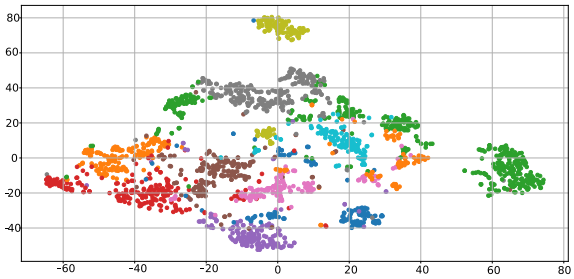
with the difficulty of their underlying objective function. In reality, we are now asking the autoencoders to extract features that are: normally distributed, separable, and reconstructible. This is relatively difficult compared to the requirement placed on non-variational multi-tasking autoencoders, i.e., extracting features that are separable and reconstructible. However, even under this difficulty, the variational multi-tasking autoencoders are performing relatively close to DBA and SDBA. In this aspect, if we see the median accuracies, we can observe that DBA and SDBA respectively obtained a 65.54% and 69.02% performance. In this aspect, the best median NCC accuracies of the variational multi-tasking

Table 4.9: Statistics assessment of the NCC accuracies that are obtained with the multi-tasking: modified VGG16, reduced Inception, and reduced ResNet architectures. These assessments were conducted using the maximum NCC accuracies on 65 UCR archive datasets using the different averaging techniques

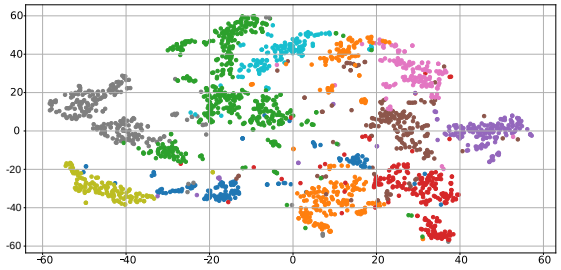
Techniques	Bot. whisker	Top whisker	25% Quant.	75% Quant.	Median
Arithmetic	8.31	96.43	40.56	69.23	52.41
DBA	28.94	100	54.39	80.75	64.54
DTAN	33.31	100	59.31	85.72	74.30
SDBA	32.83	99.05	58.33	82.04	69.02
Var_VGG_Regx_Lat x={0, 1, 2, 3}	{42.97, 45.61 42.56, 33.98 }	{100,99.33, 100, 100 }	{62.25, 65.14 62.25, 59.84}	{91.11, 89.89 90.55, 90.06 }	{79.31, 76.73 76.05, 71.57}
Var_VGG_Regx_TD x={0, 1, 2, 3}	{18.65, 20.86 15.99, 14.42}	{100, 99.05 99.05, 100}	{51.95, 51.69 50.39, 47.14 }	{77.32, 73.37 77.51, 75.39}	{63.96, 62.73 60.88, 56.99 }
Var_Inc_Regx_Lat x={0, 1, 2, 3}	{45.64, 44.21 40.59, 40.59 }	{100, 100 100, 100 }	{64.76, 61.08 61.89, 61.08}	{90.53, 89.05 89.44, 89.89}	{77.92, 79.03 77.28, 74.59}
Var_Inc_Regx_TD x={0, 1, 2, 3}	{23.19, 16.77 14.89, 14.89}	{100, 100 99.05, 99.05}	{52.19, 51.23 51.06, 46.72}	{68.54, 65.65 64.33, 60.56}	{62.34, 63.14 61.26, 59.51}
Var_ResNet_Regx_Lat x={0, 1, 2, 3}	{25.89, 38.13 29.23, 39.23}	{100, 100 100, 100}	{56.97, 60.16 58.43, 59.60}	{89.28, 90.67 90.11, 90.23}	{75.40, 76.82 76.06, 76.41 }
Var_ResNet_Regx_TD x={0, 1, 2, 3}	{17.69, 22.31 20.71, 20.30}	{100, 100 97.05, 100}	{45.95, 51.82 51.89, 50.42}	{74.52, 74.94 74.78, 74.82}	{59.89, 62.87 62.31, 60.31}

autoencoders are within the ranges of 62.87% and 63.96%. This makes them to still a significantly better performing averaging technique compared to the arithmetic mean, i.e., as shown in Figure 4.15. This in turn implies that, in the latent space, the variational multi-tasking autoencoders can extract features that mimic the effects of multiple alignments. In order to demonstrate this visually, we revisit the UCR archive's *FacesUCR* dataset and present its latent space t-SNE projections. In this regard, Figure 4.16 demonstrates the acrshorttsne projection of the dataset in the latent spaces of the non variational and variational versions of the multi-tasking autoencoders. In Figure 4.16, the projections on the left and right column corresponds to the latent embedding obtained with the non variational and variational multi-tasking autoencoders. The figures show that both versions of the multi-tasking autoencoders have the capability of extracting dense and separable latent features. However, the question now becomes, how good are the latent embedding of the variational autoencoders in the context of the time domain estimates. In order to address this question, we assess the NCC accuracies using a hypothesis tests. In this aspect, we follow the same procedure and first compare the accuracies associated with: DTAN, DBA, SDBA, arithmetic, and the estimates of the variational multi-tasking autoencoders.

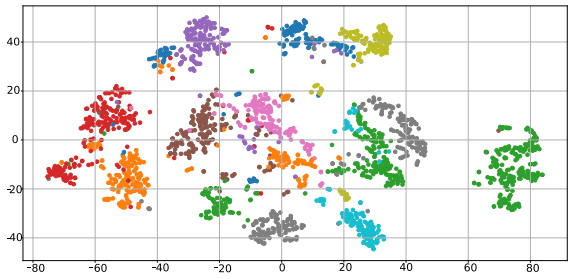
In Figure 4.17, we compare the latent space (left column) and time domain (right column) NCC accuracies of the variational multi-tasking autoencoders. Similar to their non-variational counterpart, the variational multi-tasking autoencoders obtained a better latent space registration compared to the state of the art (DTAN). This is evident in Figure 4.17 (a) where the variational multi-tasking autoencoder based on the VGG16 architecture is performing better than its counterparts, i.e., while it is trained using the first three *L2* regularization configurations. Moreover, the variational Inception



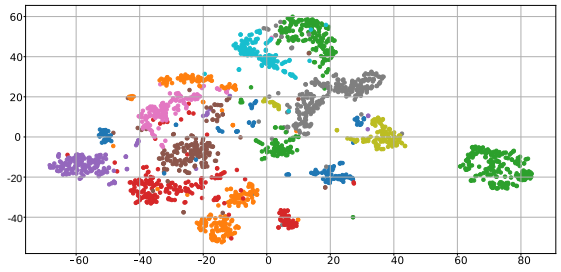
(a) *FacesUCR* in the latent space of the multi-tasking modified reduced VGG16 autoencoder



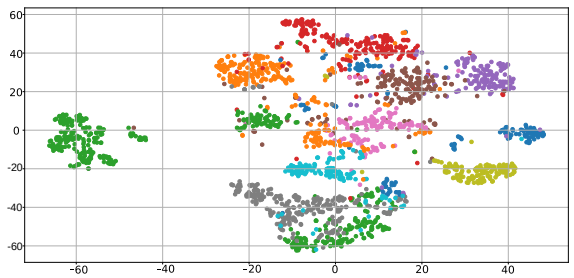
(b) *FacesUCR* in the latent space of the variational multi-tasking modified reduced VGG16 autoencoder



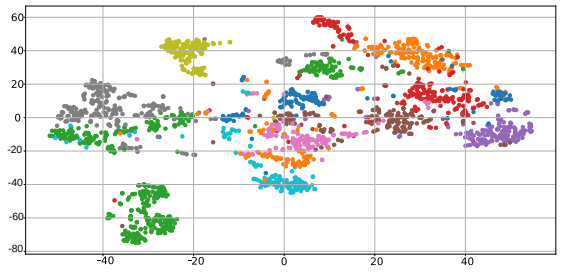
(c) *FacesUCR* in the latent space of the reduced multi-tasking Inception autoencoder



(d) *FacesUCR* in the latent space of the variational reduced multi-tasking Inception autoencoder



(e) *FacesUCR* in the latent space of the reduced multi-tasking ResNet autoencoder

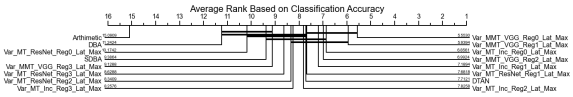


(f) *FacesUCR* in the latent space of the variational reduced multi-tasking ResNet autoencoder

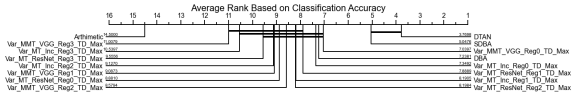
Figure 4.16: t-SNE projections for the UCR archive's *FacesUCR* test datasets in the latent space of variational and non variational multi-tasking autoencoder architectures

architecture is performing better than DTAN when trained with the first two $L2$ regularization. However, the ResNet based architecture barely outperforms DTAN when it is trained with the second $L2$ regularization. If we now place our focus on the time domain NCC accuracies, the estimates obtained with some of the variational multi-tasking autoencoders outperform DBA, i.e., as shown in in Figure 4.17 (b). In this regard, the architecture based on the VGG16 performs better than DBA when it is trained with zero $L2$ regularization. In general, we found the time domain and latent space performances of the variational versions to be more or less similar to their basic counterparts. However, to assess this claim statistically, we compared the NCC accuracies obtained with both setups on 75 UCR archive datasets. In this regard, Figure 4.18 shows the comparisons of latent space and time domain maximum and median NCC accuracies. However, for better clarity, we only took the NCC accuracies that are associated with zero $L2$ regularization. In general, Figure 4.18 (a) shows that in the latent space the non variation VGG16 based multi-tasking autoencoder is performing better. In reality, the post-hypothesis test identified that its performance is statistically indifferent to its variational

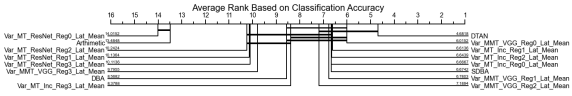
form. However, despite this equivalence, the variational version of the VGG16 is performing lower than its non variational ResNet counterpart.



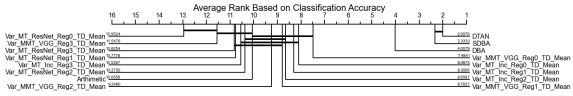
(a) Evaluation of maximum NCC accuracies obtained in the latent space of the variational multi-tasking autoencoders



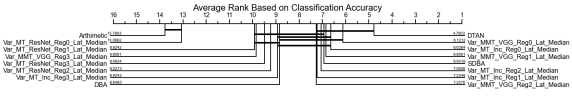
(b) Evaluation of maximum time domain NCC accuracies obtained with variational multi-tasking autoencoders



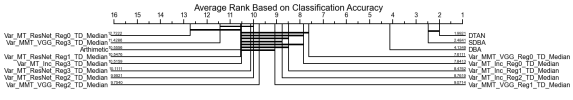
(c) Evaluation of mean NCC accuracies obtained in the latent space of the variational multi-tasking autoencoders



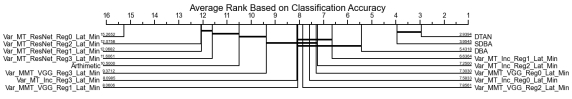
(d) Evaluation of mean time domai NCC accuracies obtained with variational multi-tasking autoencoder



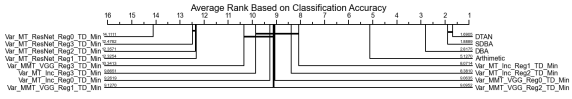
(e) Evaluation of median NCC accuracies obtained in the latent space of the variational multi-tasking autoencoders



(f) Evaluation of median time domain NCC accuracies obtained with variational multi-tasking autoencoders



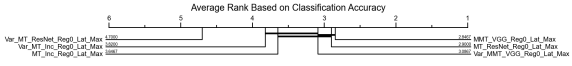
(g) Evaluation of minimum NCC accuracies obtained in the latent space of the variational multi-tasking autoencoders



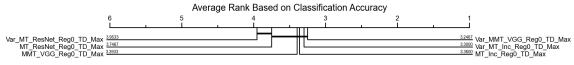
(h) Evaluation of time domain minimum NCC accuracies obtained with variational multi-tasking autoencoders

Figure 4.17: CD diagram comparison of NCC accuracies obtained from the evaluation of variational multi-tasking autoencoders. These comparison is performed using NCC conducted on 66 UCR datasets using 25 repeated training trials.

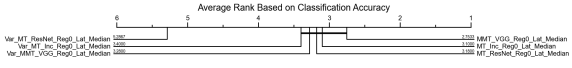
In general, we find the non variational version of the multi-tasking autoencoders are better in latent space registration. We associate this better performance to the relatively lower constraint placed on their allowable latent embedding. However, in the time domain, the variational versions of the



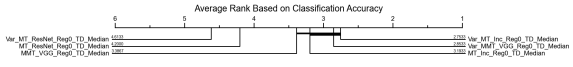
(a) Evaluation of maximum NCC accuracies obtained in the latent space of the variational and non variational multi-tasking autoencoders



(b) Evaluation of maximum NCC accuracies obtained with the time domain estimates of variational and non variational multi-tasking autoencoders



(c) Evaluation of median NCC accuracies obtained in the latent space of variational and non variational multi-tasking autoencoders

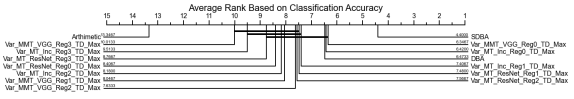


(d) Evaluation of median NCC accuracies obtained with time domain estimates of the variational and non variational multi-tasking autoencoder

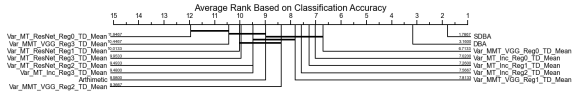
Figure 4.18: CD diagram comparisons of NCC accuracies obtained with the variational and non variational multi-tasking autoencoders. The comparison is performed using the NCC accuracies obtained from 75 UCR archive datasets.

multi-tasking autoencoders are slightly performing better than their non-variational counterparts, i.e., Figures 4.18 (b) and 4.18 (d). In this regard, the variational VGG16 and Inception are slightly performing better. In reality, the decoders of the variational multi-tasking autoencoders are expected to reconstruct latent embedding confined to a smaller region which gives them a better chance of generalizing. However, even with this advantage, we find the difference among the two versions of the autoencoders to be relatively small. In general, based on the experimental evaluations, we

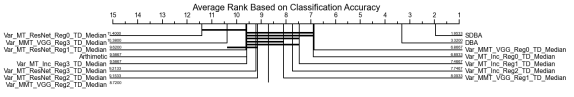
found no significant difference among the two multi-tasking setups. However, in terms of training, the variational version of the multi-tasking setups are relatively difficult to train since the overall optimization setup is expected to meet relatively more constraints. In addition to this, they also require data pre-processing if the amplitude of the averaged set is large. On the contrary, we have not seen such requirement with the non variational versions. In general, if the multi-tasking setup is to be used, we suggest the non-variational versions to be deployed. With this said, we conclude the statistical comparison of the NCC accuracies by presenting the comparison of the variational multi-tasking autoencdoer's estimate with: SDBA,DBA and arithmetic mean. In this comparison, we have included the NCC accuracies of additional 19 UCR archive datasets. We expect the additional experiment to help us validate if the performance observed with the time domain maximum NCC accuracies holds, i.e., the one shown in Figure 4.18 (b). In this context, Figure 4.19 (a) shows that some of the multi-tasking variational autoencoders are still performing better than DBA. This is evident



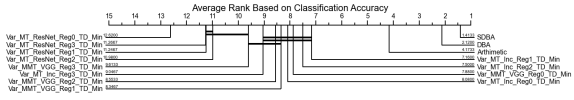
(a) Evaluation of maximum NCC accuracies obtained with the time domain estimates of variational multi-tasking autoencoders



(b) Evaluation of mean NCC accuracies obtained with the time domain estimates of variational multi-tasking autoencoders



(c) Evaluation of median NCC accuracies obtained with the time domain estimates of variational multi-tasking autoencoders



(d) Evaluation of minimum NCC accuracies obtained with the time domain estimates of variational multi-tasking autoencoders

Figure 4.19: CD diagram comparisons of NCC accuracies that are obtained using the time domain estimates of the variational multi-tasking autoencoders and their counterpart. The comparison is performed using the NCC obtained on 75 UCR archive datasets.

when architectures that are based on VGG16 and Inception gets trained with zero *L2* regularization. However, it should also be noted that as compared to the comparison made in Figure 4.13, we are assessing performance on a relatively lower number of datasets since we now have datasets that the network failed to converge for. Moreover, on a comparable number of datasets, the non variational version of the autoencoders were also performing better than DBA, i.e., Figure 4.12. Based on these assessments, we can safely conclude that there is no major difference among the two setups, i.e., despite the convergence problem associated with the variational versions.

With these observations in mind, we will conclude this subsection by presenting the time domain estimates corresponding to the UCR archive's *ECG200* and *ECGFiveDays* as a visual demonstration. However, before proceeding to the visual demonstration, we first assess the variance among the NCC accuracies obtained with the different setups. In this context, Table 4.10 shows the standard deviation (σ) among the NCC accuracies. Conforming to our previous argument, i.e., the ResNet continuously adds a constant offset that makes its training difficult , its standard deviation is higher than its counterparts. In general, in most cases the zero regularization gave a relatively reproducible results, i.e., it has narrow variance. Moreover, comparatively, the outcomes of the Inception architec-

ture is relatively stable. With this said, in Figure 4.20 we have presented the estimates generate using the variational and non variational autoencoders for a better comparion.

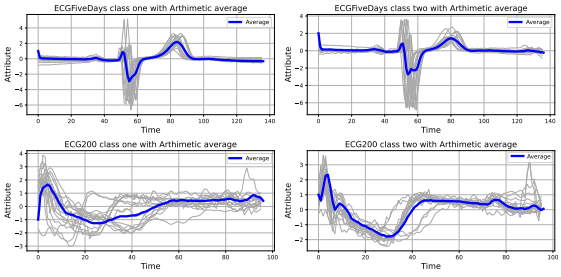
Table 4.10: Standard deviation of NCC accuracies that are obtained using the variational multi-tasking: modified reduced VGG16, reduced Inception, and reduced ResNet autoencoders.

Techniques	$\pm\sigma$ in % L2 Reg0	$\pm\sigma$ in % L2 Reg1	$\pm\sigma$ in % L2 Reg2	$\pm\sigma$ in % L2 Reg3
Var_MMT_VGG_Lat	3.85	4.48	5.05	4.21
Var_MT_Inc_Lat	4.49	3.73	4.68	3.99
Var_MT_ResNet_Lat	12.68	9.09	9.28	8.30
Var_MMT_VGG_TD	5.92	5.92	6.70	5.95
Var_MT_Inc_TD	5.50	5.32	7.35	5.44
Var_MT_ResNet_TD	9.59	8.55	8.560	8.0

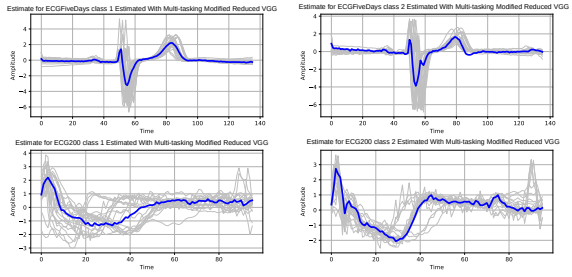
In general, the variational and non variational setups were able to obtain the NCC accuracies summarized in Table 4.19. In overall, for the two datasets, the estimates of the non variational autoencoders performed better. With these observations in mind, we proceed with the further investigation of the multi-tasking setup in the following two consecutive sub sections. In the further investigation, we try to fill gaps on the objective function of the multi-tasking setup which we believe could further improve the quality of the time domain estimates.

Table 4.11: NCC accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets that are obtained with multi-tasking autoencoders

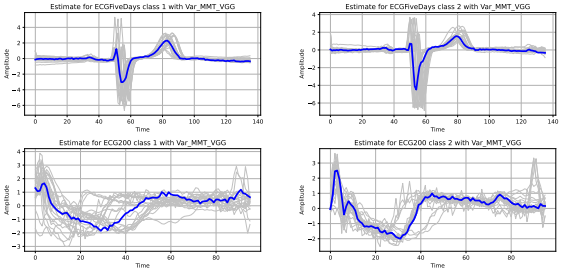
Techniques	NCC for *ECG200* in %	NCC for *ECGFiveDays* in%
Arithmetic	67	52.96
DBA	65	52.15
SDBA	73	67.02
DTAN	**79**	**97.79**
MMT_VGG_TD	77	70.27
MT_Inception_TD	**78**	72.36
MT_ResNet_TD	**78**	**72.71**
Var_MMT_VGG_TD	73	70.15
Var_MT_Inception_TD	77	70.49
Var_MT_ResNet_TD	77	**74.33**



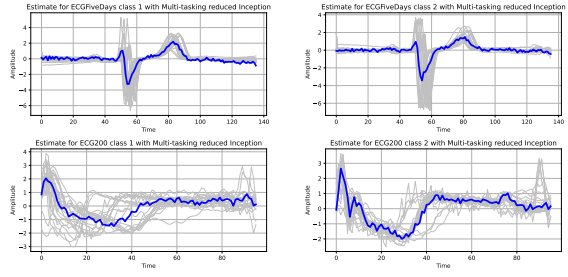
(a) Arithmetic means



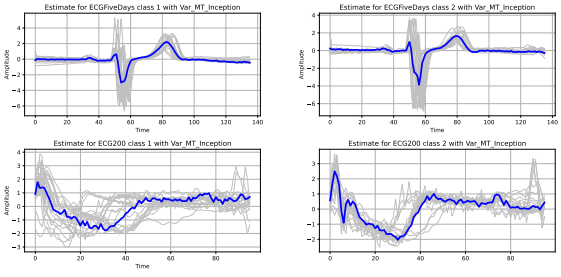
(b) Multi-tasking modified reduced VGG16's estimates



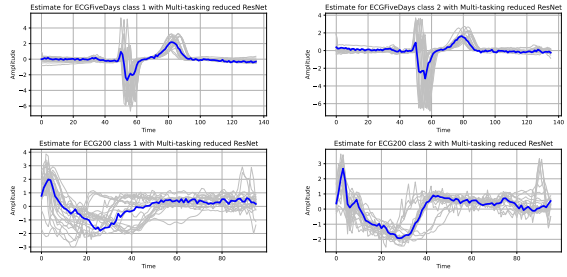
(c) Variational multi-tasking modified reduced VGG16's estimates



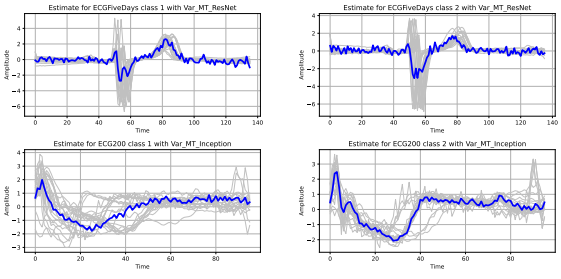
(d) Multi-tasking reduced Inception's estimates



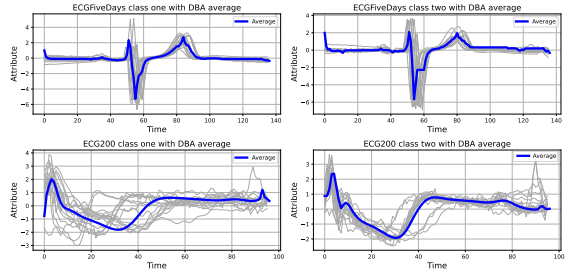
(e) Variational multi-tasking reduced Inception's estimates



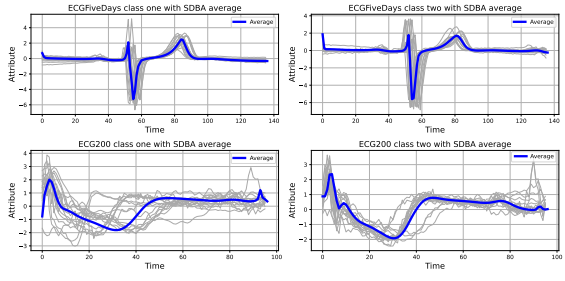
(f) Multi-tasking reduced ResNet's estimates



(g) Variatinoal multi-tasking reduced ResNet's estimates



(h) DBA's estimates



(i) SDBA's estimates

Figure 4.20: Averages estimated for the UCR archives *ECG200* and *ECGFiveDays* datasets using variational multi-tasking: modified reduced VGG16, reduced Inception, reduced ResNet, and alternative averaging techniques

4.3 Time Series Averaging Using a Multi-tasking Quantile Regression Autoencoder

In this section, we address the limitations observed in the objective function of the multi-tasking networks. In this regard, we first focus on improving the compactness of the extracted latent space reforestation. In this aspect, we first propose to introduce the latent space WGSS loss function given in (4.2), where C and K_i are the number of categories (classes) and the number of training samples per a given class. Moreover, μ_i is the arithmetic mean of the latent space representations of input series that belong to a given class C_i. We expect this loss to encourage and force the encoder to extract latent features centered around the latent means. Moreover, since (4.2) forces the per class latent feature to minimize their discrepancy with the latent means, i.e., the common landmark, the features are expected to be within the neighborhood of each other. In reality, we make (4.2) only visible to the encoder portion of the network for two reasons. First, extracting a dense latent feature is mainly the responsibility of the encoder. Secondly, we desire that the encoder takes the compactness of the extracted latent space representations into account while adjusting its weights to meet the demands of the classifier and decoder.

$$WGSS_{Latent}(Z, \mu) = \frac{1}{C}\sum_{i=1}^{C}\frac{1}{K_i}\sum_{j=1}^{K_i}||Z_j - \mu_i||_{l2} \tag{4.2}$$

In practice, the extraction of dense latent features could either improve or degrade the quality of the projected estimates. In the worst case, if the latent features are very dense and overlapping, the decoder will have difficulties distinguishing among the latent space representations of the input datasets. However, since the encoder is informed about the demands of reconstructability and distingushibility through the losses of the decoder and classifier, there is a low probability of extracting overlapping latent space representations. On the contrary, under such constraints, we expect a latent means to be within the neighborhood of the latent embedding of multiple input series. To this end, we expect the decoder to have a higher likelihood of projecting a latent means into a time domain projection that highly resembles a range of input series. In other words, by increasing the compactness of the latent features, we are indirectly trying to increase the decoder's ability to interpret latent space neighborhood points. In reality, there are also alternative mechanisms that could further assist this objective. For instance, in autoencoders, we utilize reconstruction loss to re-project latent features. However, in practice, we do not expect a reconstructed series to be an exact copy of its input counterpart. On the contrary, in the time domain, we expect the reconstructed series to be a near neighborhood of its input counterpart. Thus, if we see this from a different perspective, we can think of the reconstructed series as additional input examples. With this in mind, we propose to utilize the latent space embedding of the reconstructed output of the decoder. With this in mind, we propose to introduce a mean squared error between the latent representation of the reconstructed series ($\hat{Z}$) and their input counterparts (Z) using (4.3). Moreover, we intend to make this loss only visible to the decoder portion of the multi-tasking networks. In reality, the advantages of incorporating (4.3) into the objective function of the decoder are twofold. First, we help the decoder further assess its reconstruction capability in the latent space, i.e., in addition to the time domain. Additionally,

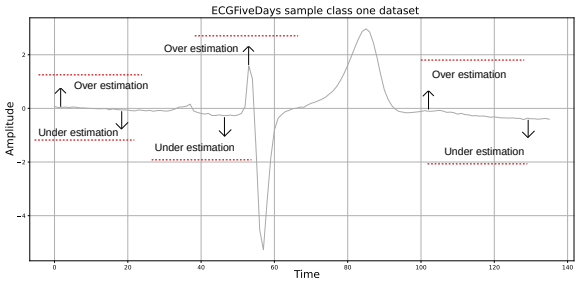
by introducing (4.3), we are making the latent space of the multi-tasking autoencoders relatively continuous from the perspective of the decoder. This is because, as the training progresses, the decoder slowly learns to reconstruct the input datasets more optimally. Thus, when this happens, the latent space projection of the reconstructed series will be in the near neighborhood of the input series's latent embedding. To this end, we are now indirectly providing the decoder with a piece of additional information on how to interpret (map) neighborhood points. This, in turn, helps the decoder to be less sensitive to small latent space perturbances. In reality, this is a positive effect in the context of projecting the latent space means. This is because, in this case, we will have relatively lower blind spots that the decoder fails to interpret optimally.

$$MSE_{Latent}(Z, \hat{Z}) = \frac{1}{C}\sum_{i=1}^{C}\frac{1}{K_i}\sum_{j=1}^{K_i}||Z_j - \hat{Z}_j||_{l2} \tag{4.3}$$

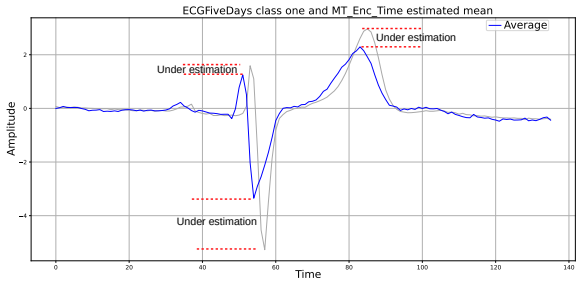
With these improvements in mind, we next place our focus on relaxing the reconstruction loss of the decoder. In this aspect, the first concern we address is the susceptibility of the Mean Squared Error (MSE) to outliers. This is because, due to the squaring operation in MSE, the reconstruction error of outliers gets significantly magnified. Thus, in such cases, the gradients of the decoder are often pulled in an undesired direction. One possible way of overcoming this challenge would be to change MSE with Mean Absolute Error (MAE) (5.10), where X, $\hat{X}\ \epsilon\ \mathbb{R}^M$ are the input and re-constructed series. Moreover, N is the number of series within the averaged set.

$$L(X,\ \hat{X}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M}\sum_{j=1}^{M}|x_{j,\ i} - \hat{x}_{j,\ i}| \tag{4.4}$$

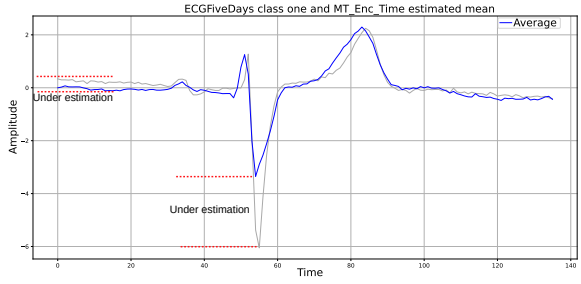
In practice, due to the removal of the squaring operation, the MAE is less sensitive to the effects of outliers pulling the decoder in an undesired direction. However, like its MSE counterpart, the MAE encourages a median reconstruction (regression). This is better demonstrated in Figure 4.21 (a) where we have indicated possible reconstruction cases using the UCR archive's *ECEGFiveDays* dataset. Based on Figure 4.21 (a), we can identify three possible reconstruction cases: over, under or perfect reconstruction. However, in practice, MSE or MAE error functions equally penalize over and under estimations (reconstructions). To this end, since a perfect reconstruction is often not guaranteed, we expect the final reconstruction for a given dataset to be along a median line that is in between the two extremes. In addition to this, since the decoder is optimizing for an average reconstruction loss (4.1), we can not expect the decoder to learn a perfect reconstruction for the individual datasets. To this end, when we re-project the latent arithmetic averages, we expect the individual reconstruction errors to aggregate and pull down the projected means close to the average (median) of the median reconstruction lines. This is better demonstrated in Figure 4.21 (d) where we plotted the reduced VGG16 multi-tasking autoencoder's estimation for the first class of the UCR archive's *ECGFiveDays* dataset. In the figure, we can observe that amplitudes of major descriptive features are often close to the median reconstruction line. This, in turn, makes a latent mean re-projection based on reconstruction losses to be relatively close and at times similar to an arithmetic mean, i.e., as shown in Figure 4.3 (b). To this end, we aim to change the decoder's objective function in a way that we have better control of



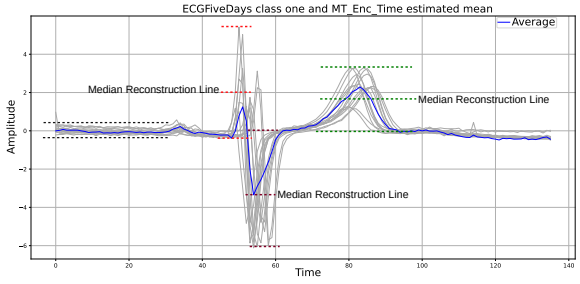
(a) Possible cases of reconstruction



(b) An estimated average and a member of the averaged set



(c) Comparison of an estimated average and a member of the averaged set



(d) An estimated average as compared to the median reconstruction line

Figure 4.21: Visual demonstration of an estimated average and the median reconstruction line

the location of the median reconstruction lines. Thus, this way, we have better control over the quality of the projected estimations. With these in mind, we identified the quantile regression loss given in (4.5) to align with the current objective, where $0 \leq \lambda \leq 1$ and $X, \hat{x} \epsilon \mathbb{R}^M$. Quantile regression sees the reconstruction problem from three angles: overestimation, underestimation, and perfect estimation. In quantile regression, we say an overestimation has occurred if $(X - \hat{X}) < 0$. On the contrary, an underestimation occurs when $(X - \hat{X}) > 0$.

$$L_Q(\lambda, X, \hat{X}) = \max\{\lambda (X - \hat{X}), (1 - \lambda) (X - \hat{X})\} \tag{4.5}$$

With this understanding, we can safely assume $\lambda (X - \hat{X})$ relates to underestimation since $0 \leq \lambda \leq 1$. On the contrary, $(1 - \lambda) (X - \hat{X})$ corresponds to overestimation. Moreover, in (4.5), λ determines how much of the under or overestimations the network penalizes. To this end, we can identify three scenarios which can favour under, over or median reconstruction (estimation): $\lambda < 0.5$, $\lambda > 0.5$ and $\lambda = 0.5$. In the first case, i.e., $\lambda < 0.5$, (4.5) encourages underestimation and discourages overestimation. This is because, the error for the under estimation ($(X - \hat{X}) > 0$) has a weighting factor of $\lambda < 0.5$. On the contrary, a $\lambda > 0.5$ discourages underestimation since now $(X - \hat{X}) > 0$ a weighting factor of $\lambda > 0.5$. Finally, at $\lambda = 0.5$, quantile regression behaves asMAE. This is because it penalizes both over and under estimations equally. However, for our case, we have reconfigured the quantile regression loss so that it penalizes over (under) estimation equally by a factor that is different from 0.5. In order to make this possible, we compute the quantile regression for a pair of λ values rather than a single λ value, i.e., λ ($\lambda = [\lambda_1, \lambda_2]$). Moreover, after computing the quantile regression loss based on the two λ values, we propose to take the maximum of the two quantile regression losses as shown in (4.6). We make this modification with the intention of defining additional reconstruction

scenarios while keeping the already available ones intact. For instance, if we set both λ values equal to each other but not necessarily to 0.5, then (4.6) will be an improved version of the basic quantile regression given in (4.5). This is because we can now discourage both over and under estimations by a factor different from 0.5.

$$L_{CQ}([\lambda_1, \lambda_2],\ X,\ \hat{X}) = \max\{L_Q(\lambda_1,\ X,\ \hat{X}), L_Q(\lambda_2,\ X,\ \hat{X})\} \tag{4.6}$$

However, if we set them to be different say $\lambda = [0.75,\ 0.25]$, then we will be penalizing over or underestimation equally by a factor of 75% (0.75). This is because, if we assume underestimation has occurred ($(X - \hat{X}) > 0$), then the first λ penalizes it by 75%. On the contrary, the second penalizes it by 25%. However, since we are taking the maximum of the two computations, we will end up with a 75% penalization. With the same analysis, if underestimation occurs ($(X - \hat{X}) < 0$), then the first λ penalizes it by 25% and the second λ by 75%. However, when we take the maximum of the two, we penalize overestimation by 75%. Additionally, we can also favor over and under estimations by setting both λ values to be equal. In this aspect, if we set both λ values to be less than 0.5, we will encourage over-estimation and vice versa. One additional point to note here is that the quantile regression loss analyzes the time series at a timestamp level. This is because we have not utilized any sort of norming operation on the errors. To this end, the over or under estimation gets performed on each timestamp as if we are performing regression. In reality, this provides a more refined control on the median reconstruction line compared to MSE and MAE. This, in turn, is expected to help us avoid latent space mean re-projections that resemble arithmetic mean.

