Deep learning for time series classification

Hassan Ismail Fawaz

Année 2020 N° d'ordre : (attribué par le SCD)

UNIVERSITÉ DE HAUTE-ALSACE
UNIVERSITÉ DE STRASBOURG

THÈSE

Pour l'obtention du grade de
Docteur de l'Université de Haute-Alsace
École Doctorale: Mathématiques, Sciences de l'Information et de l'Ingénieur (ED 269)
Discipline : Informatique

Présentée et soutenue publiquement

par

Hassan ISMAIL FAWAZ

Le 21 Septembre 2020

Deep Learning For Time Series Classification

Sous la direction du Prof. Pierre-Alain MULLER

Jury

Prof. Themis PALPANAS, Université de Paris (Rapporteur)
Prof. Pierre-François MARTEAU, Université Bretagne Sud (Rapporteur)
Prof. Anthony BAGNALL, Université d'East Anglia (Examinateur)
Prof. Laetitia JOURDAN, Université de Lille (Examinatrice)
Prof. Pierre-Alain MULLER, Université de Haute-Alsace (Directeur de thèse)
Prof. Lhassane IDOUMGHAR, Université de Haute-Alsace (co-Directeur de thèse)
Prof. Germain FORESTIER, Université de Haute-Alsace (co-Directeur de thèse)
Dr. Jonathan WEBER, Université de Haute-Alsace (Encadrant de thèse)

UNIVERSITÉ DE HAUTE-ALSACE

Résumé

Faculté des Sciences et Techniques
Institut de Recherche en Informatique, Mathématiques, Automatique et Signal

Doctor of Philosophy

Deep Learning For Time Series Classification

La science des données s'intéresse aux théories et aux algorithmes permettant d'extraire des connaissances de grandes masses de données. L'analyse de séries temporelles est le sous-domaine de la science des données qui s'intéresse à l'analyse de données composées de suites de valeurs numériques ordonnées dans le temps. Les séries temporelles sont particulièrement intéressantes car elles permettent de comprendre l'évolution des états d'un processus au cours du temps. Leur analyse peut ainsi révéler des tendances, des relations et des similarités à travers les données. De très nombreux domaines produisent des données sous la forme de séries temporelles : données de santés (électrocardiogramme, glycémie, etc.), reconnaissance d'activités, télédétection, finance (cours de bourse), industrie (capteurs).

Au sein de la science des données, la classification est une tâche supervisée qui consiste à apprendre un modèle à partir de données étiquetées organisées en classes afin de prédire la classe de nouvelles données. La classification de séries temporelles s'intéresse aux algorithmes de classification dédiés au traitement de séries temporelles. Par exemple, à l'aide d'un ensemble étiqueté d'électrocardiogrammes de patients sains ou présentant un problème cardiaque, l'objectif est d'entraîner un modèle capable de prédire si un nouvel électrocardiogramme présente ou non une pathologie. Les spécificités des données temporelles imposent le développement d'algorithmes dédiés au traitement de ces données, les modèles existants pour d'autres type de données (images, vidéos, etc.) n'étant pas toujours adaptés.

Dans ce contexte, l'apprentissage profond (deep learning) s'est imposé au cours des dernières années comme une des méthodes les plus performantes pour réaliser la tâche de classification, notamment dans le domaine de la vision par ordinateur. L'objectif principal de cette thèse a été d'étudier et de développer des modèles profonds spécifiquement construits pour la classification de séries temporelles. Nous avons ainsi réalisé la première étude expérimentale permettant de comparer les méthodes profondes existantes et de les positionner par rapport aux méthodes de l'état de l'art n'utilisant pas l'apprentissage profond. Par la suite, nous avons effectué de nombreuses contributions dans ce domaine, notamment dans le cadre de l'apprentissage par transfert, l'augmentation de données, la création d'ensembles et l'attaque adversaire. Enfin, nous avons également proposé une nouvelle architecture profonde, basée sur le célèbre réseau Inception (Google), qui se positionne parmis les plus performantes à ce jour.

Nos expériences menées sur des benchmarks comportant plus d'un centaine de jeux de données nous ont permis de valider les performances de nos contributions. Enfin, nous avons également montré la pertinence des approches profondes dans le domaine de la science des données chirurgicales (surgical data science) où nous avons proposé une approche interprétable afin d'évaluer les compétences chirurgicales à partir de données cinématiques de séries temporelles multivariées.

UNIVERSITÉ DE HAUTE-ALSACE

Abstract

Faculté des Sciences et Techniques
Institut de Recherche en Informatique, Mathématiques, Automatique et Signal

Doctor of Philosophy

Deep Learning For Time Series Classification

Data science is about designing algorithms and pipelines for extracting knowledge from large masses of data. Time series analysis is a field of data science which is interested in analyzing sequences of numerical values ordered in time. Time series are particularly interesting because they allow us to visualize and understand the evolution of a process over time. Their analysis can reveal trends, relationships and similarities across the data. There exists numerous fields containing data in the form of time series: health care (electrocardiogram, blood sugar, etc.), activity recognition, remote sensing, finance (stock market price), industry (sensors), etc.

In data mining, classification is a supervised task that involves learning a model from labeled data organized into classes in order to predict the correct label of a new instance. Time series classification consists of constructing algorithms dedicated to automatically label time series data. For example, using a labeled set of electrocardiograms from healthy patients or patients with a heart disease, the goal is to train a model capable of predicting whether or not a new electrocardiogram contains a pathology. The sequential aspect of time series data requires the development of algorithms that are able to harness this temporal property, thus making the existing off-the-shelf machine learning models for traditional tabular data suboptimal for solving the underlying task.

In this context, deep learning has emerged in recent years as one of the most effective methods for tackling the supervised classification task, particularly in the field of computer vision. The main objective of this thesis was to study and develop deep neural networks specifically constructed for the classification of time series data. We thus carried out the first large scale experimental study allowing us to compare the existing deep methods and to position them compared other non-deep learning based state-of-the-art methods. Subsequently, we made numerous contributions in this area, notably in the context of transfer learning, data augmentation, ensembling and adversarial attacks. Finally, we have also proposed a novel architecture, based on the famous Inception network (Google), which ranks among the most efficient to date.

Our experiments carried out on benchmarks comprising more than a hundred data sets enabled us to validate the performance of our contributions. Finally, we also showed the relevance of deep learning approaches in the field of surgical data science where we proposed an interpretable approach in order to assess surgical skills from kinematic multivariate time series data.

Acknowledgements

This work would not have been done without the support of many people who I wish to thank here.

I would like to start by thanking my supervisors: Prof. Germain Forestier, Dr. Jonathan Weber, Prof. Lhassane Idoumghar and Prof. Pierre-Alain Muller who guided me throughout this three years journey by sharing their wisdom and intellect while giving the freedom to pursue various research ideas allowing me to gain micro-managerial skills and autonomy.

I would like to express my gratitude to Prof. Themis Palpanas and Prof. Pierre-François Marteau having accepted to review this PhD. I would also like to thank the other members of the jury who did me the honor of judging this work: Prof. Laetitia Jourdan and Prof. Anthony Bagnall.

I would like to thank Dr. François Petitjean and Prof. Geoff Webb, as well as all of the amazing colleagues at the Monash University in Melbourne, for hosting me as a visiting researcher, allowing me to nurture my doctorate with this exceptional international experience.

I would like to thank every researcher who worked on providing and preparing the public dataset which have been the backbone of my PhD work: the University of California Riverside, the University of East Anglia and the John Hopkins University.

I would like to thank the Mésocentre of Strasbourg that provided support and access to the GPU cluster, providing this huge computational resources to launch the experiments. Similarly, many thanks to Nvidia Corp. that granted us the GTX Quadro P6000 enabling a significant speed up in development and testing.

Last but not least, I would like to thank all of my colleagues at the Université de Haute-Alsace for their contributions, both professionally and personally.