In practice, the advantage of using quantile regression is not limited to shifting the median reconstruction line. On the contrary, since we now using a reduced weighting factor for the reconstruction loss, i.e., loss ($0 < \lambda < 1$), we are encouraging the classifier to have more say on the latent space features it extracts. Thus, we also indirectly increase the probability of obtaining highly separable latent features. With these technicalities in mind, we propose to customize the objective functions of the three modules of our multi-tasking setup as follows. We propose the encoder to optimize for the losses given in (4.7). However, we propose the classifier and the decoder to respectively optimize for (4.8) and (4.9). With this said, we will first present the preliminary experimental evaluations and later and continue with the extended evaluation.

$$L_{encoder}([\lambda_1,\ \lambda_2],\ X,\ \hat{X},\ Z, Cat, P_{cat}) = L_{CQ}([\lambda_1,\ \lambda_2],\ X,\ \hat{X}) - L_{cat}(Cat, p_{cat}) + WGSS_{Latent}(Z, \mu) \tag{4.7}$$

$$L_{classifier}(Cat, p_{cat}) = -L_{cat}(Cat, p_{cat}) \tag{4.8}$$

$$L_{Decoder}([\lambda_1,\ \lambda_2],\ X,\ \hat{X},\ Z,\ \hat{Z}) = L_{CQ}([\lambda_1,\ \lambda_2],\ X,\ \hat{X}) + MSE_{Latent}(Z, \hat{Z}) \tag{4.9}$$

4.3.1 Proposed Architectures

We evaluated the proposed modifications on the objective function using the modified reduced VGG16, ResNet and Inception version two architectures shown in Figure 4.6, 4.7 and 4.8. However, in this evaluation, we have removed the last transposed *Convolutional* layer of the decoder. Additionally, we

have not evaluated the variational variants of the multi-tasking autoencoders since we have not so far seen a significant change while utilizing them.

4.3.2 Experimental Setups

We have proposed to train the non-variational versions of the multi-tasking autoencoders using an 80/20 train and validation splits. Moreover, we aim to use two sets of λ configurations, i.e, λ_{conf1} = [(0.85, 0.15), (0.75, 0.25), (0.65, 0.35), (0.5, 0.5)] and λ_{conf2} = [(0.85, 0.85), (0.75, 0.75), (0.65, 0.65), (0.15, 0.15), (0.25, 0.25), (0.35, 0.35)]. In reality, the first set of λ pair discourages either over or under estimations by a factor $0.5 \leq \lambda \leq 0.85$. However, unlike MSE and MAE, the configuration leaves a little room for under and overestimations. On the contrary, in the second λ pair configuration, if $\{\lambda_1, \lambda_2\} < 0.5$, then we will be encouraging underestimations whenever they occur. However, if $\{\lambda_1, \lambda_2\} > 0.5$, then we will encourage over estimations whenever they occurs. With this understanding, afterward, we will call λ pair that discourage over (under) estimation as λ_{conf1}. On the contrary, we will call the configurations that encourage over (under) estimation as λ_{conf2}. With these terminologies in mind, we first propose to train each multi-tasking autoencoders using the λ configurations given in $\lambda_{config1}$ for 1500 epochs on 84 UCR datasets. Following this training, for each neural network configuration and training dataset, out of the estimations based on different λ pair vales, we take the one that obtained the maximumNCC accuracies. We then aim to compare these outcomes to the outcomes of the alternatives, i.e., DBA, SDBA, DTAN and the basic multi-tasking autoencoder. After performing this comparison, we aim to select the best performing quantile multi-tasking regression network. We then aim to train the network using: $\lambda_{config1}$ configuration, the 114 UCR datasets, 1500 epochs, and 25 repeated trials. We then compare the maximum, minimum, median, and mean NCC accuracies of the network to its counterparts. Additionally, we also aim to train the best performing architecture using $\lambda_{config2}$ to assess the implication of encouraging over and underestimation. However, we will train this setup for 1500 epochs and single trials over the 114 UCR datasets. We will present the repeated trial evaluations of this and the remaining network configurations in the extended evaluation. With this said, we will proceed with the discussion of our experimental evaluations.

4.3.3 Experimental Results and Interpretation

Similar to the steps taken in the previous evaluations, we start our assessment of the multi-tasking quantile regression networks with a wins/ties/losses analysis. However, we divide our non-extended evaluation into two segments, i.e., evaluations based on 84 and 114 datasets. The evaluations based on 84 datasets include the NCC accuracies reported for DTAN. However, since the evaluation of DTAN on the additional 30 datasets is not available, we have excluded it from the 114 datasets comparison. With this said, in Table 4.12, VGG_Quant_Lat (Time) and VGG_Quant_OU_Lat (Time) corresponds to the latent space (time domain) wins/ties/losses evaluations of the quantile regression setup that is based on the VGG16 architecture. Similarly, the outcomes of the architectures based on the ResNet and Inception architectures gets reported as Res_Quant_Lat (Time) and Inc_Quant_Lat (Time). Up on evaluating the statics of this NCC accuracies, we found the architecture based on VGG16 performing

better than its counterparts. This is evident when the networks discourage over and under estimations or while they use $\lambda_{config1}$. In Table 4.12, we have marked the latent space NCC outcomes of these configurations using boldfaced letters. To this end, we conducted the evaluation of encouraging over or under estimations, i.e., using $\lambda_{config2}$, with the VGG16 based architecture. In Table 4.12 and subsequent analyses, we report the outcomes of this evaluation as VGG_Quant_OU_Lat (Time).

Table 4.12: Analysis of wins/ties/losses of the NCC accuracies that are obtained using quantile regression multi-tasking autoencoder and its counterparts

Averaging techniques	Wins	losses	ties
Arithmetic	1	83	0
DBA	0	82	2
DTAN	8	74	2
Inc_ Quant_ Lat	**5**	**74**	**5**
Inc_ Quant_ Time	0	83	1
MT_ ENC_ Lat	4	78	2
MT_ ENC_ Time	1	82	1
Res_ Quant_ Lat	**15**	**66**	**3**
Res_ Qunat_ Time	0	82	2
SDBA	8	76	0
VGG_ OU_ Qunat_ Lat	19	61	4
VGG_ OU_ Qunat_ Time	2	80	2
VGG_ Quant_ Lat	**13**	**65**	**6**
VGG_ Quant_ Time	0	82	2

According to Table 4.12, in the latent space, the VGG16 and ResNet based multi-tasking regression autoencoders performed better than their counterparts: DTAN, DBA, SDBA and the basic multi-tasking autoencoder. To further validate this observation, we next evaluate the statistics of the NCC accuracies using the box-whiskers plot shown in Figure 4.22. Moreover, the statistical parameters of the plot are shown in Table 4.13. According to Table 4.13, when we discourage over or under estimations, the reduced VGG16 obtained a latent space median accuracy of 78.54%, whereas 50% of its NCC accuracy were within the ranges of 66.82% to 91.31%. In this aspect, the ResNet and Inception setups obtained median accuracy of 75% & 71.10%. Moreover, their 50% of the NCC accuracies were respectively in between 62.18% - 90.80% and 64.45% - 92.05%. These results are in line with our observations of Table 4.12. In addition to this, in the latent space, the reduced VGG16 outperformed the state of the art (DTAN) significantly. In this regard, DTAN obtained a median accuracy of 72.94%, whereas 50% of its classification accuracy was between 58.55% and 85.45%.

One interesting observation from Table 4.13 is that the VGG16 architecture obtained a time domain statistics that is very close to DBA while it discourages over and under estimations (VGG_Quant_Time). In reality, this is very encouraging given estimates generated with the multi-tasking VGG16 architecture have no prior knowledge of DTW space. This is because we are using DTW distance to perform the time domain classification, we are transforming the estimates of DBA and SDBA to their

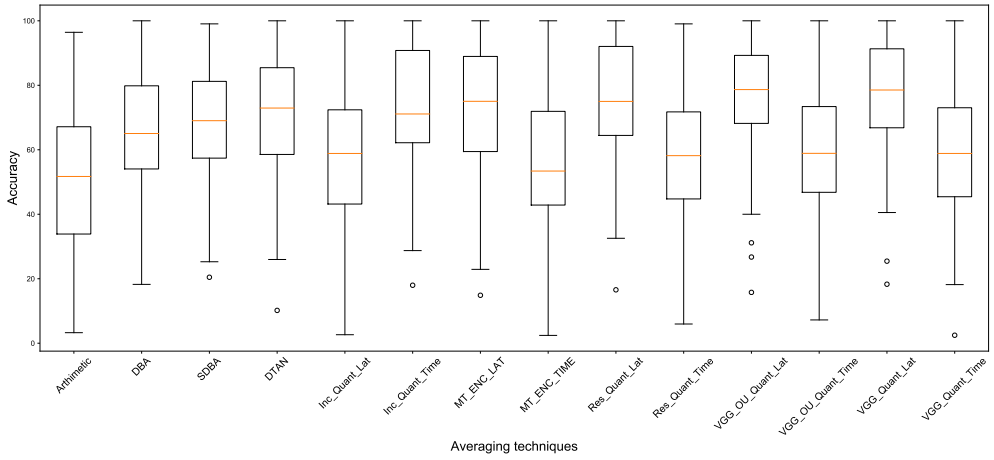


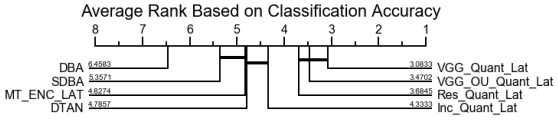
Figure 4.22: Box-whisker plot comparison of the NCC accuracies obtained with the estimates of quantile regression multi-tasking autoencoders and their counterparts. These comparison are based on 84 UCR archive datasets.

Table 4.13: Summary of the statistics for the box-whisker plot shown in Figure 4.22

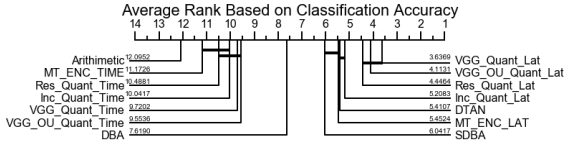
Technique	L_Q (25%)	U_Q (75%)	Lower Whisker	Upper Whisker	Median
Arithmetic	33.87	67.14	3.27	96.43	51.72
DBA	54.05	79.84	18.25	100	65.04
SDBA	57.41	81.22	25.27	99.05	69.02
DTAN	58.55	85.45	25.97	100	72.94
Inc_ Quant_ Lat	**62.18**	**90.80**	**28.73**	**100**	**71.10**
Inc_Quant_Time	43.17	72.39	2.64	100	58.86
MT_ENC_LAT	59.44	88.95	22.91	100	75.03
MT_ENC_TIME	42.85	71.92	2.43	100	53.40
Res_ Quant_ Lat	**64.45**	**92.05**	**32.54**	**100**	**75.00**
Res_Quant_Time	44.75	71.28	5.96	99.05	58.17
VGG_ OU _ Quant_ Lat	**68.18**	**89.27**	**40.00**	**100**	**78.67**
VGG_OU_Quant_Time	46.80	73.40	7.23	100	58.86
VGG _ Quant_ Lat	**66.82**	**91.31**	**40.53**	**100**	**78.54**
VGG_Quant_Time	45.43	73.03	18.18	100	58.86

registered space before the NCC. To further validate this observation, we analyze the NCC statistics using hypothesis tests. In this regard, Figures 4.23 (a) shows that the quantile regression setups that were based on the ResNet and VGG16 architectures were found to be statistically indifferent, i.e., when trained using $\lambda_{config1}$. However, despite their equivalence in the post-hypothesis test, the VGG16 obtained a better Friedman average rank. On the contrary, while encouraging over and under estimations, the VGG16 obtained better performance compared to the reduced ResNet and Inception architectures in a statistically different manner. Moreover, according to Figure 4.23 (b), encouraging over and under estimations with the VGG16 (VGG_Quant_OU_Time) appears to be giving a better time domain estimates. This performance is closely followed by the VGG16 architecture while discoraging over and under estimations (VGG_Quant_Time). Moreover, in the post-hypothesis

test, VGG_Quant_OU_Time is equivalent to the ResNet's time domain estimates (Res_Quant_Time). However, even though the quantile regression multi-tasking autoencoder is performing significantly better than its basic counterpart, it is still performing well below DBA, SDBA and DTAN in its time domain estimations. However, in this case, moving the median reconstruction line with quantile



(a) Evaluation based on NCC accuracies obtained in the registered spaces of the averaging techniques

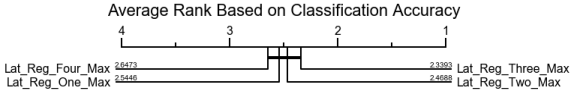


(b) Evaluation based on NCC accuracies obtained in registered and unregistered spaces of the averaging techniques

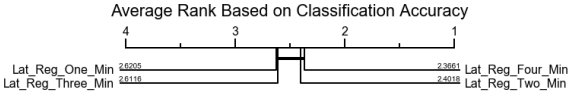
Figure 4.23: Hypothesis test based on the NCC accuracies that are obtained with the estimates of multi-tasking quantile regression autoencoders and their counterparts

regression has helped us to significantly narrow down the performance gap between the time domain estimates of the multi-tasking autoencoder and DBA. Furthermore, it should also get noted that unlike the experiments conducted for DBA, SDBA and DTAN: we only evaluated our quantile regression multi-tasking network using the outcomes of single trials. In reality, due to random weight initialization, we cannot expect single trials to capture outlier performances such as maximum NCC accuracy. On the contrary, single trials will most likely capture the median or average performance (accuracy). With this understanding, we re-trained the quantile regression multi-tasking autoencoder based on the VGG16 architecture for an additional 24 repeated trials. These repeated trials get conducted using the λ pair values given in $\lambda_{config1}$. We use this training, to further access the network using the mean, median, minimum and maximum accuracies of the 25 repeated trials.

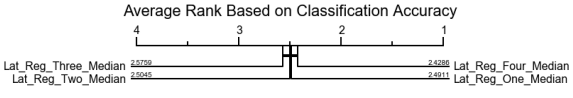
However, before directly proceeding with the hypothesis evaluation, we will first assess which of the λ pair gives better performance. In this regard, we first group the classification results of the 25 trials according to their λ values, i.e., $\lambda_{config1} = [(0.15,\ 0.85)\ , (0.25,\ 0.75),\ (0.35,\ 0.65),\ (0.5,\ 0.5)]$. We then perform two sets of hypothesis evaluations. First, we evaluate the latent space NCC accuracies of the different λ values. Following this evaluation, we conduct the same performance comparison of each λ value using the NCC accuracies of the time domain estimates. Figure 4.24 demonstrates the performance evaluation of the λ values using the latent space NCC accuracies. In the figure, we have numbered each λ pair values in $\lambda_{config1}$ according to their order of appearance. For instance, Lat_Reg_One_xxx corresponds to the λ pair $\{0.15, 0.85\}$. According to Figure 4.24 (a), the third λ pair



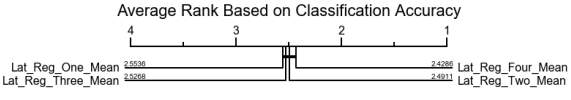
(a) Evaluation based on maximum NCC accuracies



(b) Evaluation based on minimum NCC accuracies



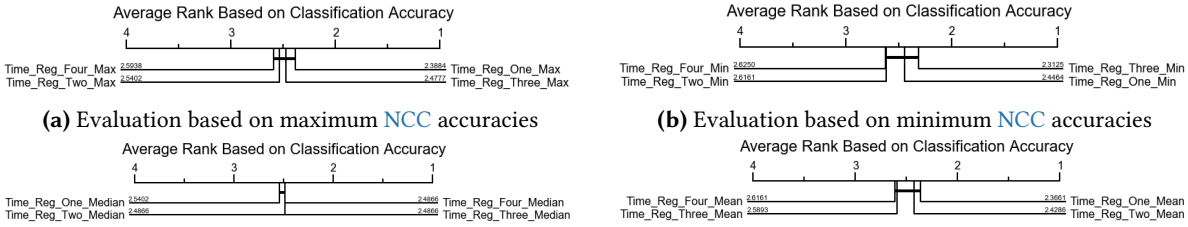
(c) Evaluation based on median NCC accuracies



(d) Evaluation based on mean NCC accuracies

Figure 4.24: Performance evaluation of quantile regression λ values based on latent space NCC accuracies

setup ((0.35, 0.65)) obtained most of the highest latent space classification accuracies. However, in the post-hypothesis test, it is considered to be statistically indifferent to the fourth λ pair value ((0.5,0.5)). On the other hand, the fourth λ pair setups obtained the best mean, median and minimum accuracies. However, in these cases, it is statistically indifferent to the first λ pairs in the minimum and mean accuracies. However, on median accuracies, it is statistically indistinguishable from the third λ pair values. Before we make the final conclusive remarks, we first assess which of the λ pair values give better performance using the time domain NCC accuracies. In this regard, in Figure 4.25 (a) & (d), the



(a) Evaluation based on maximum NCC accuracies

(b) Evaluation based on minimum NCC accuracies

(c) Evaluation based on median NCC accuracies

(d) Evaluation based on mean NCC accuracies

Figure 4.25: Performance evaluation of quantile regression λ values based on time domain NCC accuraices

first λ pairs ((0.15, 0.85)) obtained the best maximum and mean NCC accuracies. Moreover, in both cases, the post-hypothesis test reveals that they are statistically indifferent to the fourth λ pair setup. On the contrary, in the context of the minimum and median accuracies, the third and fourth λ pairs obtained the best performance. However, they are respectively statistically indifferent to the fourth and first λ pair.

In general, when we summarize the latent and time domain classification accuracies, we observed that the third λ pair ((0.35, 0.65)) obtained better performance on both time domain and latent space NCC accuracies. This is in line with our initial argument that relaxing the time domain reconstruction criteria will have a positive implication on the separability of the latent space features. This is because, in such cases, the classifier will have more influence on the encoder. In addition to this latent space implication, the third λ pair penalizes over and under estimation by only 65%. Thus, it pulls the median reconstruction line up or downwards depending on either over or under estimations are the dominant reconstructions. This, in turn, helps the decoder to compensate for reconstruction errors and the remaining effects of temporal distortion. This is expected to highly affect the peaks and troughs of the projected latent means. In addition to this observation, we also noted that the fourth λ pair ((0.5, 0.5)) has the highest minimum latent space classification accuracies, i.e., in the worst-case scenario. In reality, this λ pair configuration leaves the highest room for the categorical cross entropy and WGSS losses in (4.7). This further validated our previous argument that quantile regression can leave room for the classifier to have more say on the overall multi-tasking setup. However, in the time domain, the fourth λ pair value has the lowest worst-case classification accuracy. This is because this λ configuration behaves as a MSE or MAE reconstruction error since it penalizes both over and under estimation equally. This, in turn, supports our argument that a median reconstruction line is not suitable for optimal re-projection of the latent means. In conclusion, we suggest the utilization of $\lambda = [(0.25,\ 0.75), (0.35,\ 0.65)]$ for a better time and latent space performances. This is because

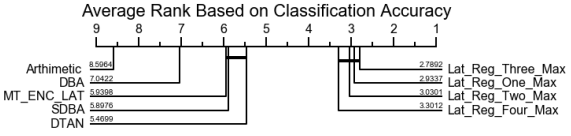
they leave room for the classifier to influence the encoder while avoiding a median reconstruction line.

With these observations in mind, we will evaluate the stability of our proposed approach. One indicating factor in this regard could be the standard deviation of the NCC accuracies. In reality, given the random nature of neural networks, we can assume the NCC accuracies to be random variables. Thus, by observing the standard deviation of the classification accuracies, we can make conclusive remarks about the reproducibility of our experimental evaluations. With this in mind, we have computed the average standard deviation of the 25 repeated trials for each λ pair value. According to Table 4.14, we can observe that the maximum latent space standard deviation is 3.246% (0.0326). On the other hand, the maximum time domain standard deviation is 4.673% (0.04673). In other words, if we, for instance, assume our mean classification accuracy is 60%, then in the worst case, latent space and time domain classification accuracies within one standard deviation would be between 58.04%-61.96% and 57.19%-62.80%. In reality, given the random nature of neural networks, we can consider this to be relatively stable.

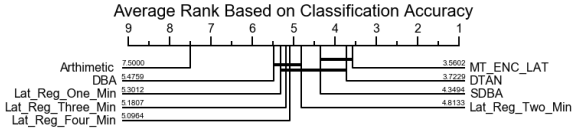
Table 4.14: The average standard deviations of the NCC accuracies obtained by different λ pair

λ pairs	Latent Space $\pm\sigma$ in %	Time Domain $\pm\sigma$ in %
(0.15, 0.85)	2.757	3.903
(0.25, 0.75)	3.246	4.257
(0.35, 0.65)	2.946	4.234
(0.5, 0.5)	3.220	4.673

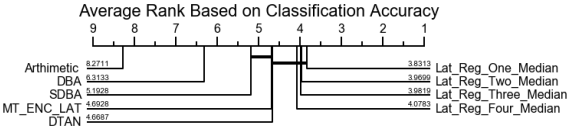
However, given the presence of random initialization in the optimization setup, the probability of a single trial capturing the maximum accuracy is relatively small. To this end, the following analysis will focus on the re-evaluation of the hypothesis tests using the outcomes of the 25 repeated trials and their mean, median, minimum, and maximum classification accuracies. In this aspect, we first re-evaluated the latent space classification accuracies on 84 datasets. We will then continue our re-evaluate of the hypothesis tests using 114 datasets. According to Figures 4.26 (a), 4.26 (c) and 4.26 (d), the quantile regression network outperformed all predecessor techniques at least by one of its λ pair setups. On the contrary, in the worst case (with its minimum classification accuracies), the performances of all



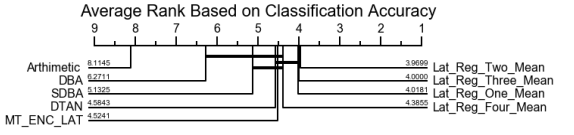
(a) Evaluation based on maximum NCC accuracies



(b) Evaluation based on minimum NCC accuracies



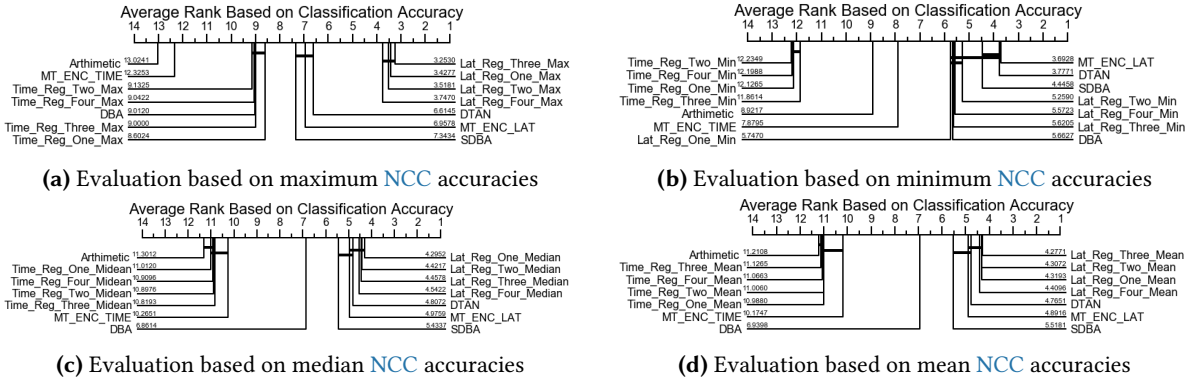
(c) Evaluation based on median NCC accuracies



(d) Evaluation based on mean NCC accuracies

Figure 4.26: Hypothesis re-evaluation for the average estimates with multi-tasking quantile regression autoencoders and their counterparts. The re-evaluation is performed using 84 UCR archive datasets and latent space NCC accuracies.

λ pair setups are bellow DBA and DTAN in their Friedman rank. However, in the post hypothesis test, the performance of the first λ pair setup ($\lambda = (0.15,\ 0.85)$) is statistically indifferent to DTAN. Generally speaking, from our previous evaluations, we expect the latent space classification to be better than the registered space performance of the alternative averaging techniques. With this in mind, when we proceed to the time domain NCC accuracies, we can see that the repeated trials have better captured the outlier performances of the quantile regression multi-tasking autoencoder. To this end, we are now able to show that the time domain estimates of the multi-tasking quantile regression network could outperform DBA's estimates, i.e., as shown in Figure 4.27 (a).



(a) Evaluation based on maximum NCC accuracies

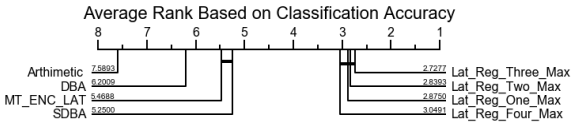
(b) Evaluation based on minimum NCC accuracies

(c) Evaluation based on median NCC accuracies

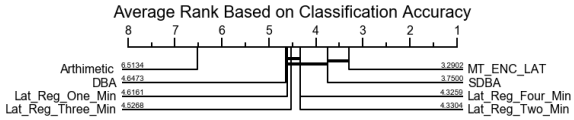
(d) Evaluation based on mean NCC accuracies

Figure 4.27: Hypothesis re-evaluation for the average estimates with multi-tasking quantile regression autoencoders and their counterparts. The re-evaluation is performed using 84 UCR archive datasets and time domain NCC accuracies.

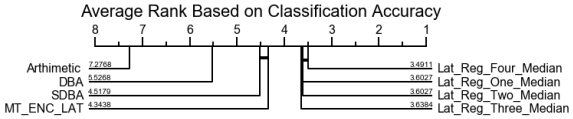
In reality, we find this to be quite encouraging. This is because, the NCC conducted with DBA's estimates is performed in DTW space which favors the estimates of DBA. In practice, the comparisons performed in a neutral space are the comparisons of the multi-tasking autoencoder (MT_ENC_TIME), quantile regression (Lat_Reg_xxx_Time) and Arithmetic estimates. This is because the NCC conducted using these techniques is in a DTW space in which they have no prior knowledge. Additionally, even if the time domain mean and median classification accuracies of VGG_Quant_Time are well below DBA, it is still interesting to see that it is still performing equivalent to the best performances of the multi-tasking autoencoder (MT_ENC_TIME). This shows that the multi-tasking quantile regression setup is a generalization of the basic-multi tasking autoencoder. In other words, as the λ gets close to one, for instance, $\lambda = [0.15, 0.85]$, the multi-tasking quantile regression becomes a relatively less outlier-sensitive basic multi-tasking autoencoder. This observation is further validated in Figure 4.27 (c) and 4.27 (d), where *MT_ENC_TIME* is found to be statistically indifferent to the multi-tasking quantile regression network that is configured with $\lambda = [(0.15, 0.85)]$. With this said, we will finalize the hypothesis re-evaluations by making a final remark on the hypothesis tests performed using 114 UCR datasets. As we stated earlier, this comparison excludes the performances of DTAN since we could not find either a standardized implementation or the evaluations for the additional 30 datasets. In general, Figure 4.28 shows that in the latent space, the performance of the multi-tasking regression network is more or less similar to the performance shown in Figure 4.26. Moreover, we still are able to outperform DBA in the maximum time domain classification accuracies. This further strengthens the observation made on the 84 UCR archive datasets where we also outperformed DBA.



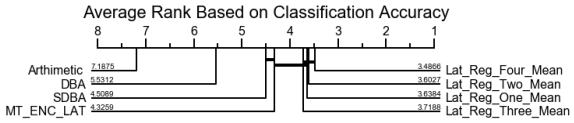
(a) Evaluation based on latent space maximum NCC accuracies



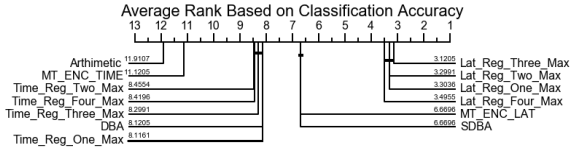
(b) Evaluation based on latent space minimum NCC accuracies



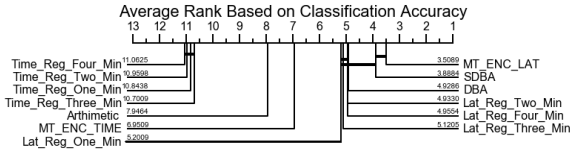
(c) Evaluation based on latent space median NCC accuracies



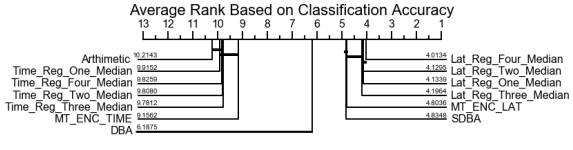
(d) Evaluation based on latent space mean NCC accuracies



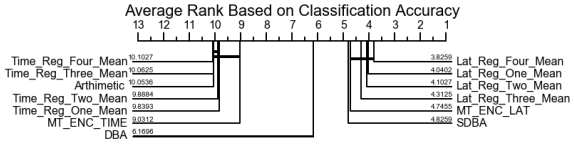
(e) Evaluation based on time domain maximum NCC accuracies



(f) Evaluation based on time domain minimum NCC accuracies



(g) Evaluation based on time domain median NCC accuracies

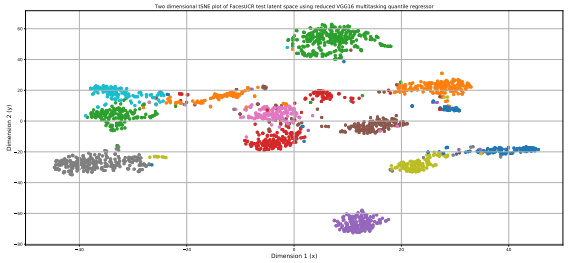


(h) Evaluation based on time domain mean NCC accuracies

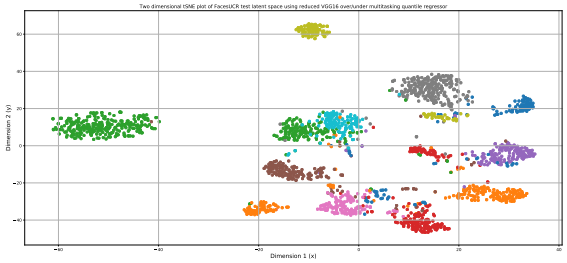
Figure 4.28: Hypothesis re-evaluation for the average estimates with multi-tasking quantile regression autoencoders and their counterparts. The re-evaluation is performed using 114 UCR archive datasets, latent space and time domain NCC accuracies.

We finally place our focus on why the VGG16 based architecture is performing better than its ResNet and Inception counterpart. This is because, in practice, we expect the Inception and ResNet to perform better than a VGG16 setup [58], [60]. As a first step to this analysis, we re-consider the t-SNE projection of the *FaceUCR* datasets shown in Figure 4.29. In the figure, we projected the *FaceUCR* test datasets in to the latent space using the VGG16, ResNet and Inception multi-tasking quantile regression networks. From the projections, we can observe that the projections of the VGG16 architectures are relatively dense. We identified two possible reasons behind this variation. First, we have not adopted the full layer arrangements of the Inception and ResNet architectures. In other words, our proposed Inception and ResNet networks are relatively shallower than the original proposals. To this end, the skip connections (memory links) evident in the Inception and ResNet architectures will inject the effects of temporal distortion into the latent space rather than serving as a way of sustaining uniform information propagation. Additionally, our reduced Inception and ResNet architectures are not purely a classification of neural networks, i.e., contrary to their original counterparts. To this end, we have no logical ground to expect the Inception and ResNet multi-tasking architectures to perform better than their VGG16 counterparts.

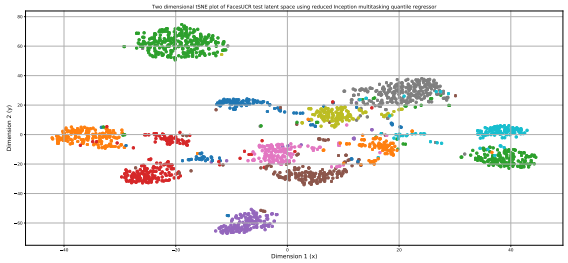
With these understandings in mind, we will conclude this section's discussion by giving the plots for the time domain estimations of the *ECG200* and *ECGFiveDays* datasets. In this aspect, in Fig-



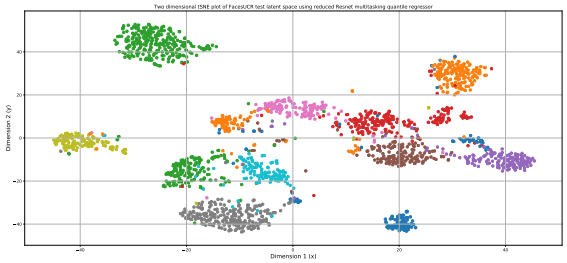
(a) *FacesUCR* in the latent space of VGG_Quant_Lat



(b) *FacesUCR* in the latent space of VGG_OU_Quant_Lat



(c) *FacesUCR* in the latent space of Inc_Quant_Lat



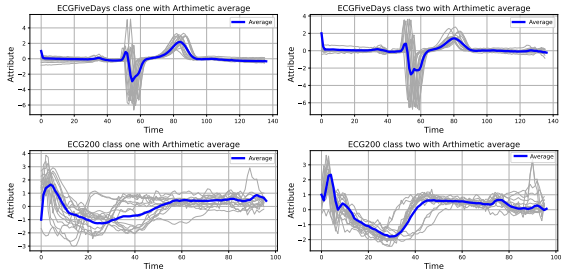
(d) *FacesUCR* in the latent space of Res_Quant_Lat

Figure 4.29: t-SNE projections for the UCR archive's *FacesUCR* test datasets. These projections are based on the latent spaces embedding obtained with the proposed quantile regression multi-tasking autoencoders

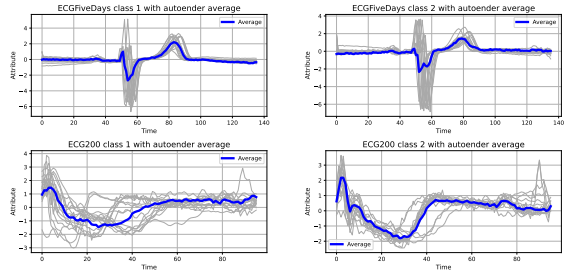
ure 4.30 (g), we can see that shifting the median reconstruction line has significantly improved the time domain re-projection. In the figure, the positive and negative peaks of the *ECGFiveDays* datasets gets estimated in a manner that is similar to the estimates of DBA and SDBA. However, unlike DBA, the multi-tasking quantile regression network is free of shape distortion that arise due to pathological association. This, in turn, has helped most of the quantile regression setups to outperform DBA in the time domain NCC. In this regard, the quantile regression multi-tasking autoencoder obtained a 76.66% classification accuracy for the *ECEGFiveDays*. On the contrary, DBA and SDBA obtained a 65.85% and 67.02% NCC classification accuracies. In conclusion, we have summarized the classification accuracies for the *ECG200* and *ECGFiveDyas* datasets in Table 4.15.

Table 4.15: NCC accuracies for the UCR archive's *ECG*200 and *ECGFiveDays* datasets.

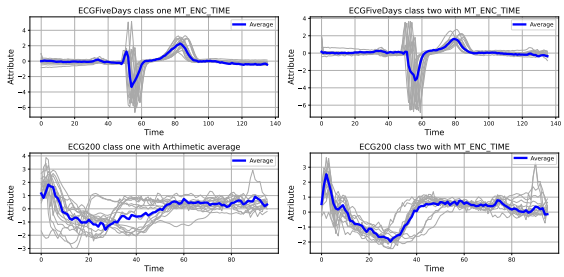
Averaging Techniques	*ECG*200 accuracy in %	*ECGFiveDays* accuracy in %
Arithmetic	67	52.96
SDBA	**73**	67.02
DBA	72	65.85
MT_ENC_TIME	72	58.65
VGG_Quant_Time	**73**	59.69
VGG_OU_Quant_Time	70	**76.66**
Res_Quant_Time	70	68.06
Inc_Quant_Time	68	64.58



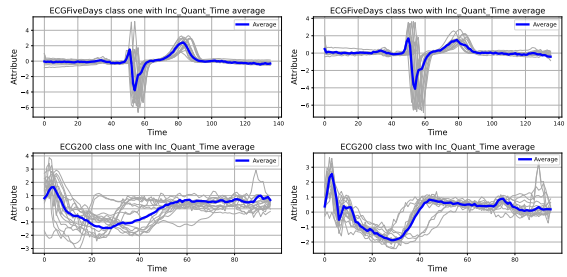
(a) Arithmetic means



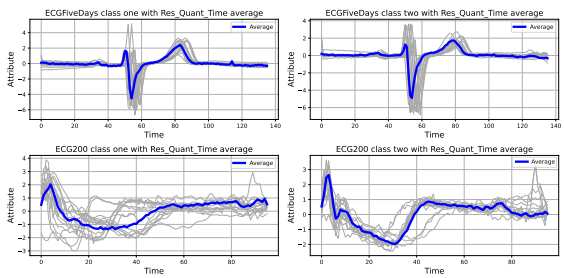
(b) Aautoencoder's estimates



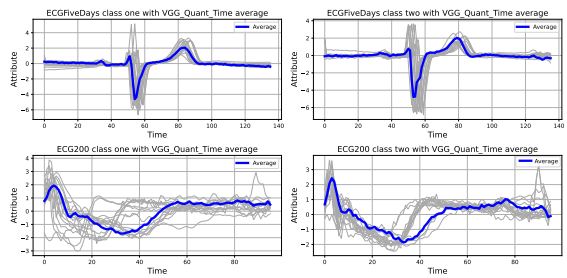
(c) Multi-tasking autoencoder's estimates



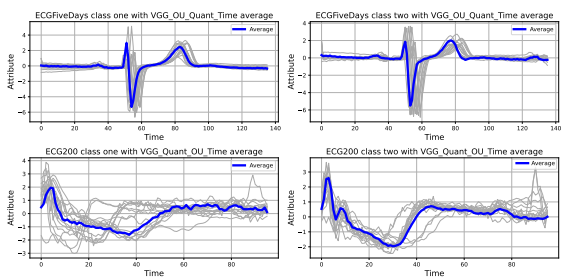
(d) Inc_Quant_Time's estimates



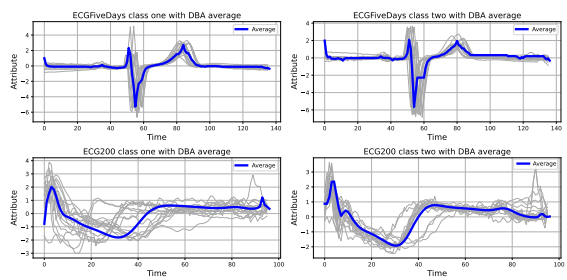
(e) Res_Quant_Time's estimates



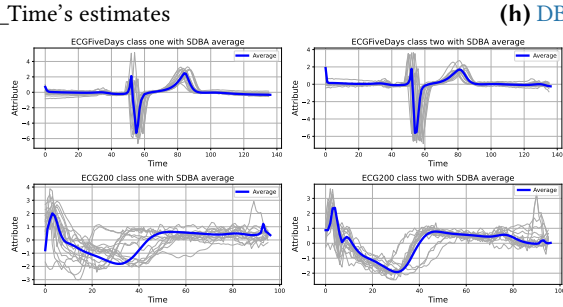
(f) VGG_Quant_Time's estimates



(g) VGG_OU_Quant_Time's estimates



(h) DBA's estimates



(i) SDBA's estimates

Figure 4.30: Averages that are estimated for the UCR archive's *ECG200* and *ECGFiveDays* datasets

4.4 Extended Evaluation of the Multi-tasking Quantile Regression Network

In the evaluation of the quantile regression network, we intensively assessed the network based on the VGG16 architecture. To make our assessment complete, in this section, we perform a similar assessment of the remaining architectures. Moreover, we also assess the implication of encouraging over and under estimations across different architectural setups. Finally, we also use these evaluations to compare performances of the basic multi-tasking and quantile regression multi-tasking autoencoders. In reality, in the previous assessment, we used the architecture shown in Figure 4.1 for the basic multitasking setup. However, for the quantile regression networks, we have made minor and major modifications that give it a better edge. Thus, in this section, we remove this bias by basing the two multi-tasking setups on similar network architectures. To meet this objective, we conduct the 25 repeated trial experiments for the remaining network configurations, i.e., Resnet_Quant_Lat (Time), Inc_Quant_Lat (Time), using the multi-tasking setups given in Figures 4.7 and 4.8. However, unlike the previous evaluation, we Incorporated the last transposed *Convolutioal* layer at the decoder. To this end, in this extended evaluation, we also reassess the performance of the VGG16 based multi-tasking quantile regression network with the new minor adjustment.

4.4.1 Experimental Setup

In the extended evaluations, we train the non-variational multi-tasking quantile regression networks using the previously proposed two λ pair setups: $\lambda_{config1} = [(0.15,\ 0.85),\ (0.25,\ 0.75),\ (0.35,\ 0.65),\ (0.5,\ 0.5)]$ and $\lambda_{config2} = [(0.85,\ 0.85),\ (0.75,\ 0.75),\ (0.65,\ 0.65),\ (0.15,\ 0.15),\ (0.25,\ 0.25),\ (0.35,\ 0.35)]$. However, unlike our previous evaluation of the quantile multi-tasking regression networks, we fully base our evaluation using the architectural configurations proposed for the multi-tasking: modified reduced VGG16, reduced Inception and reduced ResNet networks shown in Figures 4.6, 4.7 and 4.8. Despite these changes, we will force all versions of the multi-tasking quantile regression networks to optimize for the losses given in (4.7), (4.9) and (4.8). We train the proposed architectures for 1500 epochs with zero L2 regularization. Furthermore, in all training tasks, we have used a 10^{-4} learning rate and an 80/20 train and validation split. Moreover, to update the gradients, we have used the Adam optimizer configured to update the network after $\frac{N}{4}$ mini-batches, where N is the number of samples in the training set.

4.4.2 Experimental Evaluations

We have divided the experimental evaluation into two categories. First, we present the assessments that are correspond to discouraging over and under estimations by a factor less than one, i.e., when the networks get trained with $\lambda_{config1} = [(0.15,\ 0.85),\ (0.25,\ 0.75),\ (0.35,\ 0.65),\ (0.5,\ 0.5)]$. On the contrary, in the following subsection, we assess the performances of the networks while they encourage over and under estimations or when the networks are trained with $\lambda_{config2} = [(0.85,\ 0.85),\ (0.75,\ 0.75),\ (0.65,\ 0.65),\ (0.15,\ 0.15),\ (0.25,\ 0.25),\ (0.35,\ 0.35)]$. In our assessment, we

identify these different λ pair configurations using a keyword Regx, where $x = \{0,\ 1,\ 2,\ 3,\ \ldots, 4(6)\}$. With this said, we next proceed to present the first evaluation.

4.4.2.1 Extended Assessment of the Impact of Network Architectures in Multi-tasking Quantile Regression Autoencoders

We start our extended assessment of the multi-tasking quantile regression network by observing its performance in the context of NCC wins, ties, and losses. In this regard, Table 4.16 summarizes wins/ties/losses associated with each proposed architecture. In the table, we identified the outcomes of the VGG16, Inception, and ResNet based quantile regression multi-tasking autoencoders as QMMT_VGG_Regx_Lat (TD), QMMT_Inc_Regx_Lat (TD), and QMMT_ResNet_Regx_Lat (TD) where Lat (TD) corresponds to latent space and time domain. Moreover, as stated earlier, the keyword Regx is utilized to indicate the different λ configurations. With these in mind, Table 4.16 further validates our previous assessment the the VGG16 based architecture obtaining better registration in the latent space. In this regard, the VGG16 based architecture won on a total of 29 datasets. This is followed by ResNet based architecture winning on 9 datasets. However, we acknowledge that a wins/ties/losses analysis could easily get biased by the slightest difference in NCC accuracies. To this end, we assess the NCC accuracies using a box-whisker plot. In this regard, Table 4.17 shows the statistics of the box-whisker plot given in Figure 4.31.

Table 4.16: Comparison of wins/ties/losses obtained with the extended evaluations of non variational quantile multi-tasking autoencoders and their counterparts. These comparisons are obtained using the best outcomes of NCC experiments that were conducted on 75 UCR archive datasets using 25 repeated training trials and four λ pair ($\lambda = [(0.15,\ 0.85),\ (0.25,\ 0.75),\ (0.35,\ 0.65),\ (0.5,\ 0.5)]$) that discourage over and under estimations by a factor less than one.

Techniques	$L2$ Reg. (x)	Wins	Ties	Losses
Arithmetic	-	0	0	75
DBA		0	74	1
DTAN		2	2	71
SDBA		6	0	69
QMMT_VGG_Regx_Lat	{0, 1, 2, 3}	{8, 7, 7, 7}	{10, 11, 11, 9}	{57, 57, 57, 59}
QMMT_VGG_Regx_TD		{1, 0, 0, 0}	{2, 2, 2, 2}	{72, 73, 73, 73}
QMT_Inc_Regx_Lat		{2, 2, 1, 0}	{8, 4, 8, 4}	{65, 69, 66, 71}
QMT_Inc_Regx_TD		{0, 0, 0, 0}	{2, 2, 2, 1}	{73, 73, 73, 74}
QMT_ResNet_Regx_Lat		{3, 2, 2, 2}	{7, 9, 7, 9}	{65, 64, 66, 64}
QMMT_ResNet_Reg_TD		{1, 0, 0, 0}	{2, 2, 2, 2}	{72, 73, 73, 73}

In comparison, the latent space median accuracies of the quantile multi-tasking autoencoders are slightly better than their basic multi-tasking counterparts. In this aspect, unlike their basic counterparts, the median of the quantile multi-tasking autoencoders is above the 80% mark. Moreover, if we, for instance, consider and compare the basic and quantile multi-tasking VGG16 autoencoders, their best case time domain lower 25% quantiles are 52.15% and 53.14%. Additionally, their best case 75% quantities are 76.73% and 79.23%. In reality, this improvement is also evident in the remaining two architectures. For instance, for ResNet based basic and quantile multi-tasking autoencoders, their best time domain upper 75% quantile NCC accuracies are 76.83% and 80.00%. Moreover, their lower 25%

quantiles are 51.99% and 53.33%. Overall, we find the modifications have significantly improved the latent space registration. This, in turn, introduced improvements in the time domain projections. As a demonstration of this argument, we have presented t-SNE projections of the *FacesUCR* dataset as a visual demonstration. In this regard, Figure 4.32 shows the t-SNE projections of the latent embedding corresponding to the two versions of the multi-tasking setup.

Table 4.17: Statistical assessment of the NCC accuracies obtained with the extended evaluations of the quantile multi-tasking autoencoders. These assessments were conducted using the maximum NCC accuracies on 75 UCR archive datasets using the different averaging techniques

Techniques	Bot. whisker	Top whisker	25% Quant.	75% Quant.	Median
Arithmetic	7.46	96.43	37.206	67.73	52.96
DBA	28.94	100	54.82	78.55	65.14
DTAN	28.97	100	58.97	85.54	73.75
SDBA	32.83	99.05	58.41	80.87	69.71
QMMT_VGG_Regx_Lat x={0, 1, 2, 3}	{47.47, 48.27 46.93, 47.73}	{100,100, 100, 100 }	{71.38, 72.18 72.03, 71.88}	{92.73, 92.76 93.29, 93.24}	{81.97, 81.19, 82.13, 80.77}
QMMT_VGG_Regx_TD x={0, 1, 2, 3}	{17.55, 20.06, 18.49, 18.18 }	{100, 100, 100, 100}	{52.23, 53.34, 52.17, 52.24}	{78.73, 77.89 79.27, 77.99 }	{63.93, 64.06, 64.06, 64.81}
QMT_Inc_Regx_Lat x={0, 1, 2, 3}	{48.53, 47.63, 48.00, 48.27 }	{100, 100 100, 100 }	{69.28, 69.03, 68.89, 67.85}	{91.53, 91.47, 91.53, 91.03}	{80.00, 80.00, 80.00, 80.43}
QMMT_Inc_Regx_TD x={0, 1, 2, 3}	{20.56, 16.14, 17.55, 18.34}	{100, 100, 100, 100}	{50.68, 51.57, 51.17, 51.92}	{75.00, 75.27, 78.18, 76.78}	{63.04, 64.32, 64.75, 64.00}
QMT_ResNet_Regx_Lat x={0, 1, 2, 3}	{47.73, 46.93, 46.67, 48.27 }	{100, 100 100, 100}	{69.57, 68.77, 68.55, 69.55}	{92.94, 92.47, 92.02, 92.49}	{81.09, 80.50, 80.00, 80.77 }
QMT_ResNet_Regx_TD x={0, 1, 2, 3}	{20.81, 21.01, 21.26, 22.88}	{100, 100 100, 100}	{52.65, 53.23, 51.42, 52.89}	{78.56, 76.83, 80.00,79.00}	{65.27, 64.95, 64.22, 64.34}

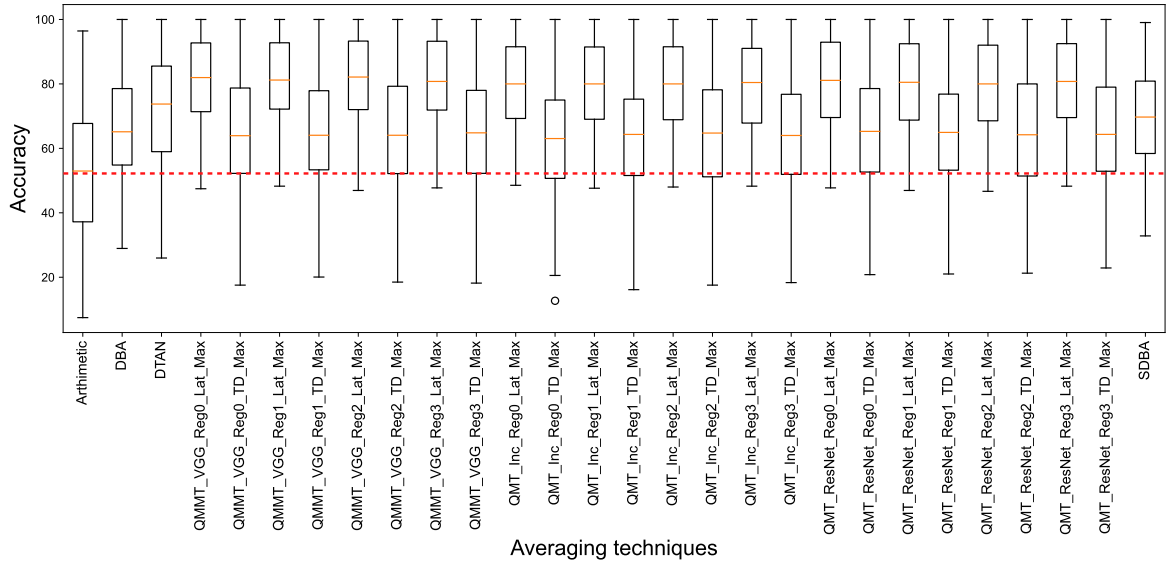
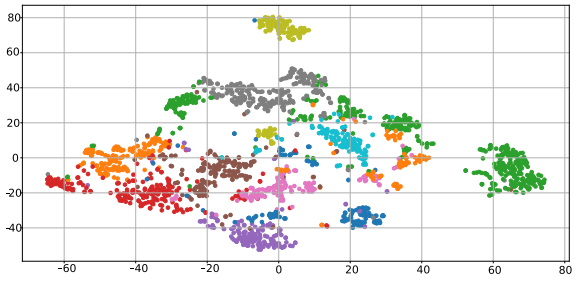
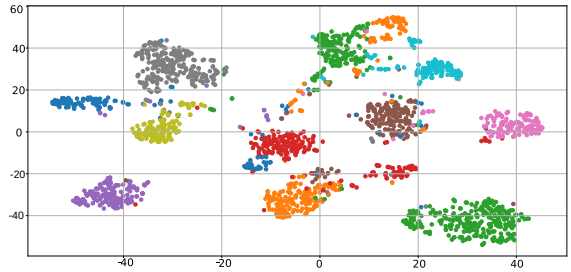


Figure 4.31: Box-whisker plot of NCC accuracies obtained with the extended evaluation of quantile regression multi-tasking autoencoders.

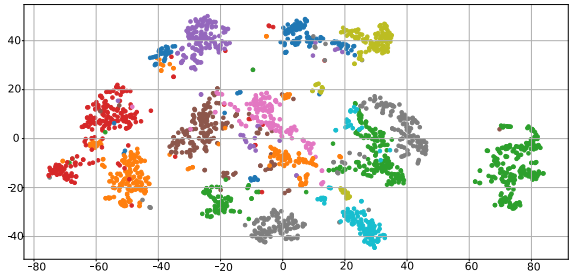
Comparatively, the embedding of the basic multi-tasking setups, i.e., shown in the left column of Figure 4.32, are less separable from their quantile multi-tasking counterparts. This further supports the argument behind the modifications made to the objective functions of the multi-tasking setups. With these observations in mind, we next assess if there are any significant changes in the performances of the multi-tasking quantile regression autoencoders, i.e., among themselves and as compared to their counterparts. In this aspect, Figure 4.33 shows the hypothesis tests based on the latent space (left column) and time domain (right column) NCC accuracies.



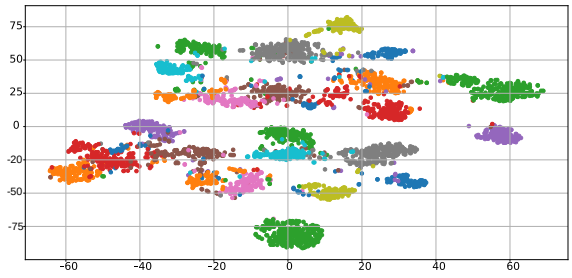
(a) *FacesUCR* in the latent space of multi-tasking modified reduced VGG16 autoencoder



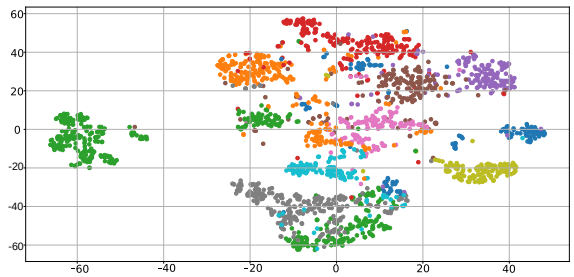
(b) *FacesUCR* in the latent space of multi-tasking QMMT_VGG_Regx_Lat



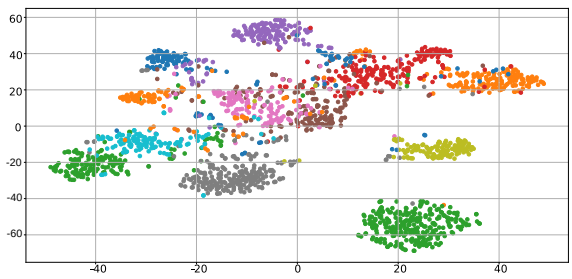
(c) *FacesUCR* in the latent space of multi-tasking reduced Inception autoencoder



(d) *FacesUCR* in the latent space of QMT_Inc_Regx_Lat



(e) *FacesUCR* in the latent space of multi-tasking reduced ResNet autoencoder

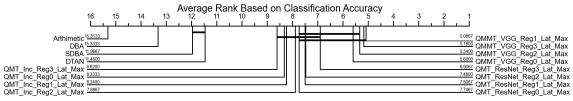


(f) *FacesUCR* in the latent space of QMT_ResNet_Regx_Lat

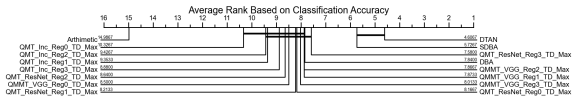
Figure 4.32: t-SNE projections for the UCR archive's *FacesUCR* test datasets in the latent spaces of multi-tasking and quantile regression multi-tasking autoencoders

According to Figure 4.33 (b), the quantile multi-tasking autoencoder that is based on the ResNet architecture is better than DBA when it is trained using the fourth λ configuration ($\lambda = [(0.5,\ 0.5)]$). In other words, the quantile multi-tasking autoencoder performs better when penalizing over and under estimations in a manner that significantly shifts the median reconstruction line. Moreover, since the given λ pair penalizes over and under estimations equally, we can safely assume the quantile regression as a relatively relaxed reconstruction loss. However, in the latent space, encouraging over and under estimations by smaller amount gave better performances, i.e., Figure Figure 4.33 (a). In

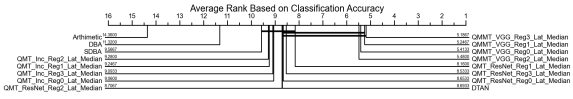
reality, since the latent space has to meet the requirements of the classifier, we can not expect the latent space to give better performances with the λ values that discourage over and under estimations. This is because with such λ values we are encouraging the network to over or under estimate an input dataset. This is contrary to the need of a classifier that needs to identify a descriptive feature that is based on input rather than its over or underestimation.



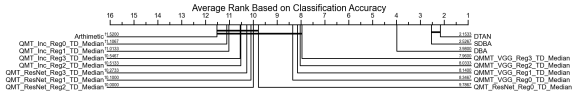
(a) Evaluation of maximum NCC accuracies obtained in the latent space of quantile regression multi-tasking autoencoders



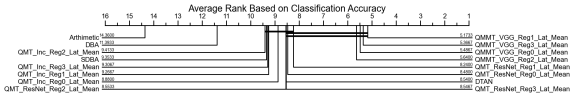
(b) Evaluation of maximum NCC accuracies obtained with the time domain estimates of quantile regression multi-tasking autoencoders



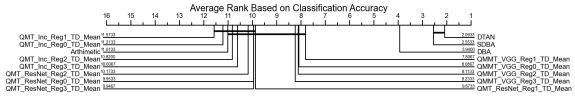
(c) Evaluation of median NCC accuracies obtained in the latent space of quantile regression multi-tasking autoencoders



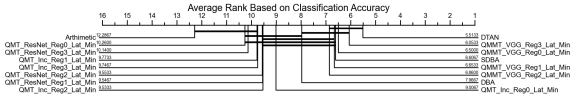
(d) Evaluation of median NCC accuracies obtained with time domain estimates of quantile regression multi-tasking autoencoders



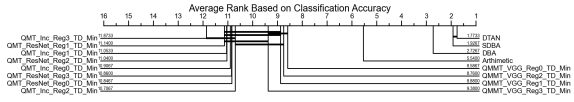
(e) Evaluation of mean NCC accuracies obtained in the latent space of quantile regression multi-tasking autoencoders



(f) Evaluation of mean NCC accuracies obtained with time domain estimates of quantile regression multi-tasking autoencoders



(g) Evaluation of minimum NCC accuracies obtained in the latent space of quantile regression multi-tasking autoencoders

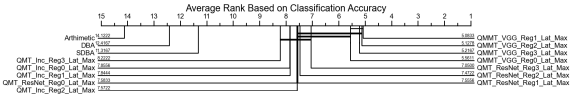


(h) Evaluation of minimum NCC accuracies obtained with time domain estimates of quantile regression multi-tasking autoencoders

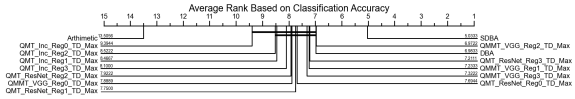
Figure 4.33: CD diagram comparisons of NCC accuracies obtained with the extended evaluation of quantile multi-tasking autoencoders. The comparison is performed using the NCC accuracies obtained from 75 UCR archive datasets.

To further asses the reported results, we have also compared the NCC accuracies corresponding to DBA, SDBA, and the quantile regression multi-tasking setups using additional 24 datasets. In this regard, Figure 4.34 shows the comparisons based on 89 UCR archive datasets. Unlike their multi-tasking counterparts, the quantile regression multi-tasking autoencoders can sustain the time domain performance. In this aspect, Figure 4.34 (b) shows that the time domain estimates generated by the quantile multi-tasking autoencoder that is based on the ResNet is still beating DBA's performance. This performance is also evident when the network gets trained with the fourth λ pair value. This further validates that the quantile-regression-based approach generates better time domain estimates than its basic multi-tasking counterpart. In this aspect, the direct comparison shown in Figure 4.35 also validates this remark. In the figure, we have compared the best time domain and latent space NCC accuracies obtained with both versions of the multi-tasking setups across the different λ configurations. In general, in the latent space, the quantile regression multi-tasking setup outperforms its basic counterparts with clear statistical demarcations. On the contrary, in the time domain, some of the estimations obtained with the basic multi-tasking autoencoders performs similarly to their

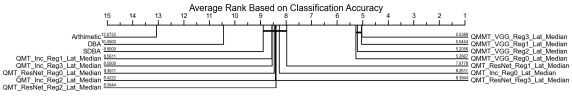
quantile regression counterparts. However, overall, we find the quantile regression multi-tasking autoencoders perform better.



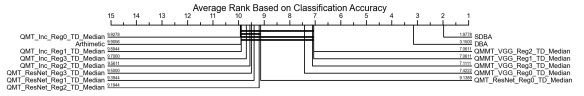
(a) Evaluation of maximum NCC accuracies obtained in the latent space of quantile regression multi-tasking autoencoders



(b) Evaluation of maximum NCC accuracies obtained with the time domain estimates of quantile regression multi-tasking autoencoders



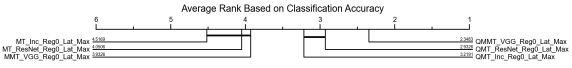
(c) Evaluation of median NCC accuracies obtained in the latent space of quantile regression multi-tasking autoencoders



(d) Evaluation of median NCC accuracies obtained with time domain estimates of quantile regression multi-tasking autoencoders

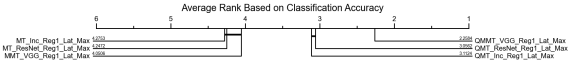
Figure 4.34: CD diagram comparisons of NCC accuracies obtained with the extended evaluation of quantile multi-tasking autoencoders. The comparison is performed using the NCC accuracies obtained from 89 UCR archive datasets.

With these observations in mind, we will next analyze the variance observed across the NCC accuracies obtained with the different network configurations. We perform this evaluation to assess the stability of the proposed architectures. Following this, we will finalize our discussion by presenting the estimates generated for UCR archive's *FacesUCR* and *ECGFiveDyas* datasets. We present the estimations to show that the introduction of the last transposed *Convolutional* layer has no significant

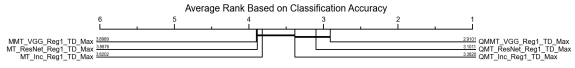


(a) Comparison of maximum NCC accuracies obtained in the latent space of basic multi-tasking and quantile regression multi-tasking autoencoders when trained with first λ pair

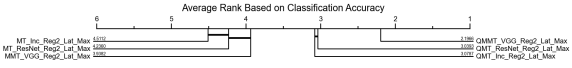
(b) Evaluation of maximum NCC accuracies obtained with the time domain estimates of basic multi-tasking and quantile regression multi-tasking autoencoders when trained with the first λ pair.



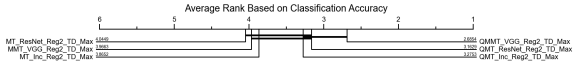
(c) Comparison of maximum NCC accuracies obtained in the latent space of basic multi-tasking and quantile regression multi-tasking autoencoders when trained with second λ pair



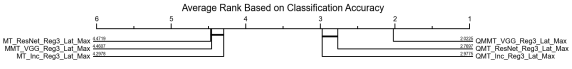
(d) Comparison of maximum NCC accuracies obtained with time domain estimates of multi-tasking and quantile regression multi-tasking autoencoders when trained with second λ pair



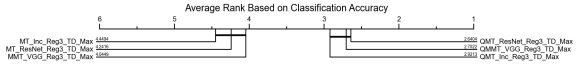
(e) Comparison of maximum NCC accuracies obtained in the latent space of multi-tasking and quantile regression multi-tasking autoencoders when trained with third λ pair values



(f) Evaluation of mean NCC accuracies obtained with time domain estimates of quantile regression multi-tasking autoencoders when trained with third λ pair values



(g) Comparison of maximum NCC accuracies obtained in the latent space of multi-tasking and quantile regression multi-tasking autoencoders when trained with λ pair values



(h) Comparison of maximum NCC accuracies obtained with time domain estimates of multi-tasking and quantile regression multi-tasking autoencoders when trained with λ pair values

Figure 4.35: Comparison of NCC accuracies obtained with multi-tasking and quantile multi-tasking autoencoders. These comparisons are performed using NCC outcomes obtained on 89 UCR archive datasets

negative implication on the quality of the time domain estimates. With this said, Table 4.18 shows that the latent space standard deviation is below the 7% mark. Moreover, the time domain standard deviation is overall below the 9% mark. In general, given the multi-tasking quantile regression networks are optimizing for a range of loss functions with different sets of requirements, we find the standard deviation is relatively close to its basic multi-tasking counterpart. To this end, we believe that the reproducibility of the experimental outcomes has a higher likelihood.