Finally, I would like to thank my parents, brothers and fiancée for their support during this three years journey, as well as in the previous years.

Contents

List of Figures

List of Tables

List of Abbreviations

CST	Classification de séries temporelles
ECG	Electrocardiogram
TSC	Time Series Classification
NN	Nearest Neighbor
DTW	Dynamic Time Warping
ED	Euclidean Distance
BOSS	Bag-of-SFA-Symbols
DNN	Deep Neural Networks
NLP	Natural Language Processing
ResNet	Residual Network
CNN	Convolutional Neural Networks
UCR	University of California Riverside
UEA	University of East Anglia
TWE	Time Warp Edit
MSM	Move Split Merge
NN-DTW	NN coupled with DTW
SVM	Support Vector Machine
COTE	Collective Of Transformation-based Ensembles
HIVE-COTE	Hierarchical Vote Collective of Transformation-Based Ensembles
ST	Shapelet Transform
GPU	Graphical Processing Unit
MTS	Multivariate Time Series
CAM	Class Activation Map
MLP	Multi Layer Perceptron
ESN	Echo State Network
FC	Fully Connected
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SDAE	Stacked Denoising Auto-Encoder
DBN	Deep Belief Network
FCN	Fully Convolutional Network
GAP	Global Average Pooling
PReLU	Parametric Rectified Linear Unit
MCNN	Multi-scale Convolutional Neural Network
WS	Window Slicing
T-LeNet	Time Le-Net
WW	Window Warping
MCDCNN	Multi Channel Deep Convolutional Neural Network
MSE	Mean Squared Error
TWIESN	Time Warping Invariant Echo State Network
SGD	Stochastic Gradient Descent
EE	Elastic Ensemble
PF	Proximity Forest

MDS	Multi-Dimensional Scaling
t-SNE	t-distributed Stochastic Neighbor Embedding
TEKA	Time Elastic Kernel Averaging
NNE	Neural Network Ensemble
FGSM	Fast Gradient Sign Method
KNN	K Nearest Neighbors
AdvProp	Adversarial Propagation
BIM	Basic Iterative Method
RF	Receptive Field
SITS	Satellite Image Time Series
OSATS	Objective Structured Assessment of Technical Skills
GMF	Global Movement Features
JIGSAWS	JHU-ISI Gesture and Skill Assessment Working Set
s-HMM	Sparse Hidden Markov Model
ApEn	Approximate Entropy
NLTS	Non-Linear Temporal Scaling

Résumé des chapitres en Français

Chapitre 1: L'état de l'art de la classification de séries temporelles

Au cours des deux dernières décennies, la Classification de Séries Temporelles (CST) a été considéré comme l'un des problèmes les plus difficiles dans la fouille de données (Yang and Wu, 2006; Esling and Agon, 2012). Avec l'augmentation de la disponibilité des données temporelles (Silva et al., 2018), des centaines d'algorithmes de CST ont été proposés depuis 2015 (Bagnall et al., 2017). En raison de leur ordre temporel, les séries sont présentes dans presque tout problème de fouille de données (Längkvist, Karlsson, and Loutfi, 2014). En fait, tout problème de classification, utilisant des données enregistrées ayant un ordre spécifique, peut être converti en un problème de CST (Cristian Borges Gamboa, 2017). Les séries temporelles sont présentes dans de nombreuses applications du monde réel, allant des soins de santé (Gogolou et al., 2018) à la reconnaissance de l'activité humaine (Wang et al., 2018; Mathis, Ismail Fawaz, and Khamis, 2020) jusqu'à la classification des scènes acoustiques (Nwe, Dat, and Ma, 2017) et la cybersécurité (Susto, Cenedese, and Terzi, 2018). De plus, la diversité des types d'ensembles de données dans l'archive UCR/UEA (Dau et al., 2019; Bagnall et al., 2017) (la plus grande base de données de références de séries temporelles) montre les différentes applications de la CST.