Table 4.18: The average standard deviations of the NCC accuracies obtained by different λ pair. In the table, we indicate the different λ pair configuration as $\lambda_x : x = \{0, 1, 2, 3\} = \{(0.15, 0.85), (0.25, 0.75), (0.35, 0.65), (0.5, 0.5)\}$, where QMMT_VGG, QMT_Inc, QMT_ResNet correspond to the quantile multi-tasking autoencoders based on the VGG16, Inception, and ResNet architectures.

Techniques	x in λ_x	Latent Space $\pm\sigma$ in %	Time Domain $\pm\sigma$ in %
QMMT_VGG	0	5.29	6.85
QMT_Inc		5.66	8.35
QMT_ResNet		6.14	7.74
QMMT_VGG	1	5.50	7.26
QMT_Inc		5.72	8.62
QMT_ResNet		5.80	7.44
QMMT_VGG	2	5.50	6.69
QMT_Inc		5.78	8.55
QMT_ResNet		5.99	7.49
QMMT_VGG	3	5.39	6.83
QMT_Inc		5.94	8.86
QMT_ResNet		6.21	7.50

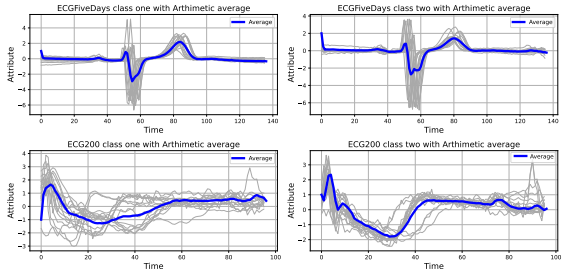
With these in mind, in Table 4.19 we have summarized the NCC accuracies that are obtained for the UCR archive's *ECG200* and *ECGFiveDays* datasets. The time domain estimates corresponding to the NCC accuracies that are summarized in the table are shown in Figures 4.36and ??. In comparison, for the *ECG200* dataset, the estimates of the quantile regression networks can obtain a NCC accuracy that is equivalent to the state-of-the-art, i.e., DTAN. Moreover, for the *ECGFiveDays* dataset, the NCC obtained with the estimates of the quantile regression autoencoder are better than alternative techniques only with the exception of DTAN.

With such observations in mind, we found the quantile multi-tasking arrangement to be the best of our proposals. However, to make this conclusive remark more complete, we next aim to assess the impact of encouraging over and under estimations. In this regard, we argued that by using over and under estimations, we could either pull up or down the median reconstruction line. In this aspect, the extended experimental evaluations of the quantile regression multi-tasking autoencoders asserted that pulling the median reconstruction line up or down has a positive implication on the quality of the time domain estimation. However, in the experiments, we have not significantly allowed over and under estimations. The maximum amount by which we encouraged over and under estimation is as high as 35%. This is because we do not consider the 50% λ pair in $\lambda_{config1}$ as a configuration that encourages over and under estimation since it penalizes both equally.

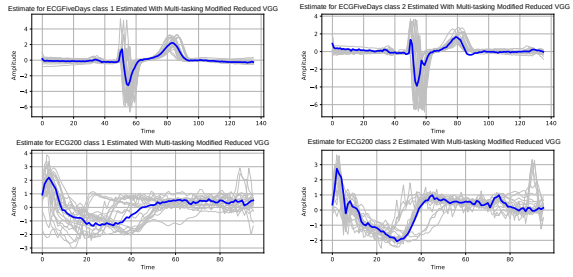
With this in mind, in the next sub section we assess two questions. First, we assess which of the two, i.e., over or under, estimations give better time domain estimates. In addition to this, we also asses by to what extent encouraging over and under estimations would give meaningful reconstructions in the context of shapes observed in the averaged set. With this said, we conclude this section by presenting the plots for the time domain estimates of the UCR archive's *ECG200* and *ECGFiveDays* datasets whose NCC accuracies are summarized in Table 4.19.

Table 4.19: NCC accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets that are obtained with multi-tasking autoencoders

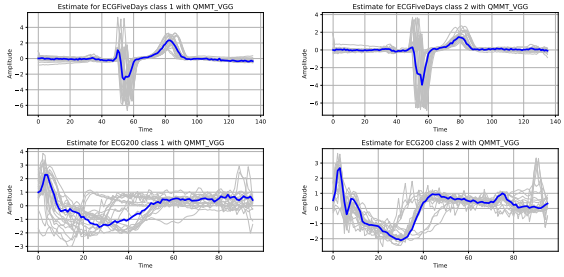
Techniques	NCC for *ECG200* in %	NCC for *ECGFiveDays* in%
Arithmetic	67	52.96
DBA	65	52.15
SDBA	73	67.02
DTAN	**79**	**97.79**
MMT_VGG_TD	77	70.27
MT_Inception_TD	78	72.36
MT_ResNet_TD	78	72.71
Var_MMT_VGG_TD	73	70.15
Var_MT_Inception_TD	77	70.49
Var_MT_ResNet_TD	77	74.33
QMMT_VGG_TD	76	**78.39**
QMT_Inception_TD	**79**	**76.54**
QMT_ResNet_TD	**79**	**76.89**



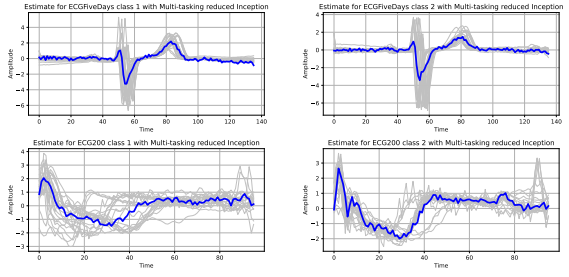
(a) Arithmetic means



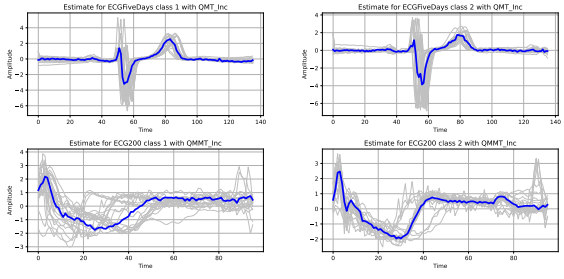
(b) MMT_VGG's estiamtes



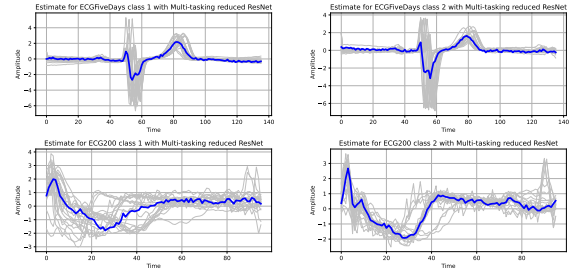
(c) QMMT_VGG's estimates



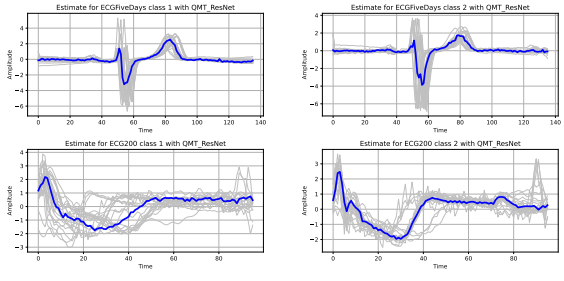
(d) MMT_Inc's estiamtes



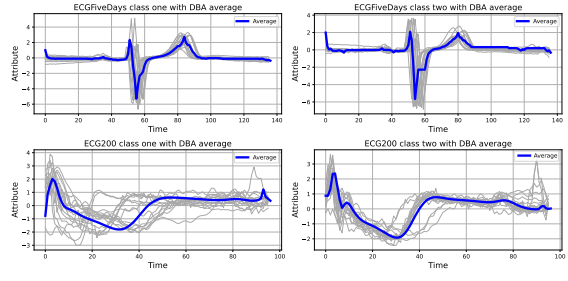
(e) QMMT_Inc's estimates



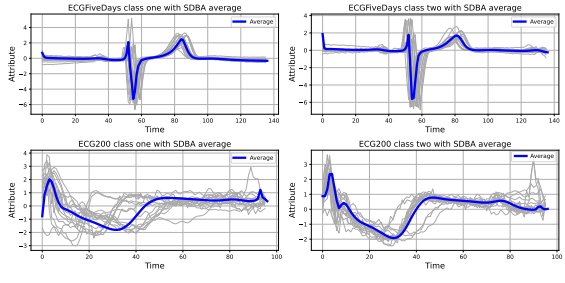
(f) MMT_ResNet's estimates



(g) QMMT_ResNet's estimates



(h) DBA's estimates



(i) SDBA's estimates

Figure 4.36: Averages estimated for the UCR archives *ECG200* and *ECGFiveDays* datasets using quantile and basic multi-tasking: modified reduced VGG16, reduced Inception, reduced ResNet, and alternative averaging techniques

4.4.3 Assessing the Impact of Encouraging Over and Under Estimations on the Quality of Time Domain Estimates

In this subsection, we have trained the quantile regression multi-tasking autoencoders using $\lambda_{config2} = [(0.85, 0.85), (0.75, 0.75), (0.65, 0.65), (0.15, 0.15), (0.25, 0.25), (0.35, 0.35)]$. As a quick reminder, the first three λ pair values encourage underestimation by more penalizing over estimation. For instance, the configuration $(0.85, 0.85)$ penalizes overestimation by 85% while penalizing underestimation by only 15%. Consequently, the last three λ pair values encourage overestimation. However, since we are now expected to train the quantile multi-tasking autoencoders on more λ pair values, with the available computational resources, we were able to perform the experiments on 72 UCR archive datasets. With these said, we will first perform wins/ties/losses using NCC accuracies obtained on 56 UCR archive datasets, i.e., including the results reported for DTAN. In this regard, based on the results shown in Table 4.20, we can safely conclude that by encouraging over or underestimations we have encouraged the classifier to have a relatively higher say on the latent embedding. This is because, in the latent space, the quantile regression multi-tasking autoencoders performed significantly better than the alternative techniques. To this end, we can assume the latent space embeddings are comparatively compact and separable. Additionally, we found the time domain estimates of the quantile regression multi-tasking autoencoders to have more ties compared to the case where they were discouraging over and under estimations. However, as we repeatedly argued, wins/ties/losses do not show how much a given technique is winning. To this end, we will next assess the statistics of the accuracies, i.e., using a box-whisker plot, for further analysis. In this aspect, Table 4.21 summarizes the statistics

Table 4.20: Statistics assessment of the NCC accuracies that are obtained with quantile multi-tasking: modified VGG16, reduced Inception, and reduced ResNet architectures that encouraged over and under estimation. These assessments were conducted using the maximum NCC accuracies that are obtained on 56 UCR archive datasets while using the different averaging techniques

Techniques	$L2$ Reg. (x)	Wins	Ties	Losses
Arithmetic	-	0	0	56
DBA		0	1	55
DTAN		1	2	53
SDBA		3	0	53
QMMT_VGG_OU_Regx_Lat	{0-5}	{2, 4, 1, 4, 8, 3}	{6, 5, 9, 9, 10, 10}	{48, 47, 46, 43, 38, 43}
QMMT_VGG_OU_Regx_TD		{0, 0, 0, 0, 0, 0}	{1, 1, 2, 2, 2, 2}	{55, 55, 54, 54, 54, 54}
QMT_Inc_OU_Regx_Lat		{0, 0, 1, 1, 1, 1}	{5, 5, 4, 3, 4, 5}	{51, 51, 51, 52, 51, 50}
QMT_Inc_OU_Regx_TD		{0, 0, 0, 0, 0, 0}	{1, 0, 2, 1, 2, 2}	{55, 56, 54, 55, 54, 54}
QMT_ResNet_OU_Regx_Lat		{0, 2, 1, 2, 0, 2}	{6, 5, 6, 5, 5, 3}	{50, 49, 49, 49, 51, 51}
QMT_ResNet_OU_Regx_TD		{0, 1, 1, 0, 1, 0}	{1, 2, 2, 1, 2, 2}	{55, 53, 53, 55, 53, 54}

for the box-whisker plot shown in Figure 4.37. Comparatively, we found a slight improvement in the statistics of the quantile regression that encouraged over and underestimations compared to its counterpart. For instance, in the latent space, the architecture based on the VGG16 got an 84.37% best case median accuracy on the 56 UCR archive datasets and while it was discouraged over and underestimation. On the contrary, while encouraging over and under estimations, it got 86.65% best median accuracy. Moreover, while discouraging over and under estimations, its best case lower

whisker, 25% quantile, and 75% quantile were: 55.26%, 72.56%, and 94.08%. On the contrary, while encouraging over and under estimation, its statistics were: 53.89%, 74.34%, and 96.48%. In addition to these improvements, in the time domain, the architecture obtained a best case median accuracy of 66.48%, i.e., while encouraging over and underestimation. On the contrary, over the same datasets, the architecture obtained a best case median NCC accuracy of 65.71%, i.e., when discouraging over and under estimations. Moreover, the time domain best case lower whisker, 25%, and 75% quantiles are 20.06%, 54.09%, and 84.47%, i.e., while it discouraged over and under estimations. On the contrary, while it encouraged over and under estimations, it obtained 23.19%, 54.92%, and 84.79% over the three statistical terms. Overall, encouraging over and under estimations across all architectures introduced slight improvements. As we argued earlier, encouraging over and under estimations leaves more room for the classifier so that it influences the type of extracted latent features. This, in turn, encourages the extraction of comparatively better separable and dense latent embeddings. With this in mind, we next assess which of the λ pairs gives better performances. In this aspect, Figure 4.38 shows the CD diagrams associated with the NCC accuracies obtained on 72 UCR archive datasets.

Table 4.21: Statistical assessment of the NCC accuracies obtained with the extended evaluations of the quantile multi-tasking autoencoders that encourage over and under estimation. These assessments were conducted using the maximum NCC accuracies that are obtained on 56 UCR archive datasets

Techniques	Bot. whisker	Top whisker	25% Quant.	75% Quant.	Median
Arithmetic	8.31	96.43	41.04	71.09	55.59
DBA	28.94	100	54.05	83.48	67.90
DTAN	33.31	100	60.79	88.29	76.69
SDBA	32.83	99.05	57.41	85.06	71.50
QMMT_VGG_OU_Regx_Lat x={0, 1, 2, 3, 4, 5}	{53.42, 49.74, 48.68, 46.18, 53.89, 46.58 }	{100,100, 100, 100, 100, 100}	{71.70, 74.34, 72.44, 74.04, 74.03, 72.64 }	{92.28, 96.45, 94.59, 94.22, 94.98, 94.89 }	{84.35, 84.54, 85.77, 86.14, 86.65,86.85 }
QMMT_VGG_OU_Regx_TD x={0, 1, 2, 3, 4, 5}	{5.02, 19.12, 20.06, 19.49, 23.13, 23.19}	{100, 100, 100, 100, 100, 100}	{44.52, 51.36 54.26, 51.83, 53.02, 54.92}	{75.06, 79.30, 82.42, 81.25, 84.48, 84.79}	{62.49, 63.70, 66.48, 64.04, 65.28, 65.50}
QMT_Inc_OU_Regx_Lat x={0, 1, 2, 3, 4, 5}	{52.24, 51.45, 53.25, 55.84, 51.29, 51.29 }	{100, 100, 100, 100, 100, 100 }	{69.70, 71.08, 70.24, 70.54, 70.11, 70.39}	{94.54, 93.62, 94.05, 94.49, 95.51, 94.38}	{80.59, 81.62, 81.25, 83.33, 82.56, 83.33}
QMMT_Inc_OU_Regx_TD x={0, 1, 2, 3, 4, 5}	{10.33, 19.29, 16.46, 22.82, 23.19, 18.34}	{100, 99.05, 100, 100, 100, 100}	{37.47, 49.34, 53.14, 48.86, 52.01, 52.38 }	{78.82, 79.66, 79.33, 80.99, 83.95, 82.98}	{61.59, 64.16, 64.69, 64.45, 64.57, 64.35}
QMT_ResNet_OU_Regx_Lat x={0, 1, 2, 3, 4, 5}	{53.25, 55.84, 53.42, 53.03, 51.97, 52.59}	{100, 100, 100, 100, 100, 100}	{70.61, 71.53, 70.93, 69.09, 70.38, 70.13}	{94.34, 93.59, 94.16, 96.04, 94.18, 94.06}	{82.72, 83.45, 83.75, 80.41, 81.30, 83.04}
QMT_ResNet_OU_Regx_TD x={0, 1, 2, 3, 4, 5}	{16.92, 20.06, 22.22, 17.60, 22.31, 22.31}	{100, 100, 100, 100, 100, 100}	{49.72, 51.69, 53.08, 44.72, 48.54, 52.04}	{80.42, 83.86, 84.46, 76.15, 81.35, 84.13}	{64.78, 63.25, 65.21, 64.18, 64.30, 65.04}

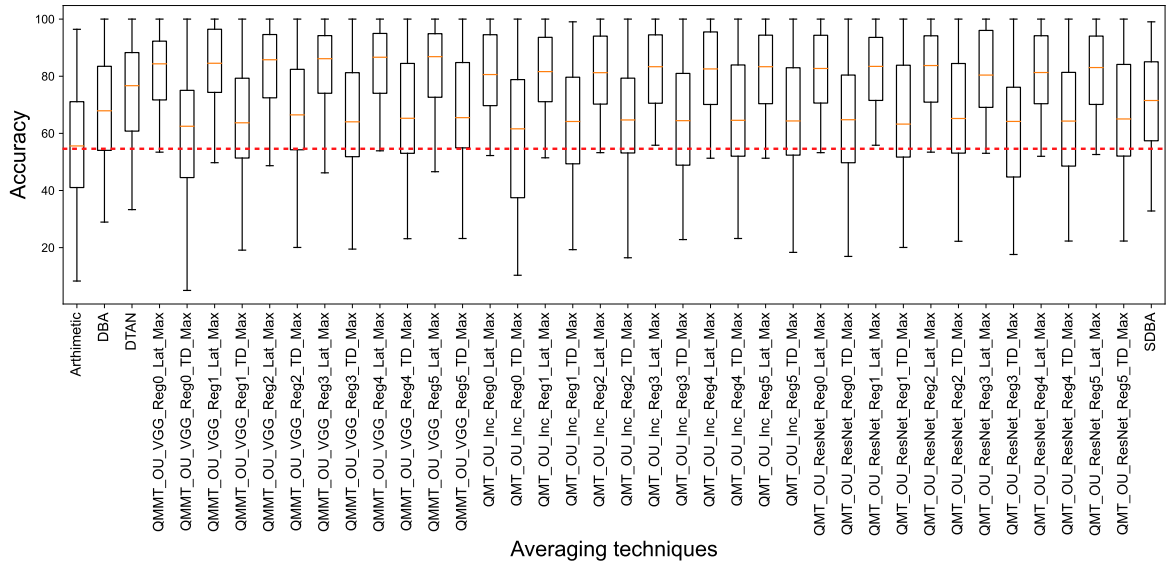
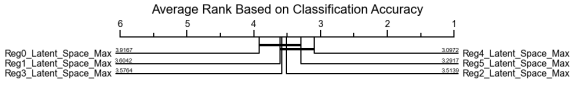
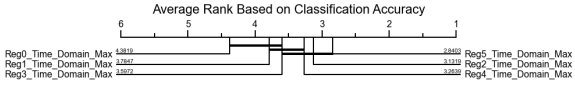


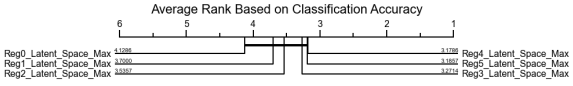
Figure 4.37: Box-whisker plot comparison of the NCC accuracies that are obtained with the estimates of quantile regression multi-tasking autoencoders while encouraging over and under estimations. These comparison are based on NCC accuracies obtained on 56 UCR archive datasets and using different averaging techniques.



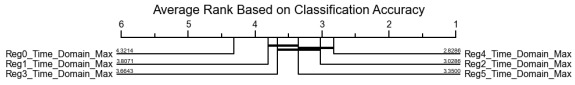
(a) Comparison of maximum latent space NCC accuracies across different λ pair values for QMMT_OU_VGG that are obtained while encouraging over and under estimations



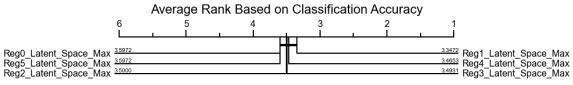
(b) Comparison of maximum time domain NCC accuracies across different λ pair values for QMMT_OU_VGG that are obtained while encouraging over and under estimations



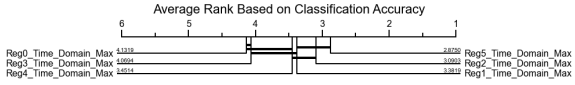
(c) Comparison of maximum latent space NCC accuracies across different λ pair values for QMT_OU_Inc that are obtained while encouraging over and under estimations



(d) Comparison of maximum time domain NCC accuracies across different λ pair values for QMT_OU_Inc that are obtained while encouraging over and under estimations



(e) Comparison of maximum latent space NCC accuracies across different λ pair values for QMT_OU_ResNet that are obtained while encouraging over and under estimations

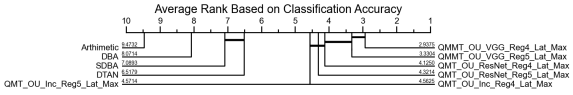


(f) Comparison of maximum latent space NCC accuracies across different λ pair values for QMT_OU_ResNet that are obtained while encouraging over and under estimations

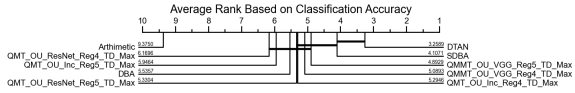
Figure 4.38: CD diagram comparisons of NCC accuracies obtained with different λ values while quantile multi tasking regression autoencoders encouraged over and under estimations. The comparison is performed using the NCC accuracies obtained on 72 UCR archive datasets.

According to Figures 4.38 (a) and 4.38 (b), the VGG16 architecture obtained better latent space and time domain NCC accuracies with the fifth (0.25, 0.25) and sixth (0.35, 0.35) λ pair values. In reality, these λ values comparatively encourage overestimations. However, the fugues also show that these λ values are respectively statistically indifferent to second and fourth λ pair values. This shows that, in the latent space, for this architecture encouraging over and underestimations have the same implication. However, in the time domain, encouraging overestimation appears to be obtaining better performances. In reality, in the latent space, the fifth λ pair value also appears to be providing better performances for the Inception architecture. However, in this case, it is statistically indifferent

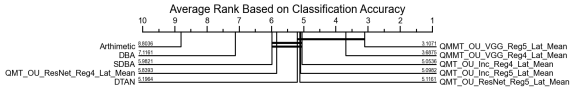
to the first λ pair value. On the contrary, in the time domain, the fifth and sixth λ values appeared statistically indifferent. This further adds to the fact that encouraging overestimation is giving better time domain performances. In general, the same fact can also be stated for the ResNet architecture. Thus, overall, we found the fifth λ (0.25, 0.25) value to give better latent and time domain estimations. If we pause at this point and think of it, our initial argument for proposing the quantile regression was the fact that the decoder of the basic multi-tasking autoencoders was bound to a median reconstruction line which we found to be behaving as arithmetic mean. Thus, it would be no surprise that an overestimation pulls up the median reconstruction line, i.e., for each time stamp, which in turn gives better performance. With this in mind, we next observe how over and under estimations perform compared to the estimates of the alternative averaging techniques. In this aspect, we first assess the performances of the estimates with the inclusion of DTAN using 56 UCR archive datasets. We then exclude DTAN and further the assessment using NCC accuracies obtained on 72 UCR archive datasets for some of which the evaluations for DTAN are missing. With these in mind, Figure 4.39 shows the performance comparisons based on the 56 UCR archive datasets, where the left and right columns correspond to the comparison made using the latent space and time domain estimates of the quantile regression multi-tasking autoencoders.



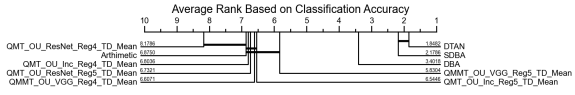
(a) Comparison of maximum latent space NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques



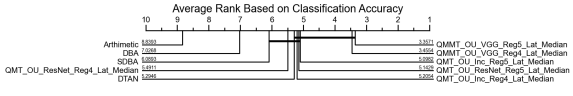
(b) Comparison of maximum time domain NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques



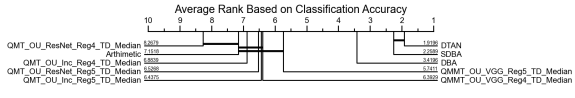
(c) Comparison of mean latent space NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques



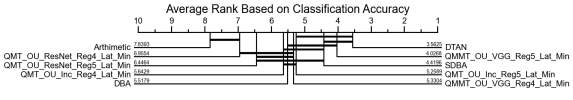
(d) Comparison of mean time domain NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques



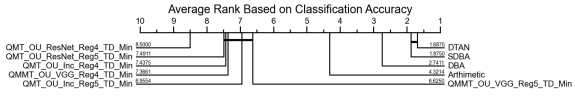
(e) Comparison of median latent space NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques



(f) Comparison of median time domain NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques



(g) Comparison of minimum latent space NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques

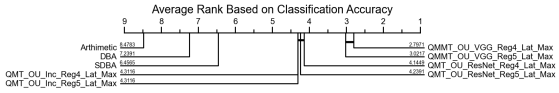


(h) Comparison of minimum time domain NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques

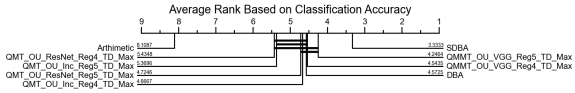
Figure 4.39: CD diagram comparisons of NCC accuracies obtained with alternative averaging techniques and quantile multi tasking regression autoencoders that encouraged over and under estimations. The comparison is performed using the NCC accuracies obtained on 56 UCR archive datasets.

According to Figures 4.39 (h) (a), the latent space performances of the quantile regression multi-tasking autoencoders are better than the state of the art (DTAN) while comparing the maximum NCC accuracies across the different averaging techniques. Moreover, according to Figure 4.39 (b), the time domain accuracies that are obtained with the VGG16 architecture obtained better performances as

compared to DBA. These performances of the VGG16 get closely followed by the Inception architecture. In general, if we compare the performances of the estimates in the registered space of the averaging techniques, the estimates of the multi-tasking autoencoders are far better than any of the alternatives. We make these conclusive remarks since their worst-case performance is comparable to the state-of-the-art as shown in Figure4.39 (g). In this aspect, in the time domain, the worst we can perform is up to arithmetic mean. Moreover, while comparing maximum performances, the multi-tasking quantile regression is performing better than DBA in DTW space. In reality, we found this assessment to also be evident while performing the comparison on 72 UCR archive datasets as shown in Figure 4.40 (b). In this aspect, we found the VGG16 architecture to perform better than the alternatives. With this said we finalize this chapter by first presenting the standard deviations (σ)



(a) Comparison of maximum latent space NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques



(b) Comparison of maximum time domain NCC accuracies with maximum accuracies obtained with the estimates of alternative averaging techniques

Figure 4.40: CD diagram comparisons of NCC accuracies obtained with alternative averaging techniques and quantile multi tasking regression autoencoders that encouraged over and under estimations. The comparison is performed using the NCC accuracies obtained on 72 UCR archive datasets.

associated with NCC. Following this, to make the discussion made so far complete, we present the

Table 4.22: The average standard deviations of the NCC accuracies obtained by different λ pair. In the table, we indicate the different λ pair configuration as $\lambda_x : x = \{0, 1, 2, 3, 4, 5\} = \{(0.85, 0.85), (0.75, 0.75), (0.65, 0.65), (0.15, 0.15), (0.15, 0.15), (0.15, 0.15)\}$, where QMMT_VGG, QMT_Inc, QMT_ResNet correspond to the quantile multi-tasking autoencoders based on the VGG16, Inception, and ResNet architectures.

Techniques	x in λ_x	Latent Space $\pm\sigma$ in %	Time Domain $\pm\sigma$ in %
QMMT_VGG	0	10.16	11.23
QMT_Inc		7.34	9.73
QMT_ResNet		6.61	7.68
QMMT_VGG	1	8.32	10.21
QMT_Inc		6.23	8.82
QMT_ResNet		6.31	7.34
QMMT_VGG	2	6.99	8.52
QMT_Inc		6.00	8.48
QMT_ResNet		8.14	10.30
QMMT_VGG	3	5.39	6.83
QMT_Inc		8.41	6.33
QMT_ResNet		6.21	7.50
QMMT_VGG	4	6.33	7.57
QMT_Inc		5.70	8.46
QMT_ResNet		6.86	9.69
QMMT_VGG	5	5.39	6.83
QMT_Inc		5.37	8.39
QMT_ResNet		6.81	8.77

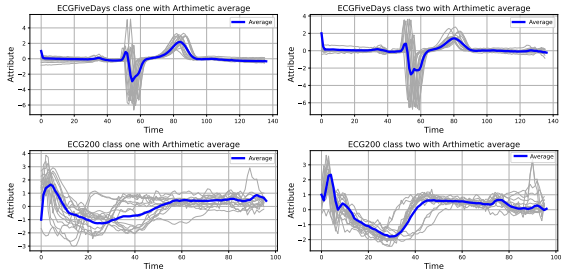
time domain estimates corresponding to the UCR archive datasets. With these in mind, in Table 4.22, we have summarized the standard deviations among the NCC accuracies obtained with 25 iterations of training. In general, for the λ pair values that encouraged overestimations, the VGG16 architecture obtained the lowest standard deviations. Additionally, we found this architecture to provide better time domain estimates compared to the alternatives. Practically, this is evident due to its layer arrangement that consecutively filters out latent features. However, overall, the NCC accuracies obtained with the Inception architecture are relatively stable across all λ pair values.