Compte tenu de la nécessité de classer avec précision les séries temporelles, les chercheurs ont proposé des centaines de méthodes pour résoudre cette tâche (Bagnall et al., 2017). L'une des approches de CST les plus populaires et traditionnelles est l'utilisation d'un classifieur NN couplé à une fonction de distance (Lines and Bagnall, 2015). En particulier, DTW lorsqu'il est utilisé avec un classifieur NN s'est révélé être une méthode de référence très solide (Bagnall et al., 2017). Lines and Bagnall, 2015 ont comparé plusieurs mesures de distance - telles que TWE (Marteau, 2009) et MSM (Stefan, Athitsos, and Das, 2013) - montrant qu'il n'y a pas de mesure de distance unique qui surpasse DTW. Ils ont également montré que l'ensembling des classifieurs NN individuels (avec différentes mesures de distance) surpasse toutes les composantes individuelles de l'ensemble. Par conséquent, les contributions récentes se sont concentrées sur le développement de méthodes d'ensembling qui supplante considérablement le NN-DTW (Bagnall et al., 2016; Hills et al., 2014; Bostrom and Bagnall, 2015; Lines, Taylor, and Bagnall, 2016; Schäfer, 2015; Kate, 2016; Deng et al., 2013; Baydogan, Runger, and Tuv, 2013). Ces approches utilisent soit un ensemble d'arbres de décision (forêt aléatoire) (Baydogan, Runger, and Tuv, 2013; Deng et al., 2013) ou un ensemble de différents types de classifieurs (SVM, NN avec plusieurs distances) sur une ou plusieurs transformations (Bagnall et al., 2016; Bostrom and Bagnall, 2015; Schäfer, 2015; Kate, 2016). La plupart de ces approches obtiennent des résultats significativement meilleurs que NN-DTW (Bagnall et al., 2017) et partagent une propriété commune, qui est la phase de transformation des données où les séries temporelles sont transformées en un nouvel espace de description (par exemple en utilisant des transformations shapelets (Bostrom and Bagnall, 2015) ou des caractéristiques DTW (Kate, 2016)). Cette notion a motivé

le développement d'un ensemble de 35 classifieurs nommé COTE (Bagnall et al., 2016) qui non seulement regroupe différents classifieurs sur la même transformation, mais regroupe à la place différents classifieurs utilisant plusieurs transformations latentes de séries temporelles. Lines, Taylor, and Bagnall, 2016; Lines, Taylor, and Bagnall, 2018 ont étendu COTE avec HIVE-COTE qui permet une amélioration significative par rapport à COTE en tirant parti d'une nouvelle structure hiérarchique avec un vote probabiliste, comprenant deux nouveaux classifieurs et deux domaines de transformation de représentation supplémentaires. HIVE-COTE est actuellement considéré comme l'état de l'art pour la classification des séries temporelles (Bagnall et al., 2017) lorsqu'il est évalué sur les 85 jeux de données de l'archive UCR/UEA.

Pour atteindre sa haute précision, HIVE-COTE devient extrêmement intensif en calcul et peu pratique pour fonctionner sur un vrai problème d'exploration de données volumineuses (Bagnall et al., 2017). L'approche nécessite l'entrainement de 37 classifieurs ainsi que la validation croisée de chaque hyperparamètre de ces algorithmes, ce qui rend l'approche impossible à appliquer dans certaines situations (Lucas et al., 2018). Pour souligner cette inaptitude, notons que l'un de ces 37 classifieurs est le ST (Hills et al., 2014) dont la complexité temporelle est $O(n^2 \cdot l^4)$ avec n étant le nombre de séries temporelles dans l'ensemble de données et l étant la longueur d'une série temporelle. À la complexité du temps d'apprentissage s'ajoute le temps de *classification* élevé de l'un des 37 classifieurs: le plus proche voisin qui doit analyser l'ensemble d'apprentissage avant de prendre une décision au moment du test. Par conséquent, étant donné que le plus proche voisin constitue une composante essentielle de HIVE-COTE, son déploiement dans un environnement en temps réel est encore limité, voire impossible. Enfin, s'ajoutant à l'énorme temps d'exécution de HIVE-COTE, la décision prise par 37 classifieurs ne peut pas être interprétée facilement par les experts du domaine, car il est déjà difficile de comprendre les décisions prises par un unique classifieur. Notons que récemment, Bagnall et al., 2020 ont proposé une nouvelle version de HIVE-COTE qui est sensiblement plus rapide, montrant l'importance de pouvoir faire évoluer les méthodes de CST.

Après avoir établi l'état de l'art actuel des classifieurs non profonds pour la CST (Bagnall et al., 2017), nous discutons du succès de l'apprentissage profond (LeCun, Bengio, and Hinton, 2015) dans diverses tâches de classification qui ont motivé l'utilisation récente de l'apprentissage profond pour la CST (Wang, Yan, and Oates, 2017). Les CNNs profonds ont révolutionné le domaine de la vision par ordinateur (Krizhevsky, Sutskever, and Hinton, 2012). Par exemple, en 2015, les CNNs ont été utilisés pour atteindre les performances au niveau humain dans les tâches de reconnaissance d'image (Szegedy et al., 2015). Suite au succès des DNNs en vision par ordinateur, de nombreuses recherches ont proposé plusieurs architectures DNN pour résoudre de nombreuses tâches de NLP telles que la traduction automatique (Sutskever, Vinyals, and Le, 2014; Bahdanau, Cho, and Bengio, 2015), la représentation vectorielle des mots (Mikolov et al., 2013; Mikolov et al., 2013) et la classification des documents (Le and Mikolov, 2014; Goldberg, 2016). Les DNNs ont également eu un impact énorme sur la communauté de reconnaissance vocale (Hinton et al., 2012; Sainath et al., 2013). Il est intéressant de noter que la similitude intrinsèque entre le NLP et les tâches de reconnaissance vocale est due à l'aspect séquentiel des données qui est également l'une des principales caractéristiques des séries temporelles.

Dans ce contexte, ce premier chapitre cible les questions ouvertes suivantes: *Quel est le meilleur DNN actuel pour la CST*? *Existe-t-il une approche DNN moins complexe que HIVE-COTE pouvant obtenir des résultats compétitifs avec l'état de l'art* ? *Quel type d'architectures DNN fonctionne le mieux pour la tâche de CST*? *Comment l'initialisation*

aléatoire affecte-t-elle les performances des classifieurs d'apprentissage profond ? Et enfin: *L'effet boîte noire des DNNs pourrait-il être évité pour fournir une interprétabilité des résultats* ? Étant donné que ces dernières questions n'ont pas été abordées par la communauté de CST, il est surprenant de voir que les DNNs n'ont pas été considéré comme classifieur précis potentiel des séries temporelles (Lines, Taylor, and Bagnall, 2018). En fait, une étude empirique récente (Bagnall et al., 2017) a évalué 18 algorithmes de CST sur 85 jeux de données, dont aucun n'était un modèle d'apprentissage profond. Cela montre à quel point la communauté n'a pas une vue d'ensemble des performances actuelles des modèles d'apprentissage profond pour résoudre le problème de CST (Lines, Taylor, and Bagnall, 2018).

Dans ce chapitre, nous avons effectué une étude comparative empirique des approches d'apprentissage profond les plus récentes pour la CST. Avec la montée en puissance des GPUs, nous avons montré comment les architectures profondes peuvent être entraînées efficacement pour apprendre de bout en bout des fonctionnalités discriminantes cachées, à partir de séries temporelles brutes. De manière similaire à Bagnall et al., 2017, afin d'avoir une comparaison équitable entre les approches testées, nous avons développé une plateforme en Python, Keras (Chollet, 2015) et Tensorflow (Abadi et al., 2015) pour entrainer les modèles d'apprentissage profond sur un cluster de plus de 60 GPUs.