With this said, in Figure 4.41, we have presented the time domain estimates that correspond to the *ECG200* and *ECGFiveDays*. These estimates are generated by the multi-tasking quantile regression autoencoders while they are encouraging and discouraging over or underestimates. Moreover, in Table 4.23, we have summarized the NCC accuracies that are obtained with the estimates. Based on the visual demonstrations of the estimates, encouraging over and under estimations comparatively pulled peak values to be above the median reconstruction line. For instance, if we compare the estimates in Figures 4.41 (c) and 4.41 (e), we can see that for the class 2 of the *ECGFiveDays* dataset the negative peak estimation is better while the autoencoder encouraged overestimation. Moreover, the

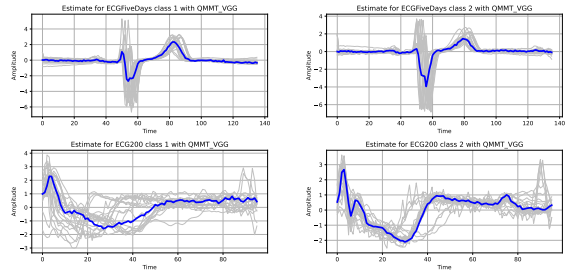
Table 4.23: NCC accuracies for the UCR archive's *ECG200* and *ECGFiveDays* datasets that are obtained with different versions of multi-tasking autoencoders and their counterparts

Techniques	NCC for *ECG200* in %	NCC for *ECGFiveDays* in%
Arithmetic	67	52.96
DBA	65	52.15
SDBA	73	67.02
DTAN	79	**97.79**
MMT_VGG_TD	77	70.27
MT_Inception_TD	78	72.36
MT_ResNet_TD	78	72.71
Var_MMT_VGG_TD	73	70.15
Var_MT_Inception_TD	77	70.49
Var_MT_ResNet_TD	77	74.33
QMMT_VGG_TD	76	78.39
QMT_Inception_TD	79	76.54
QMT_ResNet_TD	79	76.89
QMMT_OU_VGG_TD	79	78.16
QMT_OU_Inception_TD	76	79.21
QMT_OU_ResNet_TD	**82**	76.54

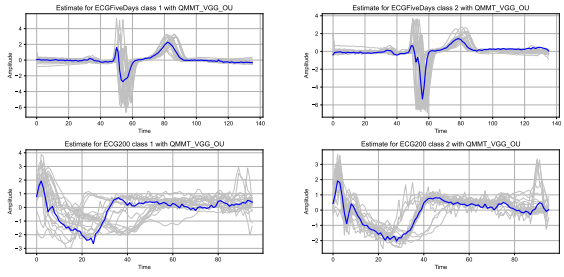
same observation is made for the first class of *ECG200* where the middle sharp rising edge gets better captured. Overall, we found encouraging over and under estimations to often give slightly better performances compared to the discouraging ones. To demonstrate this statistically, in Figure 4.42, we identified and selected the best performing λ values for both arrangements and compared them using a hypothesis test. From Figure 4.42 (a), we can see that the VGG16 architecture obtained better latent



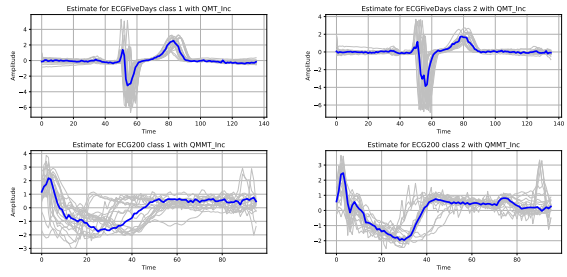
(a) Arithmetic means



(b) QMMT_VGG's estiamtes

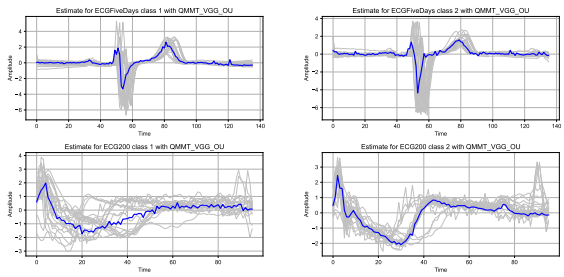


(c) QMMT_OU_VGG's estimates

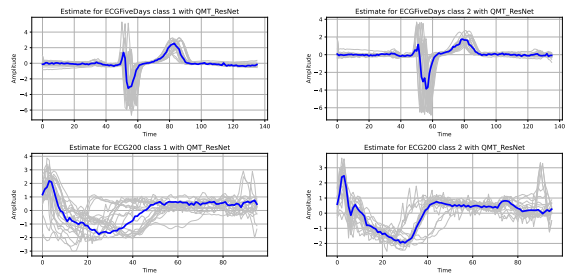


(d) QMT_Inc's estiamtes

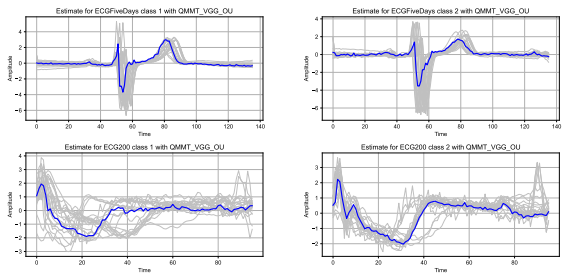
5



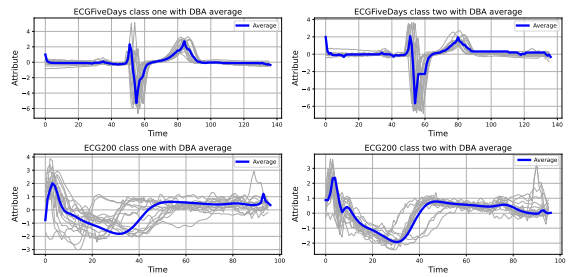
(e) QMMT_OU_Inc's estimates



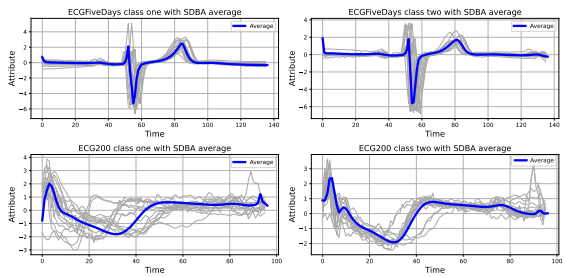
(f) QMT_ResNet's estimates



(g) QMT_OU_ResNet's estimates



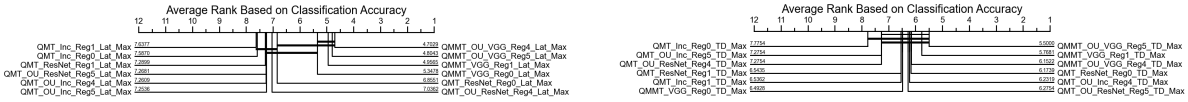
(h) DBA's estimates



(i) SDBA's estimates

Figure 4.41: Averages estimated for the UCR archive's *ECG200* and *ECGFiveDays* datasets using quantile multi-tasking autoencoders while they encouraged and discouraged over and under estimations

space Friedman rank when encouraging overestimations. However, in the Wilcoxon post hypothesis test, we found it statistically equivalent to a VGG16 and ResNet configurations that discouraged over and under estimations. Nevertheless, even under this equivalence, we also can see another VGG16 configuration that encouraged over and under estimation performing better than the alternatives in a statistically indifferent manner. Similarly, we can also observe that in the time domain the VGG16



(a) Comparison of maximum latent space NCC accuracies

(b) Comparison of maximum time domain NCC accuracies

Figure 4.42: CD diagram comparisons of NCC accuracies obtained with quantile multi-tasking autoencoders while encouraging and discouraging over and under estimations. The comparison is performed using the NCC accuracies obtained on 72 UCR archive datasets.

architecture obtained a better Friedman average rank when it encourages over and under estimations. However, the post hypothesis test shows that it is statistically indifferent to a ResNet and Inception configurations that discouraged over and under estimations. However, overall, we find the cases that encouraged over and under estimations to dominate the right side of the Friedman average rank. In conclusion, of all the multi-tasking configurations, we found the quantile regression arrangement to give better time domain estimates. However, in the context of encouraging or discouraging under and overestimations, we have made the following observations. First, if the data under observation has amplitude values that are more or less similar, we found the arrangements that discouraged over and under estimation to give better time domain estimates. This is because, for such datasets, the median reconstruction line is often optimal and there is no need to shift it significantly. On the contrary, if the amplitude variation among members of the averaged set is significant, we suggest that under and overestimations get encouraged. However, care should be taken so that it does not shift the median line in a manner that distorts the time domain projections.

5 Time Series Averages in Cluster Level Forecasting

We like to finalize this dissertation by presenting the implication of our studies in the context of a real-world application. In this regard, we present one of our work as a demonstrative example and later show how our averaging proposals can get put to use [114]. In [114], we proposed a hybrid cluster level forecasting technique to predict for the traffic load offered to Universal Mobile Telecommunication System (UMTS) radio nods (Node B) located within Addis Ababa, Ethiopia.

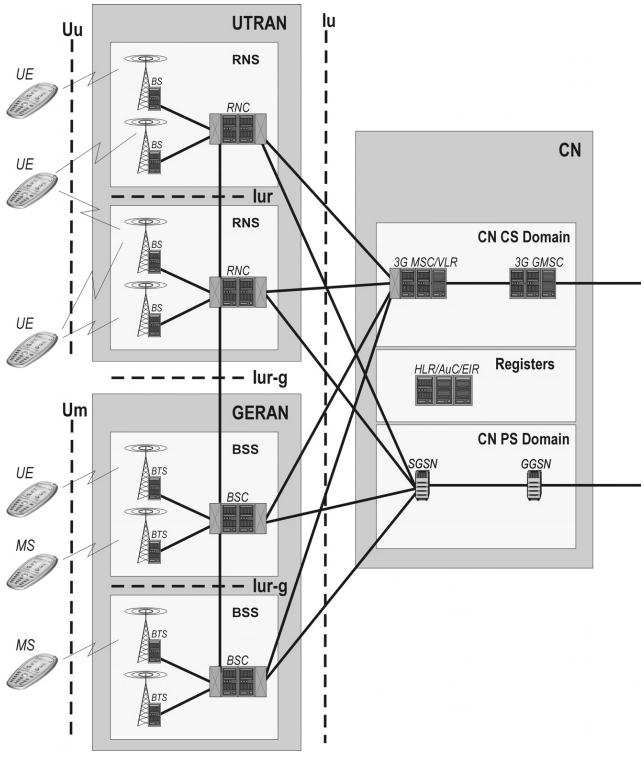


Figure 5.1: A basic UMTS based wireless communication network architecture [115]

In practice, a UMTS network is composed of three basic building blocks: the Core Network (CN), the UMTS Terrestrial Radio Access Network (UTRAN) and the User Equipment (UE) [115], [116]. In UMTS, the UTRAN contains the Base Transceiver Station (BTS) or Node B that provides a direct interface to the network user via either an omni directional or directional dipole radio antennas. In practice, a Node B has a services coverage area dependent on data rate and desired quality of service [117]. However, due to its operating frequency that could range up to 2.1 GHz, the coverage area of a UMTS Node B is significantly lower than the BTS utilized in predecessor networks such as the Global System for Mobile Communication (GSM). In practice, a GSM BTS operates on either 900 MHz or 1800 MHz ranges with slightly longer wavelengths. To this end, the electromagnetic waves generated by GSM radio nodes comparatively travel long distances without significant attenuation. In this aspect, a

UMTS get expected to have a relatively higher number of radio nods to increase its coverage area. This, in turn, requires the deployment of controlling mechanisms that ensure radio nodes provide coverage without significant interference among themselves. In this aspect, UMTS utilizes Radio Network Controller (RNC) units that are incorporated within the UTRAN [115], [116]. In general, a RNC is expected to perform: admission control, radio resource control, radio barrier setup (release), handover, etc [116]. For instance, if a user moves away from one radio node to another controlled by the same RNC, then the RNC is expected to ensure the smooth transfer of radio links without service interruption. On the contrary, if a user is moving outside the domain of the RNC and if the user is utilizing a voice service, then the RNC will pass the link management to a Mobile Switching Center (MSC) located within the core network. In general, by utilizing such a modularized approach, UMTS became one of the most successful wireless cellular communication networks starting in early 2000.

In practice, maintaining a high quality of service in such big wireless cellular networks is not a trivial task. To make matters worse, a UMTS network get expected to provide both packet and switched network services. In reality, these services place different sets of requirements to guarantee an acceptable quality of services. In this regard, network operators such as Ethio Telecom establish a range of Key Performance Indicators (KPI). For instance, for a UMTS switched voice service, operators continuously assess: call drop rates, call denial rates, successful handover rates, etc. In practice, there are a range of factors contributing to the degradation of such KPIs. For instance, a UMTS network could face a lot of unsuccessful handovers due to blind spots due to the obstruction of radio signals associated with natural or artificial landmarks. In such cases, corrective measures get taken by identifying a better location for the radio units. In another aspect, the network could be affected by an increase in call drop rates, call denial rates, and unsuccessful handovers. In practice, such cases could happen due to spikes in the offered traffic load. In this regard, operators often asses traffic loads using measurement units such as the Erlang. For instance, (5.1) demonstrates how Erlang A gets computed, where c is the average number of arriving calls within a duration T. Moreover, h is the average call duration.

$$E_A = \frac{c \times h}{T} \tag{5.1}$$

In the context of an offered load, different wireless communication systems often utilize different techniques to increase their traffic capacity. In this regard, wireless radio nodes could multiplex their users through time and frequency. In other words, a radio unit can operate on multiple frequencies (channels) that are divided in time. Moreover, in some cases such as the UMTS, the time and frequency multiplexed channels can further increase their sharing capacity using codded transmissions. However, in reality, such systematic designs often could increase sharing capacity up to a certain limit. To this end, in practice, operators are expected to often resort to network optimization. This in turn could include the deployment of additional transceiver modules or an entire radio unit. Moreover, in extreme cases, operators are expected to upgrade the utilized technology, say for instance from 3^{rd} generation (UMTS) to 4^{th} or 5^{th} generation systems. In reality, the cost of making such minor or major network optimization is often high. To this end, operators are expected to carefully assess

the *"when to"* and *"how to"* upgrade (optimize) questions. In reality, the *when to* upgrade (optimize) question is highly correlated to the investment return time of the deployed infrastructures. On the contrary, the answer to the *"how to"* upgrade question is often dependent on financial feasibility. In general, in either case, the decision has to be made in an intelligent manner.

With these in mind, in [114], we focused on one major factor that impact the KPI of UMTS's data service, i.e., offered data traffic load. In this aspect, we set out to develop an efficient forecasting technique for the offered data traffic loads to 739 UMTS radio units that were located within the areas of Addis Ababa, Ethiopia. The radio units were a part of Ethio Telecom's 3^{rd} generation wireless network. In our work, we strongly believed that by deploying efficient forecasting models, the operators could utilize short or long-term forecasts to predict minor and major optimization requirements. With this understanding, we approached the problem using two key steps. First, we needed to identify a proper forecasting model that better captures the patterns observed in the traffic loads. In addition to this, we were also expected to address the limitations observed in the identified forecasting models and propose possible mitigation techniques. With this said, we will next present the steps we took while identifying the proper forecasting model.

5.1 A Cluster Level Data Traffic Forecasting

We have started our search for a suitable forecasting model from the analysis of the traffic load datasets. In this aspect, we had 739 time series that were defined by taking an hourly measurement of the total data traffic in Giga Bytes (GB). Moreover, the measurements were taken: for four consecutive months (i.e., from September 2019 to March 2019), seven days of a week, and 24 hours of a day. In other word, the dimension (length) of the time series were $24 \times 7 \times 16 = 2688$. As demonstrative examples, in Figure 5.2, we have plotted datasets extracted from four radio nodes that correspond to data traffic loads offered within two weeks period. In reality, plots based on longer duration also presented similar repeating patterns (seasonality). In practice, such seasonalities get expected to arise from people performing their daily routines. For instance, we expect a radio node near city centers to have a peak traffic demand during working hours. Moreover, the traffic demand on such radio nodes is also expected to decrease as people return to their homes and on weekends. On the contrary, a radio node serving residential areas get expected to have a higher data traffic load when: people return to their homes, in the early mornings, early evenings, mid-night (related to cost reduction), and on weekends. Additionally, as more people join the network, the traffic load in both areas is expected to show a steady increase. In practice, this often accounts for trends evident in the datasets. Practically, the presence of such seasonalities and trends are mathematically analyzed by decomposing a dataset into its three basis components, i.e., seasonality, trend, and residue. To perform the decomposition, we can either assume a dataset is either a linear or a multiplicative combination of the three components [1]. In this regard, we chose the former approach since the datasets contained values that are zero. However, in practice, such decomposition often requires prior knowledge about the periods of seasonalities. In this aspect, since we had no prior knowledge about the exact duration of the seasons, we first had to observe the autocorrelation and partial autocorrelation plots of the datasets. In this aspect, we

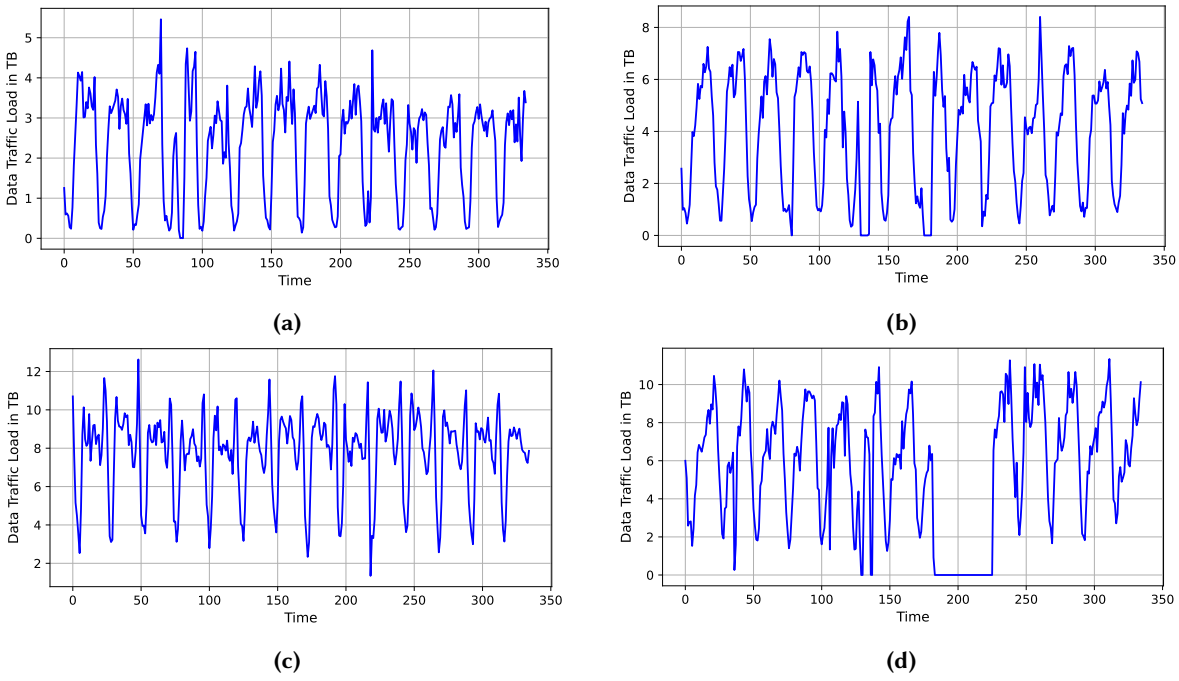
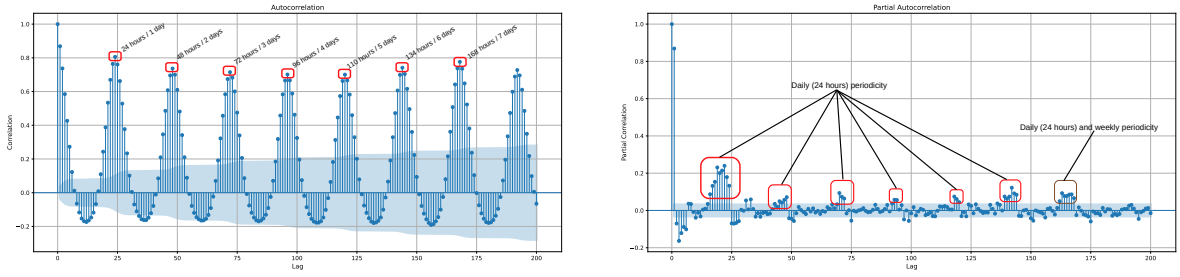


Figure 5.2: Sample data traffic loads offered to four UMTS radio nodes locate within the areas of Addis Ababa, Ethiopia

first plotted the auto and partial correlations using plots similar to Figure 5.3. From the figure, we noted that the autocorrelations obtained a significant correlation at lags of 24 hours. Moreover, they showed a slight increase of correlation every 168 hours (1 week). The same correlation patterns were also observed for the partial autocorrelation plots. To this end, we concluded that the datasets have dual seasonality, i.e., daily (24 hours) and weekly (168 hours). To further assess the validity of the

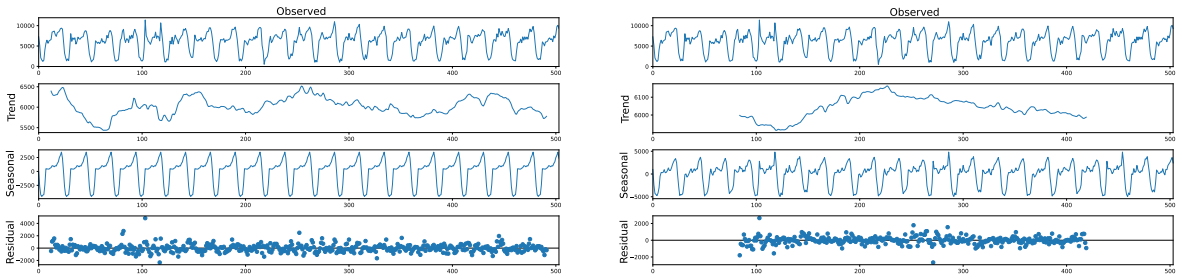


(a) Auto correlation **(b)** Partial auto correlation

Figure 5.3: Auto correlation and partial auto correlation for a sample data traffic that is offered to a UMTS radio node

dual seasonality assumption, we decomposed the datasets into their three basic components. In this regard, Figure 5.4 shows the daily and weekly decomposition of the time series whose autocorrelation plot is shown in Figure 5.3. According to Figures 5.4 (a) and 5.4 (b), the weekly and daily seasonality patterns are different. Thus, this further asserted the presence of dual seasonality. Based on these observations, we firmly concluded that our forecasting model should be able to account for dual seasonality and trend. In this aspect, we identified two possible forecasting approaches, i.e., LSTMs or classical linear and nonlinear seasonal forecasting models. However, rather than relying on one

of the approaches, we proposed to benefit from the advantages offered by both categories. To this end, we proposed a hybrid forecasting model based on a Seasonal Autoregressive Integrated Moving Average (SARIMA) [118] and LSTM neural network [119].



(a) Decomposition based on a Daily seasonality

(b) Decomposition based on a weekly seasonality

Figure 5.4: Auto correlation and partial auto correlation for a sample UMTS data traffic load. In order to plot the decomposition, we have only taken a segment of the dataset that corresponds to four weeks of measurements for better visibility

In practice, a SARIMA forecasting (regression) model is derived from its non seasonal version, i.e., an Auto Regressive Integrated Moving Average (ARIMA). Moreover, an ARIMA is in turn built from two major building blocks, i.e., Auto Regressive (AR) and Moving Average (MA). In general, given a time series of the form $Y = \{y_1, y_2, , \ldots, y_{t-1}\}$, a p^{th} order AR model tries to predict y_t using the linear combination of its p predecessor values and a constant C. This is mathematically summarized as AR(p) (5.2), where e_t is the AR forecasting error for a sample at t.

$$y_t = C + \alpha_1 \times y_{t-1} + \alpha_2 \times y_{t-2} + \alpha_3 \times y_{t-3} + \ldots + \alpha_p \times y_{t-p} + e_t \tag{5.2}$$

On the contrary, a q^{th} order MA model tries to forecast a future value using q predecessor forecasting error values generated from an AR ($\{e_{t-1}, e_{t-2}, \ldots, e_{t-q}\}$), i.e., MA(q) shown in (5.3). Thus, an Auto Regressive Moving Average (ARMA) forecasting model combines the two estimation techniques. However, in practice, most temporal datasets often contain a continuously increasing/decreasing constant offset corresponding to trends. In reality, such offsets were found to affect the performances of linear models that assume the modeled data is stationary [1]. Thus, researchers often propose to take the difference (D) of a dataset prior to fitting the forecasting models, i.e., $y\prime_t = D(y_t) = y_t - y_{t-1}$. Thus, this way, one can take the integration of the predicted values as the reverse operation, i.e., $I(y_t) = y_t + y_{t+1}$. With this in mind, researchers often propose to utilize the updated version of ARMA, i.e., the ARIMA.

$$y_t = C + \zeta_1 \times e_{t-1} + \zeta_2 \times e_{t-2} + \zeta_3 \times e_{t-3} + \ldots + \zeta_q \times e_{t-q} \tag{5.3}$$

With these understandings, we can summarize an ARMA(p, q, D) model using (5.4). However, in most practical cases, the difference operation (D) is performed d a number of times. To incorporate this concept, most literature expresses the degree of the difference operation as D^d. Moreover, to express (5.4) in a more compact manner, they also define the lag operator B, where $B^n y_t = y_{t-n}$. Thus we can write $D(y_t) = y\prime_t = y_t - y_{t-1}$ as $y\prime_t = (1 - B)y_t$. Furthermore, we can write

$y''(t) = y'_t - y'_{t-1} = y_t - y_{t-1} - (y_{t-1} - y_{t-2}) = y_t - 2 \times y_{t-1} - y_{t-2} = (1-B)^2 y_t$. Following this path, the $d^t h$ difference of y_t can be written as $y_t^d = (1-B)^d y_t$. To this end, an ARIMA(p, q, d) linear forecasting model can now compactly be written as (5.5).

$$y'_t = C + \left(\zeta_1 e_{t-1} + \zeta_2 e_{t-2} + \ldots + \zeta_q e_{t-q}\right) + \left(\alpha_1 y'_{t-1} + \alpha_2 y'_{t-2} + \ldots + \alpha_p y'_{t-p} + \epsilon_t\right) \quad (5.4)$$

$$C + \left(1 + \zeta_1 B + \zeta_2 B^2 + \ldots + \zeta_q B^q\right) e_t = \left(1 + \alpha_1 B + \alpha_2 B^2 + \ldots + \alpha_p B^p\right) (1-B)^d y_t \quad (5.5)$$

In general, by combining the AR and MA terms, the ARIMA proved to be efficient in most practical cases [1], [119], [120]. However, when datasets presented some form of seasonality, it performed poorly [118]. This is because, for the basic ARIMA, seasonal values will aggregate and behave as a trend that influences the selection of optimal model coefficients (α and ζ). To this end, in such cases, the seasonal version of the ARIMA or Seasonal Autoregressive Integrated Moving Average (SARIMA) is often proposed. In this regard, in order to better capture seasonality, a SARIMA combines N ARIMA models in a multiplicative manner as shown in (5.6), where, S_i is the i^{th} seasonality component of a datasets. In our context, since the data traffics presented two forms of seasonality, we proposed to utilize Double Seasonal Auto Regressive Integrated Moving Average (D-SARIMA) model, where, $S_1 = 24$ hours and $S_2 = 168$ hours.

$$N - SARIMA = ARIMA(p,q,d) \times ARIMA(P_1, Q_1, D_1)_{S_1} \times \ldots \times ARIMA(P_N, Q_N, D_N)_{S_N} \quad (5.6)$$

With these in mind, we next placed our focus on the residues of the Double Seasonal Auto Regressive Integrated Moving Average (D-SARIMA) model. In reality, the residues could either correspond to a portion of the dataset that can not be represented by a composition of linear models or they could be some random noise. In either case, statistical modeling of the datasets get expected to give us a better understanding of the underlying situation. In this regard, we analyzed the resemblance of our datasets to a Gaussian distribution. In reality, we chose to compare the datasets to the Gaussian distribution with the neural networks in mind. In this context, on one hand, if the datasets closely resemble some sort of Gaussian distribution, we can expect the residues to follow the same pattern due to the linear nature of the forecasting model. On the other hand, in practice, we have neural networks that successfully utilized Gaussian distributions to model temporal datasets [38]. Thus, we can safely expect LSTM network to extract some additional meaningful information from the residues. With this in mind, to visually assess the distribution of the datasets, we utilized histogram and Quantile Quantile (QQ) plots. To plot the histogram, we first divided the traffic demand into 25 Tera Bytes histogram bins. Following this, we fitted the best possible Gaussian distribution on the hourly traffic measurements. We then plotted the histograms and the fitted Gaussian distribution curve on the same figure. Finally, we observed if the histograms preserve the symmetry of a Gaussian distribution curve. On the contrary, for the QQ plots, we divided the hourly traffic measurements into different quantiles, i.e., based on their values. Following this, samples falling within a given quantile get compared to similar valued samples of a theoretical Gaussian distribution. The quantile values of the compared samples are then recorded as a two dimension point. Finally, the comparisons are plotted using a scatter plot. Thus, if the quantiles of the traffic data match the quantiles of a Gaussian

distribution, then the two dimensional points of a QQ comparison will be along the diagonal line of the scatter plot. In general, for both plots, the UMTS traffic datasets more or less followed a Gaussian distribution. For instance, in Figure 5.5, we presented the histogram and QQ plots of datasets obtained from four different UMTS radio nodes.

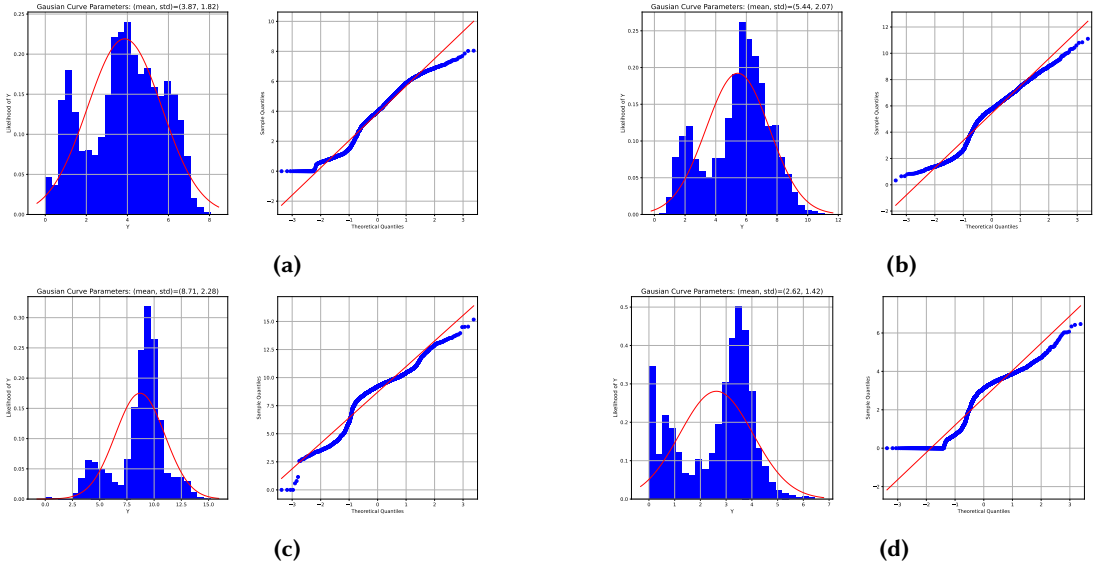


Figure 5.5: Histogram and QQ plots on the traffic datasets collected from four UMTS radio nodes located within Addis Ababa, Ethiopia

With these observations at hand, we assumed that the residues are a combination of nonlinear traffic behaviors and random noises that get distributed in a Gaussian manner. We then proposed to extract additional information from residues which are expected to be distributed in a Gaussian manner using an LSTM network. In other words, we expect the LSTM to properly model the nonlinear portion of the residues and filter out the noise. In general, with the proposed hybrid forecasting model, we generated forecasts using two steps. First, we identified the best D-SARIMA model. We then took the residue of the fitted model and used them to train an LSTM network. Finally, we estimated $\hat{y}_t$ by taking the linear combinations of the D-SARIMA and LSTM forecasts. Mathematically speaking, given $Y = \{y_1, y_2, \ldots, y_{t-1}\}$ and a residue r(t), the proposed hybrid model generates forecasts using (5.7), where $\hat{L}_t = D - SARIMA(Y)$ and $\hat{r}_t$ is a random noise term.

$$\begin{aligned} \hat{y}_t &= SARIMA(p,q,d)(P_1,Q_2,D_1)_{S_1}(P_2,Q_2,D_2)_{S_2}(Y) + LSTM(r(t)) \\ r_t &= y_t - \hat{L}_t + \hat{r}_t \end{aligned} \tag{5.7}$$

In addition to these considerations, for the proposed approach, we have kept the size of the LSTM model to be relatively small to minimize the additional computational requirement. With this in mind, we construct the network from two layers of LSTM units. Moreover, at the output we have utilized a time distributed *Dense* layer in order to deploy *Dense* layer on each time slices [29]. In general, the first two LSTM layers respectively had 128 and 64 hidden nodes that are *ReLu* activated. However, we set the output dense layers to utilize a *Sigmoid* activation function. We chose the *Sigmoid* activation since we planned to normalize the datasets prior to LSTM training and D-SARIMA fitting.

Even though we expected the hybrid approach to address most of the issues, in the end, we got left with two major challenges. The first challenge is that the D-SARIMA model is mainly designed to handle a single dataset at a time. However, at our disposal, we had 739 radio stations which we later reduced to 729 due to missing values. To this end, if we follow a direct approach, we will end up with 729 forecasting models. In practice, this would be inefficient for two main reasons. First, we will only be observing a minor subset of the operator's radio stations. In reality, the network operator has thousands of such stations distributed throughout the country. Thus, deploying forecasting models on each radio node will quickly become unscalable due to the sheer number. Secondly, if we see the forecasting problem from an operator's perspective, we expect the operator to base its higher-level decisions on observing aggregate or average traffic demands. However, base station level forecasting models often treat radio nodes as isolated entities. In reality,this is far from the dynamics of mobile communication systems [121]. In this apspect, an isolated D-SARIMA model is incapable of capturing the spatial information evident within the datasets due to user mobility. Thus, in addition to failing to provide generic picture of traffic demnads, base station based forecasting models are expeced to give poor performance. With these understandings in mind, in [114], we proposed to cluster the radio units based on their traffic patterns. We then aimed to utilize the centroid of the clusters to fit the forecasting models. To meet this objective, we utilized K-Means as our main clustering algorithm [22]. In general, we have summarized the steps taken to generate forecasts in Figure 5.6.

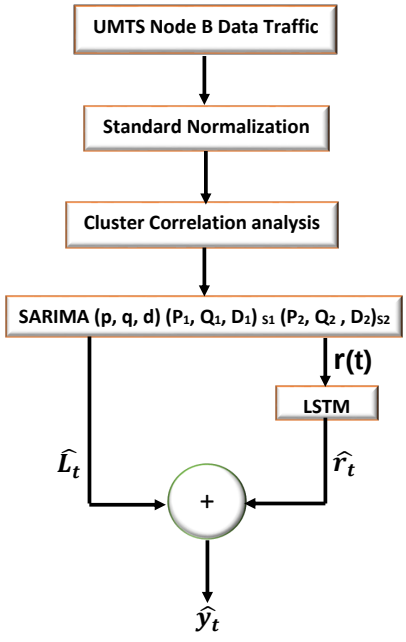


Figure 5.6: Steps taken in the proposed hybrid cluster level UMTS data traffic forecasting

In reality, given an optimal number of clusters and cluster centroids, we expected the clusters to represent a specific group of radio network users over a specific coverage area. For instance, residential areas, commercial areas, mixed-use areas, etc. However, in practice, mobile network traffic is not expected to be confined to a given geographical area due to the presence of user mobility. For example, we expect the data traffic demand to decrease in commercial areas, for instance, when shops

close at night. When this is the case, we expect a rise in the traffic demand of residential areas since users most probably have gone to their homes. To this end, we argued that the clusters can not be taken as independent of each other. On contrary, we can safely assume a given cluster embeds explicit information about the others to some degree. In this aspect, the centroids (averages) of the clusters summarize this information in a relatively unbiased manner, i.e., without explicitly favoring a given cluster member. With this understanding, in our proposed hybrid forecasting approach, we first identified the correlation between the centroids of the clusters. Following this computation, we selected a given centroid and identified its most correlated neighboring cluster centroids. We then used these centroids as exogenous variables in the D-SARIMA model fitted on the centroid of a cluster. With this said, we will next present the experimental setup and the experimental evaluations reported in [114].

5.1.1 Experimental Setup

In [114], before any model training (fitting), we first normalized the traffic datasets using the Sklearn implementation of the StandardScaler [122]. This Python package utilizes (5.8) to normalize the datasets, where $Y \in \mathbb{R}^M$, μ, and σ are respectively the normalized series, its mean and standard deviation.

$$Y_N = \frac{Y - \mu}{\sigma} \tag{5.8}$$

In reality, we only performed the normalization for datasets that had no missing values. In this regard, we identified 10 datasets (corresponding to 10 radio nodes) that had missing values for various reasons, for instance, due to power outage. Thus, in reality, we ended up only utilizing datasets obtained from 729 radio nodes, i.e., out of the 739 radio nodes. We then used these datasets for two types of forecasting, i.e., Base Station (BS) level and Cluster Level (CS). At the CS level forecasting, we first performed the normalization and then conducted an inter-cluster inertia analysis. This analysis was performed to identify the optimal cluster number. In practice, inter-cluster inertia is another term for the average per-cluster Within Group Squared Sum (WGSS). After performing this analysis, we grouped the datasets (radio nodes) into N clusters, where N is an integer that minimized the aggregate average inter-cluster inertia. Finally, we took the centroid (averages) of the clusters and segment them for training, validation, and test. In this regard, given $Y \epsilon \mathbb{R}^M$: $\lfloor 0.8 \times M \rfloor$, $\lfloor 0.1 \times M \rfloor$ and $\lfloor 0.1 \times M \rfloor$ time stamps values of Y were taken for training, validation and testing. In addition to this segmentation, we also computed an intra-cluster correlation matrix using the correlation of the cluster centroids.

After performing these pre-processing steps, we fitted a range of $SARIMA(p,\ q,\ d)(P_{S_1},\ Q_{S_1},\ D_{S_1})_{S_1}$ $(P_{S_2},\ Q_{S_2},\ D_{S_2})_{S_2}$ models for different values of $(p,\ q,\ d)$, $(P_{S_1},\ Q_{S_1},\ D_{S_1})$ and $(P_{S_2},\ Q_{S_2},\ D_{S_2})$. We performed these iterative model fitting to identify the best D-SARIMA model parameters. We have conducted these repeated trials using a function within the Smooth R package, i.e., the Automatic Multiple SARIMA (auto.msarima) [123]. After identifying the best performing D-SARIMA model, we took its residues to train the proposed LSTM network. We have trained the LSTM network using: a batch size of 24, for 100 epochs and a sequence of two days (48 hours) of past observations. On the contrary, the D-SARIMA was fitted using $3\frac{1}{4}$ months of past observations. We have finally used

the trained D-SARIMA and LSTM network to predict for future 48 hours using (5.7). In addition to this training, we also generated similar forecasts using only a D-SARIMA and LSTM models that are fitted or trained on the cluster centroids. However, for the non-hybrid D-SARIMA model, we have not included any type of exogenous variables. In other words, we have not considered the intra-cluster correlation. Finally, in order to evaluate the performances of the predictions, we utilized average RMSE and Mean Absolute Error (MAE) that are given in (5.9) and (5.10), where N is the number future time stamps the models predicted for. Overall, we made these predictions on segments of the cluster centroids. On the contrary, while assessing the prediction quality at a Base Station (BS) level, we randomly selected one of the radio nodes and we generated forecasts using: the hybrid forecasting approach, the standalone D-SARIMA and LSTM models. However, for the BS level forecasting, we have excluded incorporating exogenous variables identified using intra-cluster centroid correlation while performing cluster level predictions. Practically, the exclusion of the exogenous variables got employed in all types of proposed forecasting approaches.

$$RMSE_{forecasting} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(y_{t_i} - \hat{y}_{t_i})^2} \tag{5.9}$$

$$MAE_{forecasting} = \frac{1}{N}\sum_{i=0}^{N-1}|y_{t_i} - \hat{y}_{t_i}| \tag{5.10}$$

5.1.2 Experimental Results

We started our experimental evaluations by assessing the inter-cluster inertia. In this regard, we performed 21 K-Means clustering with cluster sizes that ranged from 1 to 21. Moreover, we performed each K-Means clustering for 1000 iterations. We have summarized the inter-cluster inertia of these K-mean trials as shown in Figure 5.7. According to Figure 5.7, the cluster inertia shows a sudden fall starting from a cluster size of two. Furthermore, it appears to be converging as the number of clusters increases. However, we noted that if we selected a higher number of clusters, it would mean more forecasting models. In other words, by defining more clusters, we would end up with a BS level forecasting. With this in mind, we decided to group the radio nodes using five clusters.

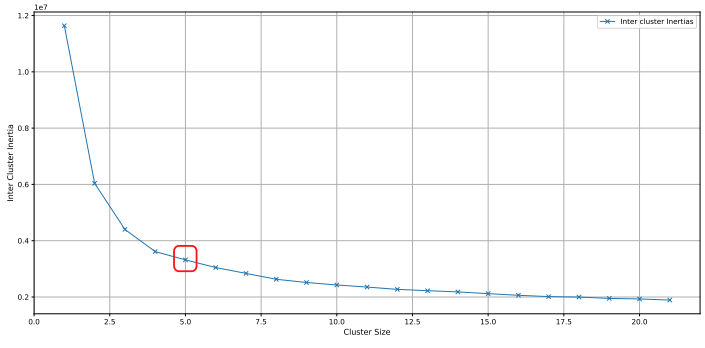


Figure 5.7: Inter cluster inertia for 729 time series corresponding to a data traffic load offered to UMTS radio nodes

In Figure 5.8, we have shown the geographical location of the radio nodes belonging to the five

clusters. Moreover, it also shows a portion of the cluster's centroids, i.e., from January 09, 2019, at 15:00 hr to January 15, 2019, at 15:00 hr. Roughly, clusters four and five corresponded to radio nodes within city centers. On the contrary, the remaining clusters were relatively located on the outskirts of the city. With this in mind, we next computed the intra-cluster correlation and plotted its heat map as shown in Figure 5.9. The correlation heat map further validates our initial argument that we can not

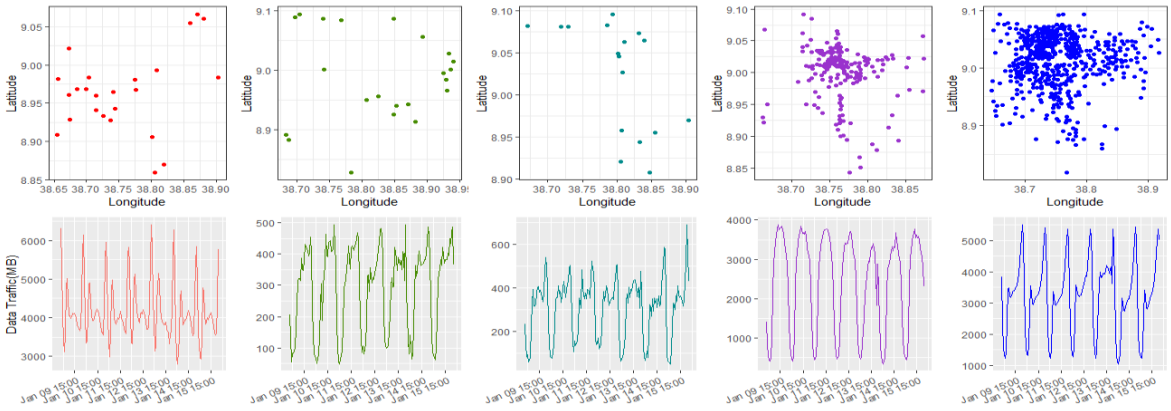


Figure 5.8: Geographical location of the clustered radio nodes and their respective cluster centroids [114]

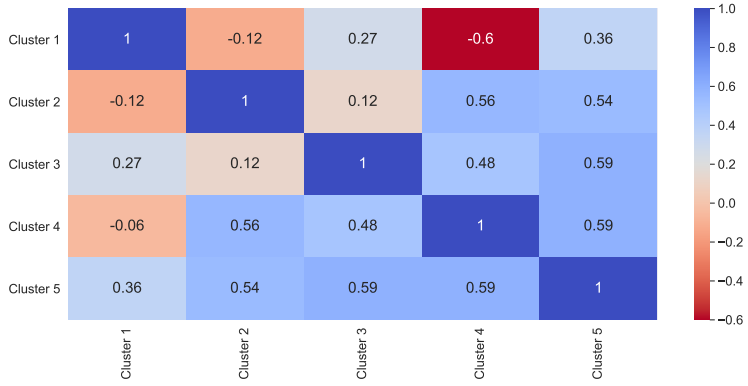


Figure 5.9: Heat map corresponding to the correlation of the centroids of the five clusters identified by K-Means [114]

consider each cluster to be independent. With this in mind, we first performed the three proposed forecasting approaches at the BS level. Moreover, for all the forecasting models that incorporated D-SARIMA, we used D-SARIMA $\{(1,\ 0,\ 2),\ (2,\ 1,\ 0)_{24},\ (0,\ 1,\ 1)_{168}\}$ which gave a better forecasting errors on a validation sets. In general, we obtained the average RMSE and MAE BS level forecasting errors shown in Table 5.1 [114].

Table 5.1: Performance comparison of a hybrid forecasting model and its counterparts

Models	RMSE	MAE
BS-DSARIMA	**1.229**	**1.385**
BS-Hybrid	1.517	1.667
BS-LSTM	1.237	1.408

The results in Table 5.1 show that the hybrid forecasting approach performed poorly when spatial information is not incorporated. Moreover, we also found the LSTM contributing negatively to the overall hybrid approach. We make this conclusive remark since, at the base station level, the D-SARIMA was modeling the datasets relatively well. In other words, for the focus radio node, the LSTM was unable to extract meaningful information from the residues. To this end, while adding forecasts generated from the residues, it introduced unnecessary offsets that shifted the aggregate forecasts in an undesired direction. We then conducted cluster-level forecasting using the centroids identified at earlier stages. We have utilized the same parameter configurations for the D-SARIMA model identified at the BS level forecasting. However, in this case, we included the centroids of the highly correlated neighboring clusters as an exogenous variable for the D-SARIMA model belonging to the hybrid forecasting model. On the contrary, for the stand-alone LSTM and D-SARIMA forecasting models, we trained (fitted) the models on the cluster centroid to which the previously analyzed BS belonged. In general, Table 5.2 summarizes the average two days of forecasting errors corresponding to the three proposed approaches.

Table 5.2: Performance comparison of a hybrid cluster level forecasting and its counterparts

Models	RMSE	MAE
CS-DSARIMA	0.548	0.872
CS-Hybrid	**0.363**	**0.416**
CS-LSTM	0.548	0.617

According to Table 5.2, the hybrid prediction model outperforms all the other approaches. Moreover, the additional spatial information provided by the clustering has improved the forecasting by a factor as high as 60%. In this regard, Figure 5.10 depicts the main reason behind this significant improvement. According to Figure 5.10, the forecasts made at the base station level were often under estimations. On the contrary, the additional spatial information has enabled the cluster level approaches to better model peak time and lower traffic demands.

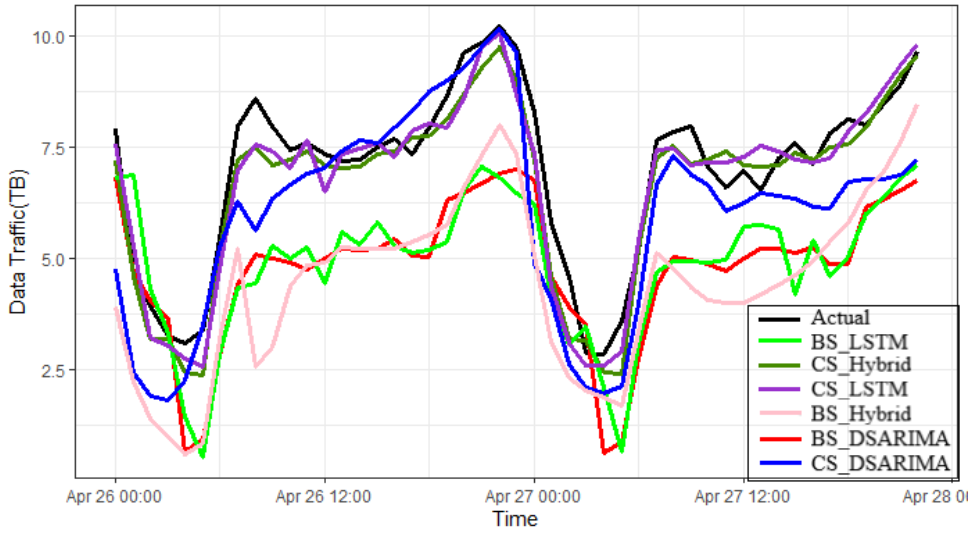


Figure 5.10: Comparison of a 48 hours forecasts that are performed at the based station and cluster level [114]

However, even though the cluster level approach showed significant improvements, we observed two major gaps that were not addressed in [114]. First, the approach got evaluated on a single base station traffic data. To this end, it was not clear whether the clustering or the hybrid approach delivered the performance boost. Moreover, in [114], the authors also acknowledged that the quality of the cluster centroid and the cluster formation process could significantly impact the performance of cluster level forecasting. In this regard, the authors suggested that alternative clustering and centroid estimation techniques should also get assessed. To answer these questions, we propose to assess the impact of the clustering and cluster centroid (average) estimation process on the performance of cluster-level forecasting. In this regard, we re-evaluate the representativeness of the cluster level forecasting compared to the traffic demands of the individual base stations. We aim to conduct the comparison using three clustering techniques: K-Means, DBA k-Means, and deep embedding clustering with time domain centroid estimated using a multi-tasking autoencoder. However, to make the comparison unbiased, we base the model on D-SARIMA.

5.2 Assessing the Impact of Clustering Techniques and Quality of Clusters Centroids on Cluster Level Forecasting

In reality, two key parameters significantly influence the performance of the cluster-level forecasting approaches, i.e., the way clusters get formulated and the representativeness of the cluster centroids (average). In the former case, time series clustering gets expected to be affected by at least the presence of outliers and temporal distortion [21]. This is because most renowned time series clustering techniques rely on distance metrics to identify cluster membership. To this end, in the presence of outliers, cluster centroids could be forced to get shifted closer to the outliers [124]. This, in turn, is expected to increase the inter-cluster inertia and decrease the representativeness of the centroids. Moreover, if the clustered sets are highly affected by temporal distortions, then estimating the cluster centroids via arithmetic mean is often not efficient [19]. With these understandings, in this section, we propose to utilize additional two clustering approaches; i.e., DBA based K-Means and deep embedding clustering [6], [19]. Moreover, since the deep embedding clustering gets performed in the latent space of a neural network, we propose to utilize our proposed multi-tasking autoencoder arrangement to estimate the time domain cluster centroids. With this said, we next present a review of the DBA based K-means and deep embedding clustering approaches.

5.2.1 Dynamic Time Warping Barycenter Averaging Based K-Means

In order to account for the impact of temporal distortions, [19] proposed to integrate DBA and DTW into K-means. In this regard, the authors suggested utilizing DBA while estimating the cluster centroids. Moreover, they proposed to utilize DTW while identifying cluster membership. However, in practice, SDBA and SDTW were also utilized for this variant of K-Means. With these modifications at hand, the overall K-mean clustering gets generalized as shown in Algorithm 2. Even though the

integration of DBA into K-Means get expected to increase its overall computational complexity, in most cases, it often gets counteracted by reduced inter-cluster inertia.

Algorithm 2: DTW and DBA based K-means.

Inputs: $Y = \{Y_1,\ Y_2,\ \ldots,\ Y_N\} :\ Y_i \epsilon \mathbb{R}^M,\ \gamma\ >\ 0$, Number of iterations (N), Number of Clusters (K), DBA or SDBA iteration (I), DBA or SDBA Tolerance (Tol) and DTW distance δ.
Initial centroids = $\{\mu_1,\ \mu_2,\ \ldots, \mu_K\}$
Initial Clusters= $\{C_1,\ C_2,\ \ldots,\ C_K\}$
while K-Means iteration < N **do**
  for j ≤ K **do**
    while DBA (SDBA) Tolerance ≤ Tol **or** DBA (SDBA) iteration ≤ I **do**
      $\mu_j = DBA(C_j)$
    end while
  end for
  D = $\{d_1,\ d_2,\ \ldots,\ d_k\}$
  for i ≤ N **do**
    for j ≤ K **do**
      $d_j = \delta(\mu_j,\ Y_i)$
    end for
    Index=min$\{d_1,\ d_2,\ \ldots,\ d_k\}$
    C_{Index} < - Y_i
  end for
end while
Output: $\{\mu_1,\ \mu_2,\ \ldots,\ \mu_K\}, \{C_1,\ C_2,\ \ldots,\ C_K\}$

5.2.2 Deep Embedding Clustering and Multi-tasking Autoencoer Based Cluster Centroid Estimation

An alternative way of improving the cluster quality would be to cluster temporal datasets based on their dominant latent features (shapes) [6], [55], [56]. In this regard, [6] proposed to cluster time series in the latent space of a denoising autoencoder which the authors named as Deep Embedding Clustering (DEC). The authors proposed to base DEC on denoising autoencoder considered as capable of generating latent features that captured the dominant features of input datasets. With this intuition, [6] proposed to first train a SDAE that were introduced in [125]. In reality, the SDAE introduced in [125] were composed of three *Dense* layers as sown in Figure 5.11. On the other hand, a more complex SDAE can get built by training a stack of such denoising autoencoders. However, when this is the case, the training process of SDAE is relatively complex compared to the training of basic autoencoders. In this aspect, to train a bigger SDAE autoencoder, an elementary SDAE is first trained to reconstruct its input. Following this, the decoder gets removed and the first two layers get used to generate the input for the next stack. This stack gets trained and used as an input generator for the following stack. With this understanding, the authors in [6] defined the encoder portion of their denoising autoencoder using a d-500-500-2000-10 arrangement, where d was the dimension of an input dataset. However, unlike [125], DEC introduced a drop out layers which are inserted in between each *Dense* layer of the elementary denoising autoencoder [6]. With this architectural setup at hand,

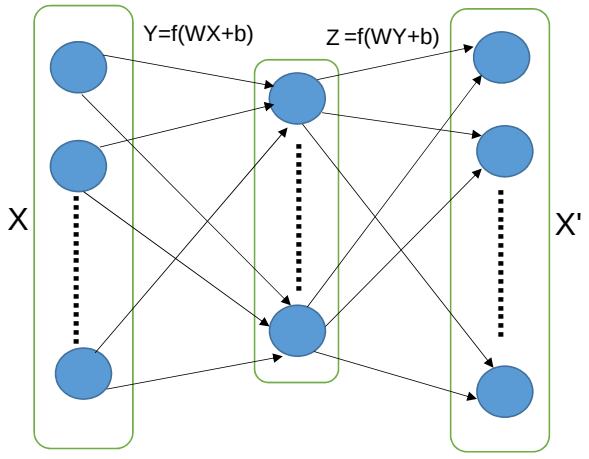


Figure 5.11: Layer arrangements for an elementary denoising autoencoder [125]

the authors first trained the stacked denoising autoencoder using a reconstruction loss (3.3). Following this training, the encoder part of the trained autoencoder got utilized to define a latent space clustering neural network. To upgrade the trained encoder into a clustering network, the authors first performed a standard K-means clustering on the latent features of the autoencoder. Following this, the latent centroids (averages) of the clusters got utilized to compute a soft cluster label assignment for the latent feature. The soft assignments got computed using the student t-distribution as a similarity measurement kernel, i.e., as shown in (5.11). In (5.11), $q_{i,j}$ is the soft assignment (likelihood) of a latent space feature Z_i belonging to cluster j given a cluster's latent centroid μ_j. Moreover, α is the degree of freedom for the t- distribution dependent on the number of series belonging to a cluster. However, since this is not evident before the clustering, [6] proposed to set it to one.

$$q_{i,j} = \frac{(\frac{1+||Z_i-\mu_j)||_{l2}}{\alpha})^{\frac{-\alpha+1}{2}}}{\sum_{j=1}^{K}(\frac{1+||Z_i-\mu_j)||_{l2}}{\alpha})^{\frac{-\alpha+1}{2}}} \tag{5.11}$$

In general, in the end, DEC aimed to learn latent features and centroids that guarantee latent space cluster assignment with a higher degree of confidence. To realize this requirement, the authors proposed to compute the KL divergence between the soft assignment and an auxiliary distribution. In this regard, the authors argued that since $q_{i,j}$ is a soft assignment (probability), they desired an auxiliary distribution that [6];

- Strengthen prediction.
- Put more emphasis on data points assigned higher confidence.
- Normalize loss contribution of each centroid to prevent large clusters from distorting the hidden feature space.

To this end, [6] defined the the auxiliary distribution to be (5.12).

$$p_{i,j} = \frac{\frac{q_{i,j}^2}{\sum_{j=1}^{K} q_{i,j}}}{\sum_{j=1}^{K} \frac{q_{i_j}^2}{\sum_{j=1}^{K} q_{i,j}}} \tag{5.12}$$

Finally, with these two distributions at hand, [6] finally proposed to retrain the encoder (which is now the clustering network) using the KL divergence between the soft probabilities ($p_{i,j}$) and its soft assignment values ($q_{i,j}$); i.e., (5.13).

$$KL(P||Q) = \sum_{i} \sum_{j} p_{i,j} \log \frac{p_{i,j}}{q_{i,j}} \tag{5.13}$$

With this approach, the authors showed that DEC could generate the highest accurate class labels for the MNSIT, STL-HOG, REUTERS-10K, and REUTERS datasets [6]. However, even though DEC was shown to be effective, due to its architectural setup, it can not generate a time domain centroid. To this end, we can not directly utilize DEC in our context. With this in mind, we propose to pair DEC with the multi-tasking setup with our proposed multi-tasking setups. With this said, we present the customization we have made on the DEC arrangement.

5.2.2.1 Deep Embedding Clustering with Time Domain Centroids

In order to enable DEC to generate time domain cluster centroids, we customized it starting from the autoencoder architecture. In this regard, we change the SDAE autoencoder which the DEC was based on with the *Convolutional* autoencoder shown in Figure 5.12.

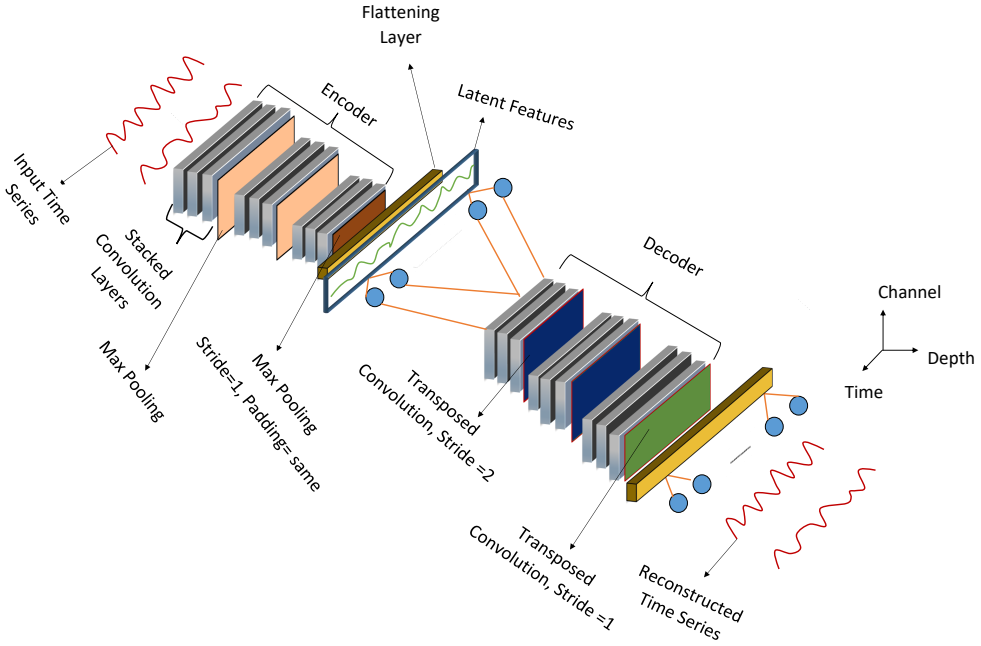


Figure 5.12: Proposed VGG16 Based Autoencoder for Deep Embedding Clustering (DEC)

We have used the VGG16 based autoencoder while augmenting time series averages from the latent

space of autoencoders. We present the archteture here for the sake of clarity. Table 3.11. In practice, in recent years, *Convolutional* layers were shown to be capable of extracting useful latent features without the need of introducing noises [57], [58], [60]. To this end, we believe that by changing the SDAE within the one shown in Figure 5.12, we would not lose significant informatoin [56]. On the contrary, we will be able to reduce the complexity associated with training a SDAE. With this in mind, we first propose to train the proposed autoencoder architecture using the reconstruction loss given in (3.3). After training, we propose to take the encoder portion of the autoencoder for the DEC arrangement. We then re-train the encoder for the KL divergence given in (5.13). In reality, after training the clustering network, we have two possibilities that could enable us to generate time domain centroids. In the first scenario, we can use the trained cluster network to generate the labels for the input datasets. We then can use the labeled datasets to train the multi-tasking autoencoder shown in Figure 5.13.

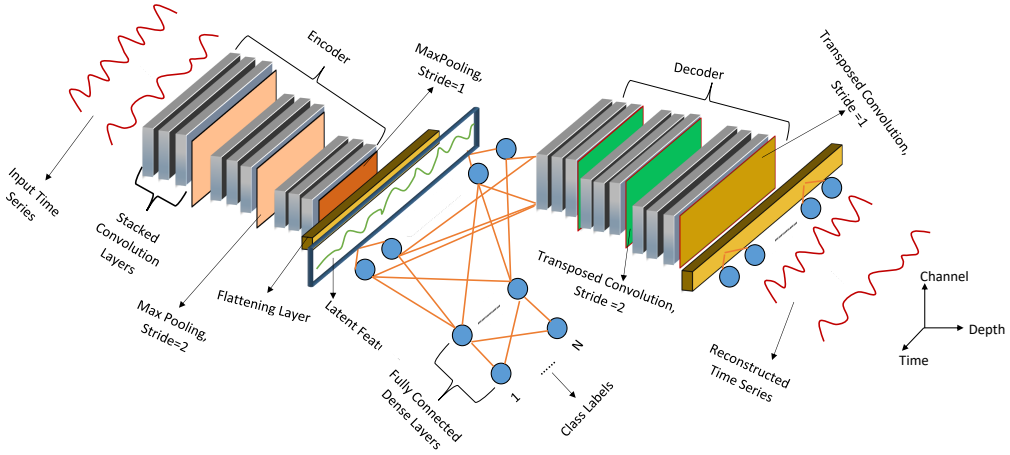


Figure 5.13: Proposed multi-tasking VGG16 based autoencoder that is to be used to generate time domain centroids for clusters identified with DEC

Alternatively, we could also use the DEC trained encoder as the encoder portion of the multi-tasking autoencoder. However, for this configuration, we could freeze the weights of the trained encoder so that the multi-tasking setup could not re-train it. Thus, in this case, we re-train the decoder and classifier on the latent features that the clustering network has previously learned. With these possibilities in mind, we will present the arrangements for the experimental evaluations and the corresponding outcomes.

5.2.2.2 Extended Experimental Setup

In our evaluations, we have kept the cluster numbers to be the same as before, i.e., five clusters. Moreover, for the DBA and basic k-means clustering, we utilize the Tslearn implementation of the algorithms [111]. In general, for both clusterings, we train the basic K-means algorithm for 1000 iterations. However, in DBA K-means, we use an additional 100 iterations that correspond to DBA. On the contrary, for the DEC clustering, we train the basic autoencoder for 1500 epochs and with zero regularization. After this training, we re-trained the encoder under DEC setup using the Python implementation of DEC given in [56]. In practice, we train the DEC setup until its KL divergence falls below

a tolerance value of 10^-3. However, we train the multi-tasking setup for 1500 epochs and zero L2 regularization. Finally, we estimate the per cluster time domain centroids by taking the arithmetic mean of the per-cluster latent features re-projected using the decoder portion of the multi-tasking autoencoder.

In contrary to the clustering, we only utilize a $D - SARIMA\{(2,\ 1,\ 2),\ (2,\ 0,\ 0)_{24},\ (2,\ 0,\ 0)_{168}\}$ which is fitted using segments of the cluster centroids as a forecasting model. We limit the number of forecasting models to one because our current focus is on the impact of the clustering rather than the type of the forecasting model. Moreover, we found the D-SARIMAto be less computationally involving than the hybrid and LSTM based approaches used in [114]. In addition to this change, we have also modified the way we aim to utilize the fitted D-SARIMA model. In this regard, after fitting the model, we aim to take the AR coefficients of the fitted model and replace the centroid segment used for fitting with the corresponding segment of the dataset under observation. Consequently, the MA part of the D-SARIMA model will use the errors of the AR for its computations. Thus, this way, we further intensify our focus on the representativeness of the fitted model. We also perform the same segment substitution for centroids estimated with K-means and DBA K-Means.

As a data pre-processing step, we convert the unit of the traffic data from Tera Byte (GB) to Giga Byte (GB) by dividing them by 1024. We mainly utilize this constant scaling to make the magnitudes of the amplitude values manageable for the neural network setups. Moreover, we also use constant scaling to avoid introducing any sort of amplitude distortion by using readily available normalization techniques. For instance, if we assume the clustered datasets to be vectors in an M dimensional space, normalization techniques such as standard scaling (5.8) are known to confine non-outlier datasets into a very small region [29]. In practice, such distortions could easily become a source of difficulty for clustering algorithms which mainly aim to separate datasets into groups (clusters). Finally, as a benchmark, instead of fitting the D-SARIMA models on cluster centroids, we fitted the D-SARIMA models on the individual cluster members identified by the different clustering algorithms.

5.2.2.3 Experimental Results

We divided our experimental evaluations into four categories. In this aspect, we first evaluate a DEC clustering that gets paired with a multi-tasking network for time domain centroid estimation. However, we set the encoder's weight to be determined by the DEC arrangement. Onwards, we identify this arrangement using the name DEC_MT_Enc_Fixed. For the second evaluation of the DEC arrangement, we utilize the same clustering and time domain centroid estimation approaches. However, in this case, we refrain from freezing the encoder's weights in the multi-tasking setup. Afterward, We identify this setup as DEC_MT. For the remaining two categories, we use DTW based K-means (DBA K-Means), i.e., with DBA centroid estimation, and basic K-means with arithmetic mean as centroids (K-means). With these nomenclatures in mind, we will first present our findings starting from the DEC_MT_Enc_Fixed configuration. In this regard, we first observe the spatial location of the clustered radio nodes. In this aspect, Figure 5.14 demonstrates the geographical location of the radio nodes compared to the map of Addis Ababa. From the figure, we noted that clusters 0 and 1 mostly correspond to residential and mixed-use areas. To be more specific, according to the latitude and

longitude information, we identified cluster zero belongs to relatively sparsely populated sub-cities such as *"Gullele"*, *"Kechene"*, etc. On the contrary, cluster one belonged to the highly populated mixed-use areas which are near the center of the city, for instance, *"Cherkos"*,*"Addis Ketema"*, etc. However, in both cases, the traffic patterns of the cluster centroids force us to speculate that most of the radio nodes for the mentioned clusters to be near residential areas rather than business centers. Contrary to this fact, clusters 2 and 3 correspond to radio nodes near *"Megenagna"* and *"Bole"*. In reality, these locations are mainly business areas. Moreover, the general population within these areas is famous for its higher data traffic demand due to the presence of large entertainment facilities. Finally, cluster 4 corresponded to radio nodes located within the vicinity of *Sidst killo*. In reality, these areas are known to accommodate most of the private and government-owned universities and densely populated residential areas. Moreover, the area also accommodates most of the foreign embassies and key governmental organizations.

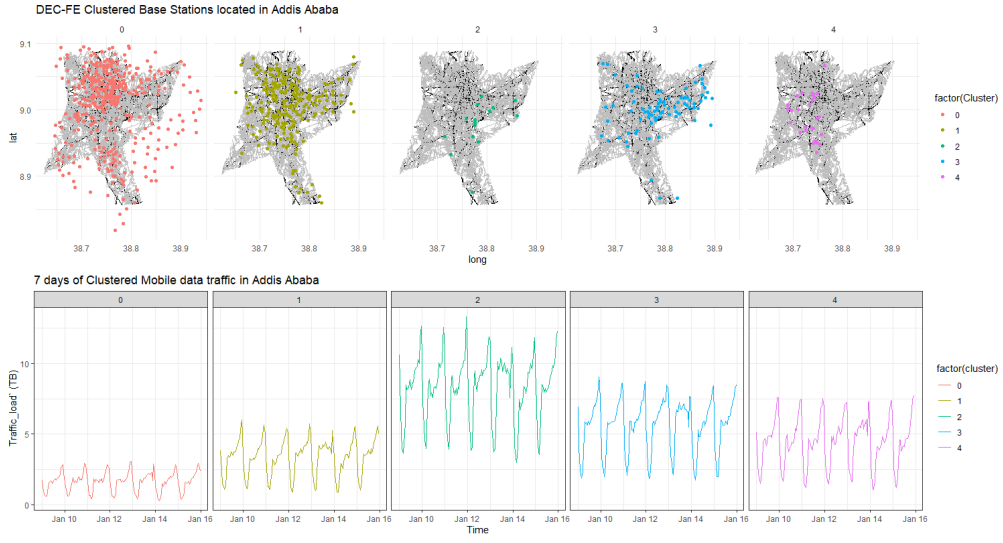


Figure 5.14: Geographical mapping of UMTS radio nodes that are clustered based on their traffic patterns. The clustering was performed using DEC and time domain centroides that are estimated using the multi-tasking autoencoder with a pretrained encoder.

In general, the cluster centroids also showed traffic demand patterns fitting the profile of the geographical locations. Moreover, despite a constant amplitude offset, overall, they showed similar seasonal patterns as shown in Figure 5.14. In Figure 5.14, we plotted the estimated time domain centroids for one week. In practice, such similarities among the cluster centroids are mainly evident due to the demography of the city. In reality, in Addis Ababa, there is no clear demarcation between business and residential areas. To this end, we expect the clusters to share similar traffic patterns as users move in between areas demarcated with the different clusters. With this in mind, we compute the intra-cluster correlation, i.e., as shown in Figure 5.15.