En plus de l'évaluation des ensembles de données univariés, nous avons testé les approches sur 12 jeux de données MTS (Baydogan, 2015). L'évaluation multivariée montre un autre avantage des modèles d'apprentissage en profondeur, qui est la capacité de gérer la malédiction de la dimensionnalité (Bellman, 2010; Keogh and Mueen, 2017) en exploitant différents degrés de fluidité dans la fonction de composition (Poggio et al., 2017) ainsi que les calculs parallèles des GPUs (Lu et al., 2015).

Quant à la comparaison des classifieurs sur plusieurs ensembles de données, nous avons suivi les recommandations de Demšar, 2006 et utilisé le test de Friedman (Friedman, 1940) pour rejeter l'hypothèse nulle. Une fois que nous avons établi qu'il existe une différence statistique dans les performances des classifieurs, nous avons suivi l'analyse post-hoc par paires recommandée par Benavoli, Corani, and Mangili, 2016 où la comparaison de rang moyen est remplacée par un test de rang signé Wilcoxon (Wilcoxon, 1945) avec la correction alpha de Holm (Holm, 1979; Garcia and Herrera, 2008).

Dans cette étude, nous avons entrainé environ 1 milliard de paramètres dans 97 jeux de données de séries temporelles univariées et multivariées. Malgré le fait qu'un grand nombre de paramètres risquent de sur-apprendre (Zhang et al., 2017) les ensembles d'entrainement relativement petit dans les archives UCR/UEA, nos expériences ont montré que non seulement les DNNs sont capables de surpasser considérablement le NN-DTW, mais sont également capables d'obtenir des résultats qui ne sont *pas significativement* différents de COTE et HIVE-COTE en utilisant une architecture de réseau résiduel profond (He et al., 2016; Wang, Yan, and Oates, 2017). Enfin, nous avons analysé comment de mauvaises initialisations aléatoires peuvent avoir un effet significatif sur les performances d'un DNN.

En conclusion, avec les problèmes de fouille de données de plus en plus fréquents, tirant parti d'architectures plus approfondies qui peuvent apprendre automatiquement de bout en bout des données annotées, l'apprentissage en profondeur est une approche très attrayante. Dans ce chapitre, nous avons montré le potentiel des réseaux de neurones profonds pour le problème de CST, néanmoins ces modèles complexes d'apprentissage automatique peuvent encore bénéficier de nombreuses techniques de régularisation, qui est l'objectif principal du chapitre suivant.

Chapitre 2: Regularisation des réseaux de neurones profonds

Les modèles d'apprentissage profond ont généralement plus de paramètres à entrainer qu'il n'y a d'instances d'entrainement. Néanmoins, dans le chapitre précédent, nous avons montré comment ces réseaux de neurones artificiels sont capables d'atteindre de bonnes capacités de généralisation par rapport aux algorithmes de CST traditionnels. Pourtant, la plupart de ces modèles nécessitent une sorte de régularisation afin de minimiser l'erreur de généralisation (en d'autres termes minimiser la différence entre l'erreur d'entrainement et d'erreur de test). Dans ce chapitre, nous présentons les quatre principales techniques pour régulariser les DNNs pour la CST.