As per our expectation, the first two clusters (0 and 1) showed a higher correlation. Similarly, clusters 2 & 4 and 1 & 4 also show such high correlations. In general, we found the correlation results in line with our initial geographical remarks. Moreover, as stated earlier, we utilize the most correlated cluster centroids as exogenous variables on another as we fit the D-SARIMA model. With

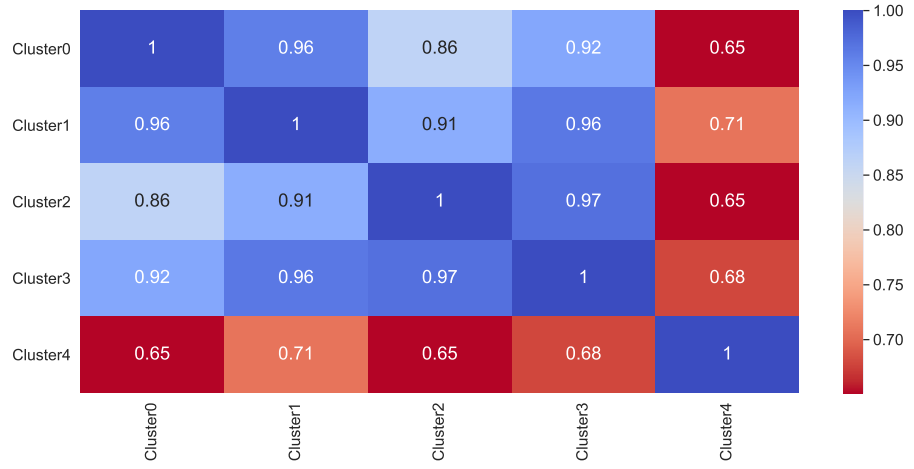


Figure 5.15: Intra cluster correlation among cluster centroides that are estimated with a multi-tasking autoencoer where its encoder was pre-trained with a DEC setup

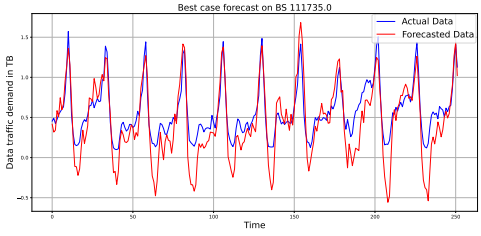
this in mind, we next take a 14 weeks segment of the centroids to fit the D-SARIMA models. We utilize the fitted D-SARIMA models to generate a forecast for the individual cluster members for a duration of $1\frac{1}{2}$ weeks. However, as stated in the experimental setup subsection, we substituted the 14 weeks centroid segment with segments extracted from the individual datasets. On the contrary, we have also fitted the D-SARIMA models on the individual cluster members as a benchmark, where we have not captured the spatial correlation with exogenous variables. In general, Table 5.3 summarizes the aggregate per cluster RMSE and MAE. In the table, the cluster level and the base station level forecasts get differentiated using the keywords CS and BS, i.e., CS-RMSE and BS-RMSE. According to Table 5.3, the cluster-level forecasting model has better captured the overall traffic pattern and generated relatively optimal forecasts. This is mainly due to the inclusion of spatial information through the cluster correlation matrix.

Table 5.3: Aggregate average per-cluster forecasting errors using a D-SARIMA model that is fitted on clusters and centorides defined by DEC and a multi-tasking autoencoder. For this arrangement, we have trained the multi-tasking's encoder using the DEC setup. Thus, wile training the multi-tasking autoencoder we freezed the weights of the encoder.

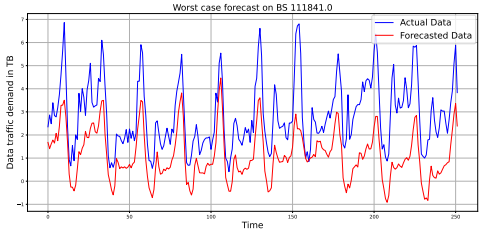
Cluster	CS-RMSE	CS-MAE	BS-RMSE	BS-MAE	# Radio units
Cluster0	0.657	0.502	0.736	0.558	376
Cluster1	1.113	0.824	1.332	1.017	229
Cluster2	1.919	1.419	2.085	1.602	15
Cluster3	1.488	1.161	1.754	1.363	82
Cluster4	2.032	1.694	1.965	1.604	27
Mean	1.442	1.120	1.574	1.229	729

In general, we found this to be quite encouraging since we have significantly reduced the number of required forecasting models while obtaining better forecasting errors, i.e., the benchmark. Finally, to visually demonstrate the performance of the cluster level approach, we have plotted cluster level forecasts for datasets selected from clusters 0 and 4. To generate the plots shown in Figure 5.16,

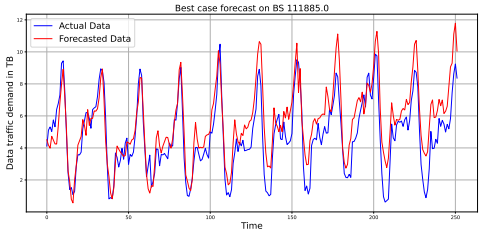
we identified and used the forecasts of the cluster members that gave the minimum and maximum MAE errors. Overall, the figures demonstrate the shape preservation capability of the cluster level forecasting even under worst cases. With these said, we assess the DBA K-Means based approach.



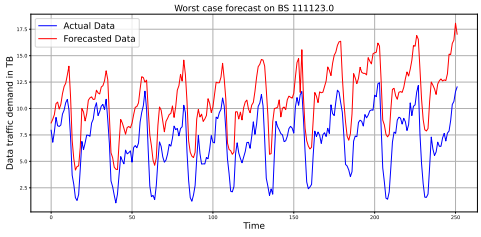
(a) Cluster0 best CS forecast MAE = 0.198



(b) Cluster0 worst CS forecast MAE = 1.734



(c) Cluster4 best CS forecast MAE = 1.049



(d) Cluster4 worst CS forecast MAE = 3.778

Figure 5.16: Visual demonstration of best and worst case forecasts that are based on DEC_MT_Enc_Fixed

Following the same experimental setups we used for the DEC_MT_Enc_Fixed, we first visually analyzed the geographical location of the clusters identified by DBA K-Means. In this regard, Figure 5.17 shows the spatial locations of the clusters as compared to the map of Addis Ababa. Overall, the DBA based K-Means also identified similar geographical areas we mentioned earlier. In general, we noted the following DEC_MT_Enc_Fixed to DBA K-Means cluster correspondences: $(DEC\_MT\_Enc\_Fixed,\ DBA\ K-Means) = \{(Cluster0,\ Cluster3),\ (Cluster1,\ Cluster2),\ (Cluster2,\ Cluster1),\ (Cluster3,\ Cluster0)$ and $(Cluster4, Cluster4)\}$. However, in terms of the centroids, the estimates generated using the DEC_MT_Enc_Fixed setups are relatively smooth. In this aspect, the centroids generated with DBA K-Mans get highly impacted by pathological association. In Figure 5.17, the association is evident with the pointy peaks and constant horizontal slops in the estimated centroids. In practice, the presence of constant offsets among cluster centroids is the main contributor to DBA's pathological associations. In this aspect, in chapter two, Figure 2.4 (b) demonstrated how such constant offsets could influence DTW to identify warping paths that are far from the diagonals of the global cost matrix. In reality, we expect the shape distortion on the centroids to have an impact on the intra-cluster correlation. In this aspect, Figure 5.18 shows that the intra-cluster correlation is relatively low. However, strictly speaking, there can be two reasons why this is so. In this regar, as a first reason, we can point to the shape distortion for the degradation of the intra-cluster correlation. However, the lower intra-cluster correlation could also be associated with the fact that DBA K-Means has identified clusters far from each other. We speculate the second reason is not true since we see no significant spatial difference between the clusters identified with DEC_MT_ Enc_Fixed and DBA K-means besides cluster numbering (order). In general, in theory, we expected the forecasting models based on DBA K-Means to perform poorly since now a centroid used as an exogenous variable is less correlated to

Figure 5.17: UMTS radio nodes clustered based on their traffic patterns using DBA K-means

the input of the D-SARIMA models. However, contrary to our expectation, Table 5.4 the DBA based K-Mean obtained better forecasting errors, i.e., compared to the DEC_MT_Enc_Fixed. Moreover, the DBA K-Means approach also obtained better performances, i.e., compared to the BS level forecasts. In reality, there are different reasons behind the better performance of the DBA K-Means the cluster level forecasts. For instance, the number of cluster members in Table 5.3 is relatively unbalanced as compared to the clusters formed by DBA K-Means. In practice, since we are clustering the same

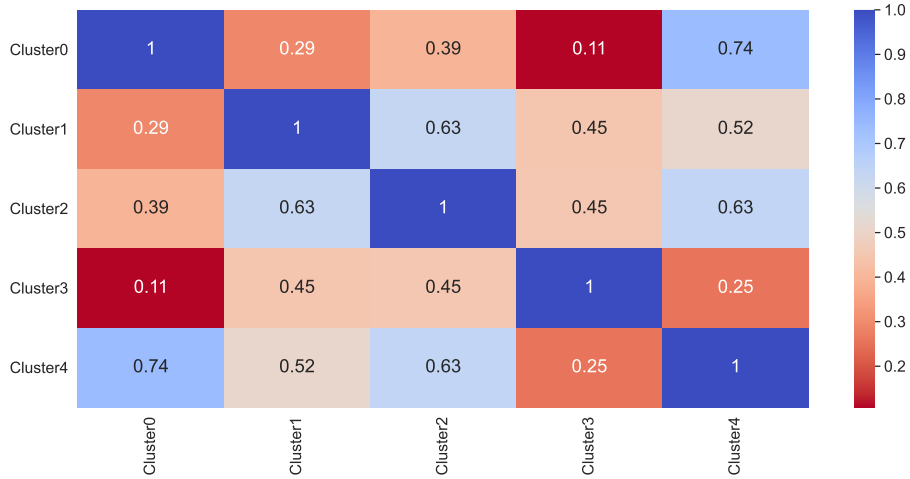


Figure 5.18: Intra cluster correlation between cluster centroides that are estimated with DBA K-Means

datasets, we expect the clusters obtained by both clustering techniques to be relatively comparable. However, since this is not the case, there is a higher likelihood that the DEC setup has grouped a set of extreme cases (outliers) in one of its clusters, for instance, Cluster4. In practice, the centroid of such clusters is often not a good representative of their members. For instance, in the context of the multi-tasking setup, we can not expect the latent space representation of such a cluster to be compact. This is contrary to the underlying assumption behind the multi-tasking setup, i.e., per-class (cluster)

Table 5.4: Aggregate average per-cluster forecasting errors with forecasting models fitted on the centroids of clusters that are defined by DBA K-Means

Cluster	CS-RMSE	CS-MAE	BS-RMSE	BS-MAE	# Radio units
Cluster0	0.714	0.524	0.868	0.664	186
Cluster1	2.129	1.715	2.012	1.565	59
Cluster2	1.079	0.789	1.205	0.919	237
Cluster3	0.416	0.318	0.330	0.239	124
Cluster4	1.468	1.156	1.652	1.288	123
Mean	1.161	0.901	1.214	0.935	729

latent features have a compact latent representation due to their inherent similarity. To this end, we can not expect the multi-tasking setup to optimally re-project the latent estimates. Consequently, we can not also expect the forecasting model fitted on such centroids to perform better.

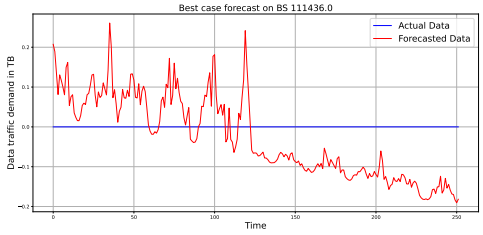
Strictly speaking, we have to note that for the DEC_MT_Enc_Fixed we have constrained the encoder of the multi-tasking network from training itself. However, in reality, the objective functions of the DEC and the multi-tasking autoencoder are different. To this end, we are limiting the multi-tasking network from using its full penitential since one of its key units is now untrainable. To further support our argument, in Table 5.5 we have presented the cluster level forecasting errors obtained when the multi-tasking autoencoder is trained from scratch, i.e., using the same clusters identified with the DEC_MT_Enc_Fixed setup.

Table 5.5: Aggregate average per-cluster forecasting errors obtained while using a D-SARIMA model that is fitted on the centroids of clusters defined by the DEC_MT_ENC_Fixed setup. However, for this evaluation, we estimate the centroides by training a multi-tasking autoenoder from scratch

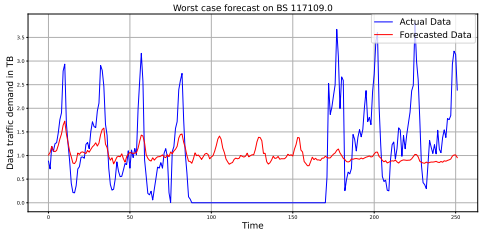
Cluster	CS-RMSE	CS-MAE	# Radio units
Cluster0	0.682	0.528	376
Cluster1	1.232	0.940	229
Cluster2	1.932	1.468	15
Cluster3	1.471	1.158	82
Cluster4	1.346	1.057	27
Mean	1.333	1.030	729

Even though the aggregate forecasting errors are slightly higher than the DBA k-Mean, we have to also note that the DBA K-Means has comparatively lower randomness in terms of outcomes. In this regard, a key contribution to the outcome randomness in DBA K-Means is the random initialization of the centroids. However, in practice, the repeated iterations of DBA often smooth out this randomness. For instance, we have executed the DBA K-Means using two separate repeated trials. Even though the trials took us two days to complete, we found the variation in the outcomes to be insignificant. Thus, we finally took the centroids that obtained an 80.933% and 100% Euclidean and DTW distance based NCC accuracy. In this context, the DEC-based approach has a slightly higher source of randomness. For instance, the weight idealizations in the autoencoder, the multi-tasking average estimation process,

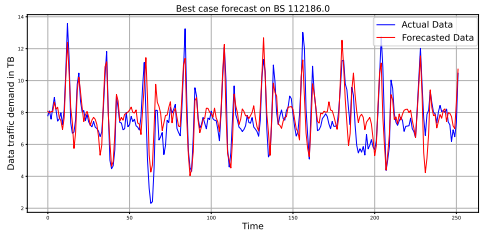
and the intermediate latent space K-Means clustering. To this end, for the DEC approach, we expect a slightly higher variation in the outcomes of repeated trials. With this in mind, we executed the DEC approach for an additional six repeated trials which took two days to complete. Contrary to our previous approach, in these trials, we have refrained from constraining the encoder portion of the multi-tasking autoencoder and trained it from scratch. However, before going into the details of the best performing outcome, we will conclude the discussion of DBA K-Means forecasting by presenting the visual demonstration of the worst and best case forecasts.



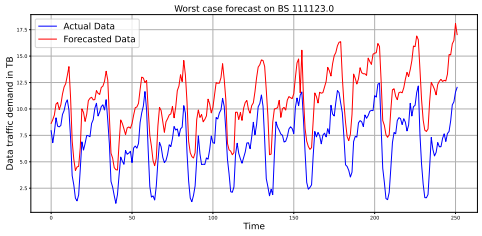
(a) Cluster3 best BS forecast MAE = 0.096



(b) Cluster3 worst BS forecast MAE = 0.767



(c) Cluster1 best BS forecast MAE = 0.744



(d) Cluster4 worst BS forecast MAE = 5.237

Figure 5.19: Visual demonstration of best and worst forecasts for a D-SARIMA model fitted with centroieds estimated using DBA

In the DEC_MT_ENC_Fixed experiment, we have selected the time domain centroids that scored 85.33% and 93.83% NCC accuracies while using DTW and euclidean distances. Moreover, in the latent space, they obtained a 97.26% NCC accuracy while using euclidean distance. In this aspect, for the second evaluation of the DEC arrangement (DEC_MT setups), we have selected the cluster formation that obtained a time domain DTW and euclidean distance NCC accuracies of 75% and 92.45%. On the other hand, the selected setup obtained a latent space euclidean distance NCC accuracy of 93.55%. One interesting point we note here is that, due to the presence of trend (an increasing DC offset), DTW distance appears to be performing poorly. With this in mind, we then observed the geographical location of the newly formed clusters using Figure 5.20.

In general, the new DEC clusters mostly overlap with the clusters identified by DBA K-Means. In this context, we can form the following cluster correspondences: $(DBAK-Means, DEC\_MT) = \{(Cluster0, Cluster0), (Cluster1, Cluster1), (Cluster2, Cluster3), (Cluster3, Cluster4), (Cluster4, Cluster2)\}$. However, in the context of the cluster centroids, the multi-tasking autoencoder setup approach generated relatively smoother estimates. Moreover, as in the case of DEC_MT_Enc_Fixed, the estimated centroids mostly differ by a constant offset. To this end, we expect the intra-cluster correlation to be relatively high. This expectation gets validated with the heat map of the intra-cluster correlation shown in Figure 5.21. However, contrary to DEC_MT_ENC_Fixed, we now have balanced

Figure 5.20: UMTS radio nodes clustered based on their traffic patterns using the DEC_MT arrangement.The clustering was performed using DEC and the time domain centroides are estimated by training a multi-tasking neural network from scratch

clusters. This, in turn, is reflected in the performance of the forecasting models which are better than the DBA K-mean arrangement as shown in Table 5.6. In general, the improved cluster formation has validated our argument that DEC_MT_ENC_Fixed clustered outliers. Moreover, the performance improvement also shows the ability of the DEC to capture and group similar patterns more efficiently. Moreover, it also further shows the ability of the multi-tasking setup to extract representative centroids without the need of constraining its encoder. With this said, we will conclude the discussion for DEC_MT by presenting the visual demonstrations of the worst and best case forecasts. For the demonstration, we have selected samples from Cluster3 and Cluster2. Generally, the lower traffic demand forecasts based on DEC_MT setup are better than forecasts based on DBA K-means forecasts. One possible contributor in this regard is the ability of the multi-tasking autoencoder to generate relatively smoother cluster centroids.

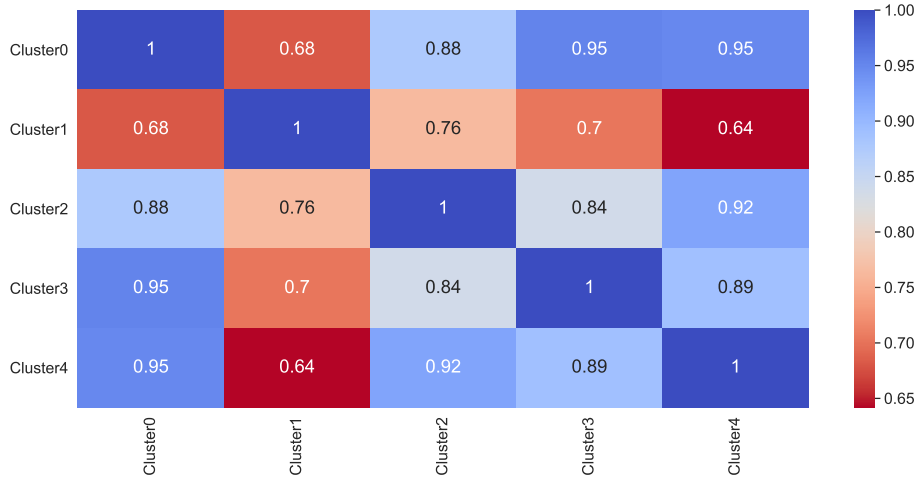
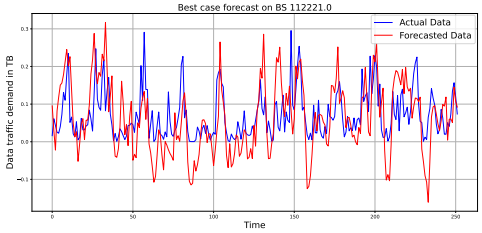
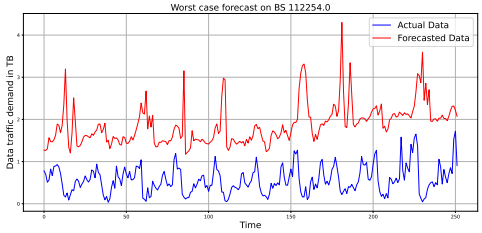


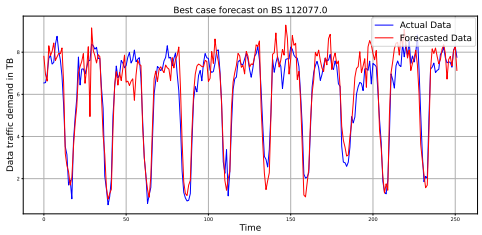
Figure 5.21: Intra cluster correlation between cluster centroides that are estimated with multi-tasking autoencoder that is trained from scratch



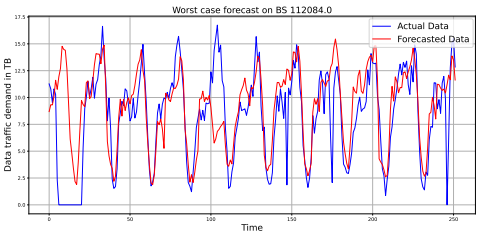
(a) Cluster3 best CS forecast MAE = 0.062



(b) Cluster3 worst CS forecast MAE = 1.309



(c) Cluster4 best CS forecast MAE = 0.595



(d) Cluster4 worst CS forecast MAE = 2.314

Figure 5.22: Visual demonstration of best and worst forecasts for a D-SARIMA model fitted with centroieds estimated using a multi-tasking autoencoder trained from scratch

Table 5.6: Aggregate average per-cluster forecasting errors with the forecasting model fitted on the centroids of clusters defined by a multi-tasking autoencoder

Cluster	CS-RMSE	CS-MAE	BS-RMSE	BS-MAE	# Radio units
Cluster0	1.104	0.816	1.417	1.092	164
Cluster1	1.050	0.778	1.205	0.932	88
Cluster2	0.409	0.305	0.426	0.308	165
Cluster3	1.524	1.145	1.867	1.458	105
Cluster4	0.863	0.671	0.992	0.758	207
Mean	0.990	0.743	1.181	0.909	729

Finally, we will conclude this chapter by presenting the experimental outcomes of the K-Means approach. In this regard, we also start our assessment of the basic K-Means by observing the geographical location of the clusters. In this aspect, Figure 5.23 demonstrates the location of the clusters compared to the map of Addis Ababa. In terms of their spatial location, we find the clusters identified by DBA k-Means and K-Means to be highly similar. Thus, the cluster correspondence for this case becomes: $(DBAK-Means,\ K-Means) = \{(Cluster0,\ Cluster0),\ (Cluster1,\ Cluster4),\ (Cluster2,\ Cluster1),(Cluster3,\ Cluster3),\ (Cluster4,\ Cluster3)\}$. However, comparatively, the centroids identified by the basic K-means are relatively smoother than those identified with DBA K-Means. In this regard, we identified two key contributing factors. First, as we stated earlier, the data traffic has an increasing trend. To this end, the trend introduces constant offsets that are problematic in the context of DTW warping. Additionally, we observed that euclidean distance was obtaining better accuracies on the NCC we conducted while selecting better performing centroids and cluster formations. Thus, this implies that for the datasets, the impact of temporal distortion is minimal to cause significant distortion on arithmetic means. Thus, in the context of the smoothness, the situation favored arithmetic means rather than their DBA counterparts. With these observations in mind, we next conduct the

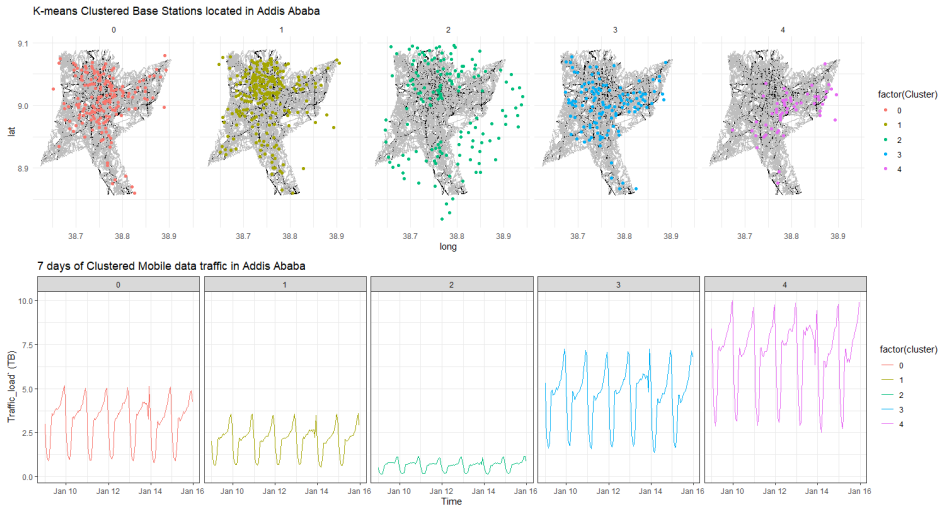


Figure 5.23: UMTS radio nodes clustered based on their traffic patterns using K-Means

intra-cluster correlation analysis to identify the exogenous variables of the D-SARIMA models. In this aspect, the correlation heat map of the K-Mean highly resembles the maps shown in Figures 5.15 and 5.21. We find this to be logical given the circumstances and nature of the datasets. In general, based on the results shown in Table 5.7, the forecasting errors for the K-Means approach stand third, i.e., the forecasting errors are lower than DEC_MT_Enc_Fixed but higher than DEC_MT. However, in this case, there is almost no difference between CS and BS level forecasting. This further validates our initial concern about the inability of arithmetic mean to capture descriptive per-cluster features even under favorable conditions.

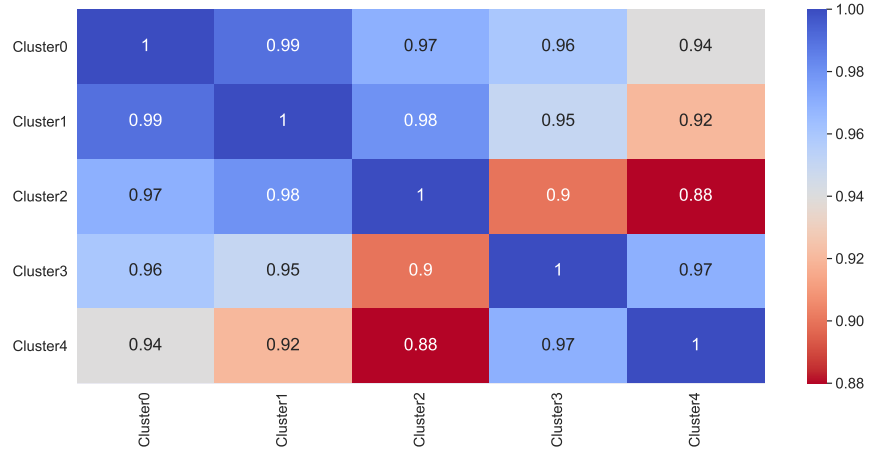


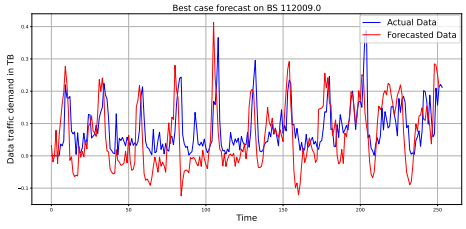
Figure 5.24: Intra cluster correlation between cluster centroides that were estimated with basic K-means

With these said we will finalize the analysis of experimental outcomes by presenting the plots of the best and worst case forecasts as shown in Figure 5.25. For the plots, we followed the same approach used so far and selected samples from clusters that have worst and best aggregate average per-cluster forecasting errors; i.e., in this case, Cluster2 and Cluster4.In conclusion, in this chapter, we argued that in wireless communication networks radio nodes are expected to exhibit spatial correlation. Thus, to capture the spatial correlation between the nodes through clustering. In reality, we proposed to cluster

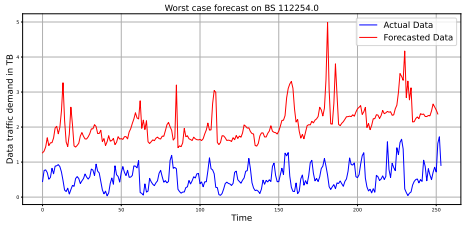
Table 5.7: Aggregate average per-cluster forecasting errors with the forecasting model fitted on the centroids of clusters defined using a basic K-Means

Cluster	CS-RMSE	CS-MAE	BS-RMSE	BS-MAE	# Radio units
Cluster0	1.131	0.859	1.263	0.968	171
Cluster1	0.938	0.739	0.961	0.735	236
Cluster2	0.389	0.293	0.388	0.282	146
Cluster3	1.774	1.452	1.664	1.289	122
Cluster4	1.906	1.522	1.993	1.562	54
Mean	1.228	0.973	1.254	0.967	729

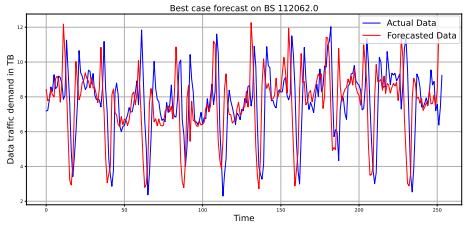
the radio nodes for two main reasons. First, we expect the clusters to identify radio nodes offering similar traffic patterns to be grouped. This grouping, in turn, helped us to summarize the overall traffic patterns through their centroids which in turn significantly reduced the number of forecasting models. Additionally, by identifying the most correlated cluster centroids, we are able to incorporate the spatial



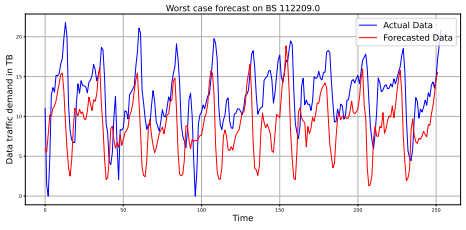
(a) Cluster2 best CS forecast MAE = 0.068



(b) Cluster2 worst CS forecast MAE = 1.626



(c) Cluster4 best CS forecast MAE = 0.907



(d) Cluster4 worst CS forecast MAE = 4.140

Figure 5.25: Visual demonstration of best and worst forecasts for a D-SARIMA model fitted with centroids estimated using a a basic K-Means

correlation among the radio nodes. An alternative solution in this regard would have been to divide the geographical location associated with the radio nodes into grids. We then could have associated the traffic pattern of a grid with the traffic pattern of a radio node belonging to the grid. However, the outcome of this approach could get clouded by many challenges. For instance, determining the appropriate size of a grid would have been one problem. This is because the coverage area of radio nodes is often dependent on network parameters. Thus, there is a possibility that multiple radio nodes could fall within a grid. Moreover, even if we manage to define an appropriate grid size, defining a propagation model that governs the spatial correlation would have been relatively challenging. In this aspect, the cluster-level approach provided a relatively easy way of capturing the spatial correlation among radio nodes evident due to land use. In this regard, we find the approach based on representation learning comparatively efficient and useful compared to evaluated alternatives.

6 Summary, Conclusions & Outlook

In this dissertation, we showed the challenges associated with the estimation of time series averages. To elaborate on the challenges, we first identified the requirements associated with estimating an *"optimal"* time series average. In reality, even though we found the concept of *"optimality"* to be rather ambiguous, we noted that researchers often agree on at least two key terms. First, an average time series get expected to preserve descriptive shapes observed in the averaged set. Moreover, an average get expected to minimize its discrepancy with members of the averaged set. To meet these requirements, currently available averaging heuristics mainly relied on utilizing different kinds of alignment techniques, for instance, DTW, correlation, and velocity fields. Overall, the techniques mainly emphasized utilizing the alignment algorithms to register averaged series to their arithmetic means in a space that is possibly different from the time domain. For instance, DTW based techniques warp the averaged series into DTW space. On the contrary, velocity field-based approaches morphed the averaged series through controlled re-sampling. In general, despite the difference in the alignment techniques, pioneering techniques tried to address time series averaging as an alignment problem. Consequently, the averaging problem often gets clouded with the challenges associated with the difficulty of simultaneously performing multiple alignments.

With these understandings, in this dissertation, we proposed to approach time series averaging as a generative problem through deep representation learning. Overall, we aimed to estimate time series averages from their latent space representations. With this objective in mind, we performed rigorous assessments of the latent space embedding of neural network architectures presumed to be either semi or fully generative. In general, we argued that given appropriate neural network architectures and accompanying objective functions, we could mimic the effects of multiple alignments through their latent embedding. To validate this argument, we first proposed to assess the latent space of autoencoders performing a basic encoding and decoding operation. In reality, we choose autoencoders as our optimization setup for two reasons. First, autoencoders can reconstruct latent embedding through their decoders. Thus, it provided a way to generate time domain equivalents for means estimated in the latent space. Additionally, in practice, we found autoencoders deployed as one of the building blocks of most generative neural network architectures. This, in turn, aligned with our objective of approaching time series averaging as a generative problem. Generally, our assessments of the basic and variational versions of the autoencoders revealed their latent embedding is sufficient to generate time domain estimates better than a time domain arithmetic mean. Overall, we attribute this performance improvement to the filtering action of the proposed network architecture. In reality, we built the autoencoder arrangements from *Convolutional* layers. In practice, *Convolutional* layers are known to be good at analyzing shapes irrespective of their locations. To better understand the advantage of this concept, we can consider concepts in the signal analysis as illustrative examples.

In practice, we can at least observe signals in two different domains, i.e., time and frequency domains. Moreover, in most cases, certain behavior of signals presumed relatively complex in the time domain can be relatively simple to interpret in the frequency domain. For instance, if we consider a group of band-limited signals, for instance, smooth signals that are free of sharp edges but shifted versions of each other as an example. In the frequency domain, the signals will get mapped to similar spectral components. Consequently, in the frequency domain, it would be easier to identify the signals that share a common ground. Using this analogy, we can assume the filters of *Convolutional* layers within the proposed autoencoders as processing components focusing on identifying shapes despite their time shift. In reality, this remark is in line with our observation of the latent space projection of the input series and the associated latent space NCC accuracies. In this regard, the comparatively denser latent space t-SNE projections implied that the *Convolutional* filters were able to identify descriptive features that are common among input series that share common ground, for instance, similar classes. In another analogy, we can also think of the learned latent representations as if they were the most dominant N principal components, where in this case, the components get learned through a complex nonlinear transformation. Consequently, we expect the latent embedding of the input series to appear to have dense lower dimensional representations when transformed with t-SNE or other dimensional reduction techniques.