Apprentissage par transfert: Nous avons récemment montré que les CNNs peuvent atteindre des performances similaires à celles de l'état de l'art. Cependant, malgré les performances élevées de ces CNNs, les modèles d'apprentissage profond sont toujours sujets au sur-apprentissage. Un exemple où ces réseaux de neurones ne parviennent pas à généraliser est lorsque l'ensemble des données d'apprentissage est très petit. Nous attribuons cette énorme différence de précision au phénomène de sur-apprentissage, qui reste un domaine de recherche ouvert dans la communauté (Zhang et al., 2017). Ce problème est connu pour être atténué à l'aide de plusieurs techniques de régularisation telles que l'apprentissage par transfert (Yosinski et al., 2014a), où un modèle entraîné sur une première tâche est ensuite affiné sur un ensemble de données cible. L'apprentissage par transfert est actuellement utilisé dans presque tous les modèles d'apprentissage en profondeur lorsque l'ensemble de données cible ne contient pas suffisamment de données étiquetées (Yosinski et al., 2014a). Malgré son récent succès en vision par ordinateur (Csurka, 2017), l'apprentissage par transfert a rarement été appliqué aux modèles d'apprentissage profond dédiés aux séries temporelles. L'une des raisons de cette absence est probablement le manque d'un grand ensemble de données généralistes similaire à ImageNet (Russakovsky et al., 2015) ou OpenImages(Krasin et al., 2017) mais pour les séries temporelles. De plus, ce n'est que récemment que l'apprentissage en profondeur s'est avéré efficace pour la CST (Cui, Chen, and Chen, 2016) et il reste encore beaucoup à explorer dans la construction de réseaux de neurones profonds pour l'analyse de séries temporelles (Cristian Borges Gamboa, 2017). Comme le transfert de modèles d'apprentissage en profondeur, entre les différents jeux de données des archives UCR/UEA (Dau et al., 2019), n'a pas été étudié de manière approfondie, nous avons décidé de nous y atteler dans le but ultime de déterminer à l'avance quels types de jeux de données pourraient bénéficier du transfert de modèles CNNs et améliorer leur précision.

Ensembling: Une autre façon d'améliorer les classifieurs basés sur les réseaux de neurones est de construire un ensemble de modèles d'apprentissage profond. Cette idée semble très intéressante pour les tâches de CST car l'état de l'art évolue vers des solutions d'ensemble (Lines, Taylor, and Bagnall, 2018; Lines and Bagnall, 2015; Bagnall et al., 2017; Baydogan, Runger, and Tuv, 2013). De plus, les ensembles de réseaux de neurones profonds semblent obtenir des résultats très prometteurs dans de nombreux domaines de l'apprentissage automatique supervisé tels que la détection des lésions cutanées (Goyal and Rajapakse, 2018), la reconnaissance d'expression faciale (Wen et al., 2017) et le remplissage automatique des seaux (Dadhich, Sandin, and Bodin, 2018). Par conséquent, nous proposons de regrouper les modèles actuels d'apprentissage profond pour la CST développés dans le chapitre précédent, en construisant un modèle composé de 60 réseaux de neurones profonds différents: 6 architectures différentes (Wang, Yan, and Oates, 2017; Zheng et al.,

2014; Zhao et al., 2017; Serrà, Pascual, and Karatzoglou, 2018) chacun avec 10 initialisations différentes des poids du modèle. En évaluant sur les 85 jeux de données de l'archive UCR/UEA, nous démontrons une amélioration significative par rapport aux classifieurs individuels tout en atteignant des performances très similaires à HIVE-COTE: méthode ensembliste contenant 37 classifieurs différents et représentant actuellement l'état de l'art. Enfin, en nous inspirant de nos résultats sur l'apprentissage par transfert (Ismail Fawaz et al., 2018d), nous remplaçons les réseaux initialisés de manière aléatoire par un ensemble construit à partir de modèles affinés à partir des 84 autres jeux de données de l'archive, et montrons une amélioration significative pour la CST.