Overall, we found the estimates of the autoencoders to be encouraging given we have assessed their quality in DTW space. This is because, while performing the NCC, we have used DTW distance to measure the discrepancy between class members and their respective class averages. However, at this point, we have to note two critical questions. First, given we are using DTW distance, will a time domain estimate that gets generated from a latent space be comparable to an estimated generated with DTW based techniques such as DBA? If the answer to this question is no, the second question would be, what alternative metric would have been better to assess the quality of the time domain estimates generated by the different approaches? In reality, we found the answers to both questions relatively challenging. This is because, in the context ofDTW distance, averages generated using DTW will get favored. This is because, while generating DTW based estimates, we register the training set to the estimates in DTW space. To this end, when we perform a NCC on a test set, i.e., using DTW distance, we transform the class averages and members of the test set to a space that is favorable to the estimates. However, this is not true for estimates generated using the autoencoders that have no information about DTW space, i.e., neither at estimation nor at test time. If this is the case, the next logical question would be, why did we choose to utilize NCC as an evaluation technique? In this regard, we have noted the possibility of utilizing WGSS as an alternative. However, we also noted that the WGSS loss function could get dependent on an underlying registration technique through the distance function (d). This is at least true for the cases of estimates generated with DBA and SDBA. To this end, for such an evaluation technique, the overall comparison would still favor the averaging techniques based on DTW. However, even under such biased comparison, we consider the estimates of the autoencoders to be encouraging for two reasons. First, if we account for the bias and consider only the NCC accuracies obtained in the registered space of the different averaging techniques; NCC accuracies obtained in the latent space of the autoencoders were better than the state-of-the-art.

Additionally, we also have to note that in the learning process, the autoencoders were not given any additional information, unlike their counterparts. For instance, DTAN utilizes class information while performing the diffeomorphic transformation. Moreover, to generate per-class averages, the estimates of DBA and SDBA got generated on a per-class basis that indirectly introduced class information into the averaging process. In this regard, for the autoencoders, there are no such supervisions at the time of training. On the contrary, we only used the class label information when generating the per-class averages after completing the training process. Due to these observations, we strongly believe the potential of the autoencoder based approaches can not get ignored. Moreover, their unsupervised nature could also get used if the need arises. However, if there are ways that class information could get incorporated into the estimation process, i.e., as in the case of the forecasting problem presented in chapter five, we propose the multi-tasking approaches. Specifically, we suggest the deployment of a quantile regression-based multi-tasking autoencoder since they better mimicked the effect of multiple alignments in the latent space.

In conclusion, we have made the basic and the multi-tasking autoencoder approaches as generic as possible. In this regard, we have not utilized any input pre-processing, domain knowledge, and input or task-specific layer organization. However, we also note that recently task-specific neural network architectures are being proposed for temporal datasets. For instance, recently, an architecture named the Inception Time was proposed. In reality, it was able to outperform an ensemble of classifiers that were mainly using DTW distance. In this context, in this dissertation, we have shown the implication of layer arrangement by making successive improvements to the proposed network architectures. Thus, motivated by these observations, we believe the customization of such task-specific architectures to the proposed averaging techniques could further improve the quality of the latent embeddings and time domain estimates generated from them. In addition to this, due to time and computational resource limitations, we only assessed a limited number of objective functions in their original format. In this regard, we believe there is a range of additional objective functions that could contribute positively to the overall learning process. For instance, for the variational and nonvariational autoencoders, we could have forced the architectures to reconstruct a noisy version of their inputs. We believe that such requirements could help the networks to further focus on the most descriptive features that guarantee lower reconstruction error. This, in turn, could assist the separability and compactness of the latent embeddings. This is because for inputs that share common backgrounds, for instance, similar classes, we expect the most descriptive latent features (principal components) to be relatively similar. In general, we believe that the possibility of further upgrades is not limited to this scope. On the contrary, there is a range of better feature extraction techniques either under study or known to the general public. To this end, we next focus on customizing such proposals to better suit the demands of time series averaging.

Bibliography

[1] W. William W.S., **Time Series Analysis: Univariate and Multivariate Methods**. Pearson Addison Wesley, 2006 (see pages 1, 166, 168, 169).

[2] C. Yanping, E. Keogh, H. Bing, B. Nurjahan, B. Anthony, M. Abdullah, and B. Gustavo, *The ucr time series classification archive*, www.cs.ucr.edu/~eamonn/time_series_data/, Jul. 2015 (see pages 1, 2, 22, 47, 58–61, 63, 64, 66–68).

[3] R. Simoes, G. Camara, G. Queiroz, F. Souza, P. R. Andrade, L. Santos, A. Carvalho, and K. Ferreira, **Satellite image time series analysis for big earth observation data**, *Remote Sensing*, vol. 13:no. 13, 2021 (see page 1).

[4] L. Jason, **Time series classification through transformation and ensembles**, PhD dissertation, University of East Anglia, 2015 (see pages 1–3, 59, 64).

[5] L. Parmentier, O. Nicol, L. Jourdan, and M.-E. Kessaci, **Autotsc: Optimization algorithm to automatically solve the time series classification problem**, in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov. 1-3, 2021, 412–419 (see page 1).

[6] J. Xie, R. Girshick, and A. Farhadi, **Unsupervised deep embedding for clustering analysis**, in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, New York, NY, USA, Jun. 19-24, 2016, 478–487 (see pages 2–4, 55, 70, 176–179).

[7] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, **Deep learning for time series classification: A review**, *Data Mining and Knowledge Discovery*, vol. 33:no. 4, 2019, 917–963 (see pages 2, 28–30, 48, 58, 69, 77).

[8] V. S. Siyou Fotso, E. Mephu Nguifo, and P. Vaslin, **Frobenius correlation based u-shapelets discovery for time series clustering**, *Pattern Recognition*, vol. 103, 2020, 263–301 (see page 2).

[9] J. Lin and Y. Li, **Finding structural similarity in time series data using bag-of-patterns representation**, in *International conference on scientific and statistical database management*, New Orleans,LA, USA, Jun. 2-4, 2009, 461–477 (see page 2).

[10] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, **TS-CHIEF: A scalable and accurate forest algorithm for time series classification**, *Data Mining and Knowledge Discovery*, vol. 34, 2020, 742–775 (see page 2).

[11] J. Lin, E. Keogh, L. Wei, and S. Lonardi, **Experiencing sax: A novel symbolic representation of time series**, *Data Mining and knowledge discovery*, vol. 15:no. 2, 2007, 107–144 (see page 2).

[12] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, **The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances**, *Data Mining and Knowledge Discovery*, vol. 31:no. 3, 2017, 606–660 (see pages 2, 3).

[13] A. Bagnall, L. Davis, J. Hills, and J. Lines, **Transformation based ensembles for time series classification**, in *Proceedings of the 2012 SIAM international conference on data mining*, California, USA, Apr. 26-28, 2012, 307–318 (see pages 2–4, 58, 68).

[14] G. Lalit, M. Dennis L, T. Ravi, and S. Panagiotis G, **Nonlinear alignment and averaging for estimating the evoked potential**, *IEEE transactions on biomedical engineering*, vol. 43:no. 4, 1996, 348–356 (see pages 2, 4–6, 11, 21, 22, 47, 56).

[15] N. Vit and R. Chotirat Ann, **Shape averaging under time warping**, in *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, Pattaya, Chonburi, Thailand, May 6-9, 2009, 626–629 (see pages 2, 4–6, 11, 13, 21, 23–25, 47, 56).

[16] F. Petitjean and P. Gançarski, **Summarizing a set of time series by averaging: From steiner sequence to compact multiple alignment**, *Theoretical Computer Science*, vol. 414:no. 1, 2012, 76–91 (see pages 2, 4–6, 13, 14, 21–23, 25, 26, 47, 56, 57, 75, 76, 109).

[17] T. Nguyen, N. Meger, C. Rigotti, C. Pothier, N. Gourmelen, and E. Trouve, **A pattern-based mining system for exploring displacement field time series**, in *2019 International Conference on Data Mining Workshops (ICDMW)*, Beijing, China, Nov. 8-9, 2019, 1110–1113 (see page 2).

[18] A. Bagnall and J. Lines, **An experimental evaluation of nearest neighbour time series classification**, *arXiv preprint arXiv:1406.4757*, 2014 (see pages 3, 47).

[19] P. François, G. Forestier, G. Webb, A. Nicholson E, Y. Chen, and E. Keogh, **Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm**, *Knowledge and Information Systems*, vol. 47:no. 1, 2016, 1–26 (see pages 3, 26, 176).

[20] R. A. Shapira Weber, M. Eyal, N. Skafte, O. Shriki, and O. Freifeld, "Diffeomorphic temporal alignment nets: Supplementary material," in *Advances in Neural Information Processing Systems 32*, Vancouver, BC, Canada, Dec. 8-14, 2019, 6574–6585 (see pages 3, 6, 19, 27, 46, 75, 76, 109).

[21] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, **Time-series clustering–a decade review**, *Information Systems*, vol. 53, 2015, 16–38 (see pages 3, 4, 11, 176).

[22] B. Hans-Hermann, **Origins and extensions of the k-means algorithm in cluster analysis.** *Journal Électronique d'Histoire des Probabilités et de la Statistique [electronic only]*, vol. 4, 2008 (see pages 3, 4, 171).

[23] P. François, A. Ketterlin, and P. Gançarski, **A global averaging method for dynamic time warping, with applications to clustering**, *Pattern Recognition*, vol. 44:no. 3, 2011, 678–693 (see pages 3–5, 11, 26, 38).

[24] T. M. L. Wigley, K. R. Briffa, and P. D. Jones, **On the average value of correlated time series, with applications in dendroclimatology and hydrometeorology**, *Journal of Applied Meteorology and Climatology*, vol. 23:no. 2, 1984, 201–213 (see page 3).

[25] R. Shapira Weber, M. Eyal, N. Skafte, O. Shriki, and O. Freifeld, "Diffeomorphic temporal alignment nets," in *Advances in Neural Information Processing Systems 32*, Vancouver, BC, Canada, Dec. 8-14, 2019, 6574–6585 (see pages 4, 5, 21, 43, 44, 46, 47, 56, 57, 76, 79, 80, 109, 113).

[26] M. Tadayon and Y. Iwashita, **A clustering approach to time series forecasting using neural networks: A comparative study on distance-based vs. feature-based clustering methods**, *CoRR*, vol. abs/2001.09547, 2020. [Online]. Available: https://arxiv.org/abs/2001.09547 (see page 4).

[27] E. J. Keogh and M. J. Pazzani, **Derivative dynamic time warping**, in *Proceedings of the 1st SIAM International Conference on Data Mining (SDM)*, Chicago, IL, USA, Apr. 5-7, 2001, 1–11 (see pages 4, 16).

[28] S. Soheily-Khah, A. Douzal-Chouakria, and E. Gaussier, **Progressive and iterative approaches for time series averaging**, in *Proceedings of the 1st International Conference on Advanced Analytics and Learning on Temporal Data*, Valencia, Spain, Jul. 6-7, 2015, 111–117 (see pages 4, 5).

[29] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, and V. Zocca, **Python Deep Learning: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow**, 2nd. 2019 (see pages 4, 27–33, 35, 37–39, 48, 50, 53, 69, 170, 181).

[30] M. Morel, C. Achard, R. Kulpa, and S. Dubuisson, **Time-series averaging using constrained dynamic time warping with tolerance**, *Pattern Recognition*, vol. 74, 2018, 77–89 (see pages 4, 5).

[31] D. Schultz and B. Jain, **Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces**, *Pattern Recognition*, vol. 74, 2018, 340–358 (see pages 5, 6, 18, 19, 21, 26, 27).

[32] S. Hiroaki and C. Seibi, **Dynamic programming algorithm optimization for spoken word recognition**, *IEEE transactions on acoustics, speech, and signal processing*, vol. 26:no. 1, 1978, 43–49 (see pages 5, 9, 11, 17, 20, 21).

[33] P. John and G. Luis, **K-shape: Efficient and accurate clustering of time series**, in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, Melbourne, Victoria, Australia, 31 May–4 June ,2015, 1855–1870 (see pages 5, 21, 43).

[34] D. Schultz and B. Jain, **Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces**, *Pattern Recognition*, vol. 74, 2018, 340–358 (see pages 6, 9, 13, 18).

[35] M. Brill, T. Fluschnik, V. Froese, B. Jain, R. Niedermeier, and D. Schultz, **Exact mean computation in dynamic time warping spaces**, *Data Mining and Knowledge Discovery*, vol. 33:no. 1, 2019, 252–291 (see pages 6, 9).

[36] B. J. Jain, V. Froese, and D. Schultz, **An average-compress algorithm for the sample mean problem under dynamic time warping**, *CoRR*, vol. abs/1909.13541, 2019. [Online]. Available: http://arxiv.org/abs/1909.13541 (see pages 6, 8).

[37] M. Cuturi and M. Blondel, **Soft-dtw: A differentiable loss function for time-series**, in *Proceedings of the 34th International Conference on Machine Learning*, Sydney, NSW, Australia, Aug. 6-11, 2017, 894–903 (see pages 6, 11, 13, 16–18).

[38] D. P. Kingma and M. Welling, **Auto-encoding variational bayes**, in *2nd International Conference on Learning Representations, ICLR*, Banff, AB, Canada, Apr. 14-16, 2014 (see pages 6, 57, 58, 83, 90, 169).

[39] G. Ian J., P.-A. Jean, M. Mehdi, X. Bing, W.-F. David, O. Sherjil, C. Aaron, and B. Yoshua, **Generative adversarial nets**, in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, Montreal, Canada, Dec. 8-13, 2014, 2672–2680 (see pages 6, 57, 58).

[40] B. K. Iwana and S. Uchida, **An empirical survey of data augmentation for time series classification with neural networks**, *PLOS ONE*, vol. 16:no. 7, 2021, 1–32 (see pages 6, 57, 58).

[41] J. Zhao and L. Itti, **Shapedtw: Shape dynamic time warping**, *Pattern Recognition*, vol. 74, 2018, 171–184 (see pages 12, 16, 20, 55).

[42] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, **Weighted dynamic time warping for time series classification**, *Pattern Recognition*, vol. 44:no. 9, Computer Analysis of Images and Patterns, 2231–2240 (see pages 12–16).

[43] E. Vidal Ruiz, F. Casacuberta Nolla, and H. Rulot Segovia, **Is the dtw "distance" really a metric? an algorithm reducing the number of dtw comparisons in isolated word recognition**, *Speech Communication*, vol. 4:no. 4, 1985, 333–344 (see page 13).

[44] S. Salvador and P. Chan, **Toward accurate dynamic time warping in linear time and space**, *Intelligent Data Analysis*, vol. 11:no. 5, 2007, 561–580 (see pages 13, 20, 21).

[45] B. Lathi, **Signal Processing and Linear Systems**, 2nd. pearson, 2014 (see pages 14, 31).

[46] M. Cuturi, J.-P. Vert, O. Birkenes, and T. Matsui, **A kernel for time series based on global alignments**, in *IEEE International Conference on Acoustics, Speech, and Signal Processing*. Honolulu, HI, USA, May 15-20, 2007, II-413-II–416 (see page 17).

[47] C. Nello and R. Elisa, 928–932, in, *Encyclopedia of Algorithms*, Boston, MA: Springer US, 2008 (see page 17).

[48] H. Shimodaira, K.-i. Noma, M. Nakai, and S. Sagayama, **Dynamic time-alignment kernel in support vector machine**, in *Advances in Neural Information Processing Systems*, vol. 14, Vancouver, British Columbia, Canada, Dec. 3-8, 2001 (see page 17).

[49] F. Itakura, **Minimum prediction residual principle applied to speech recognition**, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23:no. 1, 1975, 67–72 (see pages 20, 21).

[50] B. J. Jain, V. Froese, and D. Schultz, **An average-compress algorithm for the sample mean problem under dynamic time warping**, *CoRR*, vol. abs/1909.13541, 2019. [Online]. Available: http://arxiv.org/abs/1909.13541 (see pages 21, 26, 27).

[51] L. Kaufman and P. J. Rousseeuw, **Finding Groups in Data: An Introduction to Cluster Analysis**. John Wiley, 1990 (see page 23).

[52] B. J. Jain and D. Schultz, **A reduction theorem for the sample mean in dynamic time warping spaces**, *ArXiv*, vol. abs/1610.04460, 2016 (see page 26).

[53] T. Tsegamlak, M. Devanne, J. Weber, H. Dereje, and G. Forestier, **Time series averaging using multi-tasking autoencoder**, in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, Baltimore, MD, USA, Nov. 9-11, 2020, 1065–1072 (see page 27).

[54] B. Yegnanarayana, **Artificial Neural Networks**. Prentice-Hall of India Private Limited, 2005 (see pages 27–29, 35, 38).

[55] J. Xie, R. Girshick, and A. Farhadi, **Unsupervised deep embedding for clustering analysis**, in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, vol. 48, NY, USA, Jun. 20-22, 2016, 478–487 (see pages 30, 48, 177).

[56] B. Lafabregue, J. Weber, P. Gançarski, and G. Forestier, **End-to-end deep representation learning for time series clustering: A comparative study**, *Data Mining and Knowledge Discovery*, 2021, 1–53 (see pages 30, 55, 58, 70, 177, 180).

[57] K. Simonyan and A. Zisserman, **Very deep convolutional networks for large-scale image recognition**, in *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, CA, USA, May 7-9, 2015 (see pages 31, 32, 39, 48–51, 55, 72, 180).

[58] H. Kaim, Z. Xiangyu, R. Shaoqing, and S. Jian, **Deep residual learning for image recognition**, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, Jun. 27-30, 2016 (see pages 31, 32, 39, 48, 50–52, 55, 83, 86, 144, 180).

[59] P. Baldi, **Autoencoders, unsupervised learning, and deep architectures**, in *Proceedings of ICML workshop on unsupervised and transfer learning*, Edinburgh, Scotland, 26 June–1 July ,2012, 37–49 (see pages 32, 69, 84).

[60] C. Szegedy, L. Wei, J. Yangqing, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, **Going deeper with convolutions**, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, Jun. 7-12, 2015, 1–9 (see pages 32, 39, 48, 53–55, 83, 88, 144, 180).

[61] H. I. Fawaz, L. Benjamin, G. Forestier, P. Charlotte, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P. A. Muller, and P. François, **Inceptiontime: Finding alexnet for time series classification**, *Data Mining and Knowledge Discovery*, vol. 34, 2020, 1936–1962 (see pages 32, 39, 48, 58).

[62] P. Goyal, S. Pandey, and K. Jain, **Deep Learning for Natural Language Processing: Creating Neural Networks with Python**. Apress, 2018 (see pages 32, 33).

[63] S. Hochreiter and J. Schmidhuber, **Long short-term memory**, *Neural Computation*, vol. 9:no. 8, 1997, 1735–1780 (see page 33).

[64] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, **Learning representations by back-propagating errors**, *Nature*, vol. 323:no. 9, 1986, 533–536 (see pages 35–37).

[65] D. P. Kingma and J. Ba, **Adam: A method for stochastic optimization**, in *3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 7-9, 2015 (see pages 37, 75).

[66] X. Glorot and Y. Bengio, **Understanding the difficulty of training deep feedforward neural networks**, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Sardinia, Italy, May 13-15, 2010, 249–256 (see pages 38–43, 50, 55, 82, 88, 89).

[67] K. He, X. Zhang, S. Ren, and J. Sun, **Delving deep into rectifiers: Surpassing human-level performance on imagenet classification**, in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 7-13, 2015, 1026–1034 (see pages 39, 42, 50, 55, 82, 88, 89).

[68] C. François *et al.*, *Keras*, https://keras.io, 2015 (see pages 39, 42).

[69] S. K. Kumar, **On weight initialization in deep neural networks**, *arXiv*, vol. abs/1704.08863, 2017. [Online]. Available: http://arxiv.org/abs/1704.08863 (see page 43).

[70] A. Srivastava and E. P. Klassen, **Functional and Shape Data Analysis**. Springer, 2016, vol. 1 (see pages 43, 44).

[71] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu koray, **Spatial transformer networks**, in *Advances in Neural Information Processing Systems*, vol. 28, Montreal, Quebec, Canada, Dec. 7-12, 2015, 2017–2025 (see pages 43, 46).

[72] Y. Kowsar, M. Moshtaghi, E. Velloso, J. C. Bezdek, L. Kulik, and C. Leckie, **Shape-sphere: A metric space for analysing time series by their shape**, *Information Sciences*, vol. 582, 2022, 198–214 (see page 43).

[73] C. Chen and A. Srivastava, **Srvfregnet: Elastic function registration using deep neural networks**, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 19-25, 2021, 4462–4471 (see page 44).

[74] N. S. Detlefsen, O. Freifeld, and S. Hauberg, **Deep diffeomorphic transformer networks**, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, Jun. 18-23, 2018, 4403–4412 (see pages 44, 45).

[75] A. Krizhevsky, I. Sutskever, and G. E. Hinton, **Imagenet classification with deep convolutional neural networks**, *Communications of the ACM*, vol. 60:no. 6, 2017, 84–90 (see pages 48, 49).

[76] G. Dong, G. Liao, H. Liu, and G. Kuang, **A review of the autoencoder and its variants: A comparative perspective from target recognition in synthetic-aperture radar images**, *IEEE Geoscience and Remote Sensing Magazine*, vol. 6:no. 3, 2018, 44–68 (see pages 58, 69).

[77] E. S. Trust, **Powering the nation**, *Department for Environment, Food and Rural Affairs (DEFRA)*, 2012 (see pages 59, 60).

[78] C. Michael, K. Tom, D. Vanda, F. Steven, H. Tarek, H.-B. Richard, H. Tom, L. Charlie, S. Wilson, S. Vladimir, M. David, and L. Jing, **Utilising smart home data to support the reduction of energy demand from space heating insights from a uk field study**, in *Proceedings of the 8th International Conference on Energy Efficiency in Domestic Appliances and Lighting*, Sydney, NSW, Australia, Aug. 26-28, 2015, 1049–1063 (see pages 59, 60).

[79] C. Gisler, A. Ridi, and J. Hennebert, **Appliance consumption signature database and recognition test protocols**, in *WOSSPA2013 The 9th International Workshop on Systems, Signal Processing and their Applications*, Algiers, Algeria, May 12-15, 2013, 336–341 (see pages 59, 60).

[80] R. T. Olszewski, **Generalized feature extraction for structural pattern recognition in time-series data**, PhD dissertation, USA, 2001 (see pages 61, 62).

[81] Y. Chen, Y. Hao, T. Rakthanmanon, J. Zakaria, B. Hu, and E. Keogh, **A general framework for never-ending learning from time series streams**, *Data Mining and Knowledge Discovery*, vol. 29, 2014, 1622–1664 (see pages 61, 62).

[82] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. Ivanov, R. Mark, J. Mietus, G. Moody, C. Peng, and H. Stanley, **Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals**, in *Circulation. 2000 Jun 13;101(23):E215-20*, USA, 2000 (see pages 61, 62).

[83] T. Lugovaya, **Biometric human identification based on electrocardiogram**, Msc Thesis, Faculty of Computing Technologies and Informatics, Electrotechnical University, Saint Petersburg, Russian Federation, 2005 (see pages 61, 62).

[84] F. Fang and T. Shinozaki, **Electrooculography-based continuous eye-writing recognition system for efficient assistive communication systems**, *PLOS ONE*, vol. 13:no. 2, 2018, 1–20 (see pages 61, 62).

[85] C. Sapsanis, G. Georgoulas, A. Tzes, and D. Lymberopoulos, **Improving emg based classification of basic hand movements using emd**, in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Osaka, Japan, Jul. 3-7, 2013, 5754–5757 (see pages 61, 62).

[86] D. S. Willett, J. George, N. S. Willett, L. L. Stelinski, and S. L. Lapointe, **Machine learning for characterization of insect vector feeding**, *PLOS Computational Biology*, vol. 12:no. 11, 2016, 1–14 (see pages 61, 62).

[87] D. Eads, D. Hill, S. Davis, S. J. Perkins, J. Ma, R. Porter, and J. Theiler, **Genetic algorithms and support vector machines for time series classification**, in *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation V*, Seattle, Washington, USA, Jul. 9-10, 2002, 74–85 (see pages 62, 63).

[88] Y. Chen, A. Why, G. Batista, A. Mafra-Neto, and E. Keogh, **Flying insect detection and classification with inexpensive sensors**, *Journal of visualized experiments : JoVE*, vol. 92, 2014 (see pages 63, 64).

[89] H. Hamooni and A. Mueen, **Dual-domain hierarchical classification of phonetic time series**, in *2014 IEEE International Conference on Data Mining*, Shenzhen, China, Dec. 14-17, 2014, 160–169 (see pages 63, 64).

[90] A. C. Jalba, H. W. Michael, and R. Jos, **Automatic segmentation of diatom images for classification**, *Microscopy Research and Technique*, vol. 95, 2004, 72–85 (see page 64).

[91] L. Ye and E. Keogh, **Time series shapelets: A new primitive for data mining**, in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France, 28 June–1 July ,2009, 947–956 (see pages 65, 66).

[92] A. Mueen, E. Keogh, and N. Young, **Logical-shapelets: An expressive primitive for time series classification**, in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, California, USA, Aug. 21-24, 2011, 1154–1162 (see page 66).

[93] N. Saito, **Local feature extraction and its applications using a library of bases**. Yale University, 1994, 269–451 (see page 66).

[94] A.-J. Osama, E. K. Kemsley, and W. Reginald H., **Detection of adulteration in cooked meat products by mid-infrared spectroscopy**, *Journal of Agricultural and Food Chemistry*, vol. 50:no. 6, 2002, 1325–1329 (see pages 66, 67).

[95] S. Lu, G. Mirchevska, S. S. Phatak, D. Li, J. Luka, R. A. Calderone, and W. A. Fonzi, **Dynamic time warping assessment of high-resolution melt curves provides a robust metric for fungal identification**, *PLOS ONE*, vol. 12:no. 3, 2017, 1–21 (see pages 66–68).

[96] M. Guillame-Bert and A. Dubrawski, **Classification of time sequences using graphs of temporal constraints**, *The Journal of Machine Learning Research*, vol. 18:no. 1, 2017, 4370–4403 (see pages 66–68).

[97] A. Baldridge, S. Hook, C. Grove, and G. Rivera, **The aster spectral library version 2.0**, *Remote Sensing of Environment*, vol. 113:no. 4, 2009, 711–715 (see page 66).

[98] X. Huang, Y. Ye, L. Xiong, R. Y.K. Lau, N. Jiang, and S. Wang, **Time series k-means: A new k-means type smooth subspace clustering for time series data**, *Information Sciences*, vol. 367-368, 2016, 1–13 (see page 66).

[99] B. Dana H., **Modular learning in neural networks**, in *Proceedings of the Sixth National Conference on Artificial Intelligence*, vol. 1, Seattle, Washington, Jul. 13-17, 1987, 279–284 (see page 69).

[100] L. Karl Pearson, **On lines and planes of closest fit to systems of points in space**, *Philosophical Magazine*, 6th ser., vol. 2, 1901, 559–572 (see pages 69, 75).

[101] D. Charte, F. Charte, M. J. del Jesus, and F. Herrera, **An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges**, *Neurocomputing*, vol. 404:no. 3, 2020, 93–107 (see page 69).

[102] T. Howley, M. G. Madden, M.-L. O'Connell, and A. G. Ryder, **The effect of principal component analysis on machine learning accuracy with high dimensional spectral data**, in *Applications and Innovations in Intelligent Systems XIII*, Mar. 2006, 209–222 (see page 70).

[103] J. Lever, M. Krzywinski, and N. Altman, **Principal component analysis**, *Nature Methods*, vol. 14, 2017 (see page 70).

[104] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, **Why does unsupervised pre-training help deep learning?** *Journal of Machine Learning Researches*, vol. 11, 2010, 625–660 (see pages 70, 71).

[105] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, **Extracting and composing robust features with denoising autoencoders**, in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, Jul. 5-9, 2008, 1096–1103 (see page 71).

[106] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis, **Learning temporal regularity in video sequences**, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, Jun. 27-30, 2016 (see page 71).

[107] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. v. den Hengel, **Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection**, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea, 27 October–2-November ,2019, 1705–1714 (see page 71).

[108] B. Zong, Q. Song, M. Renqiang Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, **Deep autoencoding gaussian mixture model for unsupervised anomaly detection**, in *6th International Conference on Learning Representations, ICLR*, Vancouver, BC, Canada, 30 April–3 May ,2018 (see page 71).

[109] G. E. Hinton and R. R. Salakhutdinov, **Reducing the dimensionality of data with neural networks**, *Science*, vol. 313:no. 5786, 2006, 504–507 (see page 71).

[110] L. van der Maaten and G. Hinton, **Visualizing data using t-sne**, *Journal of Machine Learning Research*, vol. 9, 2008, 2579–2605 (see page 75).

[111] omain Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, and E. Woods, **Tslearn, a machine learning toolkit for time series data**, *Journal of Machine Learning Research*, vol. 21:no. 118, 2020, 1–6 (see pages 76, 180).

[112] J. Demšar, **Statistical comparisons of classifiers over multiple data sets**, *Journal of Machine learning research*, vol. 7:no. Jan, 2006, 1–30 (see pages 76, 77).

[113] A. Barua, P. Kishore Deb, R. Maheshwari, and R. K. Tekade, "Chapter 10 -statistical techniques in pharmaceutical product development," in *Dosage Form Design Parameters*, ser. Advances in Pharmaceutical Product Development and Research, Academic Press, 2018, 339–362 (see pages 76, 77).

[114] S. Bethelhem S., D. Tsegamlak T., T. Getinet, T. Yonas Y., and W. Dereje H., **Hybrid prediction model for mobile data traffic: A cluster-level approach**, in *2020 International Joint Conference on Neural Networks (IJCNN)*, Glasgow, United Kingdom, Jul. 19-24, 2020, 1–8 (see pages 164, 166, 171, 172, 174–176, 181).

[115] H. Kaaranen, A. Ahtiainen, L. Laitinen, S. Naghian, and V. Niemi, **UMTS Networks: Architecture, Mobility and Service**, 2nd. John Wiley and Sons, 2005 (see pages 164, 165).

[116] B. Walke, P. Seidneberg, and A. M. P, **UMTS The Fundamentals**, 1st. Wiley, 2003 (see pages 164, 165).

[117] A. M. Fazlu, M. M. Abu Kyum, and M. F. Hossain, **Performance analysis of umts cellular network using sectorization based on capacity and coverage**, *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 2, 2011, 98–104 (see page 164).

[118] S. Yantai, M. Yu, J. Liu, and O. Yang, **Wireless traffic modeling and prediction using seasonal arima models**, in *IEEE International Conference on Communications, 2003. ICC '03.* Vol. 3, 2003, 1675–1679 (see pages 168, 169).

[119] A. Azari, P. Papapetrou, S. Denic, and G. Peters, **Cellular traffic prediction and classification: A comparative evaluation of lstm and arima**, in *Discovery Science*, Cham: Springer International Publishing, 2019, 129–144 (see pages 168, 169).

[120] D. Zeng, J. Xu, J. Gu, L. Liu, and G. Xu, **Short term traffic flow prediction using hybrid arima and ann models**, in *2008 Workshop on Power Electronics and Intelligent Transportation System*, Guangzhou, China, Aug. 4-5, 2008, 621–625 (see page 169).

[121] J. G. Proakis and M. Salehi, **Fundamentals of Communication Systems**. Prentice Hall, 2002 (see page 171).

[122] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, **Scikit-learn: Machine learning in Python**, *Journal of Machine Learning Research*, vol. 12, 2011, 2825–2830 (see page 172).

[123] R. J. Hyndman and Y. Khandakar, **Automatic time series forecasting: The forecast package for r**, *Journal of Statistical Software*, vol. 27:no. 3, 2008, 1–22 (see page 172).

[124] S. Gupta, R. Kumar, K. Lu, B. Moseley, and S. Vassilvitskii, **Local search methods for k-means with outliers**, *Proceedings of the VLDB Endowment*, vol. 10:no. 7, 2017, 757–768 (see page 176).

[125] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, **Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion**, *Journal of Machine Learning Research*, vol. 11, 2010, 3371–3408 (see pages 177, 178).

List of Publications

Articles in Refereed Journals

[1] **Estimating time series averages from latent space of multi-tasking neural networks**, *Knowldge and Information Systems (KAIS)*, Under Review. Joint work with T. T. Debella, M. Devanne, J. Weber, D. H. Woldegebreal, and G. Forestier.

Articles in Refereed Conference Proceedings

[2] **Hybrid prediction model for mobile data traffic: A cluster-level approach**, in *2020 International Joint Conference on Neural Networks (IJCNN)*, Glasgow, United Kingdom, Jul. 19-24, 2020, 1–8. Joint work with S. Bethelhem S., D. Tsegamlak T., T. Getinet, T. Yonas Y., and W. Dereje H..

[3] **Time series averaging using multi-tasking autoencoder**, in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, Baltimore, MD, USA, Nov. 9-11, 2020, 1065–1072. Joint work with T. Tsegamlak, M. Devanne, J. Weber, H. Dereje, and G. Forestier.

[4] **Deep representation learning for cluster-level time series forecasting**, in *8th International conference on Time Series and Forecasting (ITISE)*, Gran Canaria, Spain, Jun. 27-30, 2022, 1–11. Joint work with T. T. Debella, B. S. Shawel, M. Devanne, J. Weber, D. H. Woldegebreal, S. Pollin, and G. Forestier.