Augmentation de données: Bien que les CNNs profonds récemment proposés aient atteint des hautes performances pour la CST sur l'archive UCR/UEA (Wang, Yan, and Oates, 2017), ils montrent toujours de faibles capacités de généralisation sur certains petits jeux de données tels que l'ensemble de données CinCECGTorso avec 40 instances d'entraînement. Cela est surprenant car le NN-DTW fonctionne exceptionnellement bien sur ce jeu de données, ce qui montre la relative facilité de cette tâche de classification. Ainsi, les similitudes entre séries temporelles dans ces petits jeux de données ne peuvent pas être capturées par les CNNs en raison du manque d'instances étiquetées, ce qui pousse l'algorithme d'optimisation du réseau à être bloqué dans des minimums locaux (Zhang et al., 2017). Ce phénomène, également connu sous le nom de sur-apprentissage dans la communauté du machine learning, peut être résolu en utilisant différentes techniques telles que la régularisation ou simplement la collecte de données étiquetées supplémentaires (Zhang et al., 2017) (qui, dans certains domaines, sont difficiles à obtenir). Une autre technique bien connue est l'augmentation des données, où les données synthétiques sont générées à l'aide d'une méthode spécifique. Par exemple, les images contenant des numéros de rue sur des maisons peuvent être légèrement pivotées sans changer leur numéro réel (Krizhevsky, Sutskever, and Hinton, 2012). Pour les modèles d'apprentissage en profondeur, ces méthodes sont généralement proposées pour les données d'image et se généralisent mal aux séries temporelles (Um et al., 2017). Cela est probablement dû au fait que pour les images, une comparaison visuelle peut confirmer si la transformation (telle que la rotation) n'a pas modifié la classe de l'image, tandis que pour les séries temporelles, on ne peut pas facilement confirmer l'effet de telles transformations ad hoc sur la nature d'une série. Nous proposons de tirer parti d'une technique d'augmentation de données basée sur DTW spécifiquement développée pour les séries temporelles, afin d'améliorer les performances d'un ResNet profond pour la CST. Nos expériences préliminaires révèlent que l'augmentation des données peut améliorer considérablement la précision des CNNs sur certains jeux de données tout en ayant un impact négatif faible sur d'autres jeux de données.

Attaques adversaires: Comme nous l'avons déjà discuté, la CST est utilisée dans diverses tâches d'exploration de données du monde réel, allant des soins de santé et de la sécurité (Tan, Webb, and Petitjean, 2017; Tobiyama et al., 2016) à la sécurité alimentaire et surveillance de la consommation d'énergie (Owen and Foreman, 2012). Avec des modèles d'apprentissage en profondeur révolutionnant de nombreux domaines de l'apprentissage automatique tels que la vision par ordinateur (Krizhevsky, Sutskever, and Hinton, 2012) et le traitement du langage naturel (Yang et al., 2018; Wang, Li, and Xu, 2018), nous avons montré dans le chapitre précédent que ces modèles ont commencé à être adopté pour les tâches de CST (Ismail Fawaz et al., 2019d). Après l'avènement du deep learning, les chercheurs ont commencé à étudier la vulnérabilité des réseaux profonds aux attaques adversaires (Yuan et al., 2017). Dans le cadre de la reconnaissance d'images, une attaque

adversaire consiste à modifier une image originale afin que les changements soient quasiment indétectables par un humain (Yuan et al., 2017). L'image modifiée est appelée une image adversaire, qui sera mal classée par le réseau de neurone, tandis que l'image d'origine est correctement classée. L'une des attaques les plus célèbres de la vie réelle consiste à modifier une image de panneau de signalisation afin qu'elle soit mal interprétée par un véhicule autonome (Eykholt et al., 2018). Une autre application est l'altération du contenu illégal pour le rendre indétectable par les algorithmes de modération automatique (Yuan et al., 2017). Bien que ces approches aient été intensivement étudiées dans le contexte de la reconnaissance d'image, elles n'ont pas été étudiées en profondeur pour la CST. Cela est surprenant car les modèles d'apprentissage en profondeur deviennent de plus en plus populaires pour classer les séries temporelles (Ismail Fawaz et al., 2019d). En outre, des attaques adverses potentielles sont présentes dans de nombreuses applications où l'utilisation de données de séries temporelles est cruciale. Par exemple, dans ce chapitre on montre la similitude entre une série temporelle originale et perturbée de spectrographe de grains de café. Alors qu'un réseau de neurones profond classe correctement la série d'origine en tant que grains Robusta, l'ajout de petites perturbations le classe comme Arabica. Par conséquent, étant donné que les grains Arabica ont plus de valeur que les grains Robusta, cette attaque pourrait être utilisée pour tromper les tests de contrôle des aliments et, éventuellement, les consommateurs. Nous présentons, transférons et adaptons les attaques qui se sont avérées efficaces sur les images, aux données de séries temporelles. Nous présentons également une étude expérimentale utilisant les 85 jeux de données de l'archive UCR/UEA (Dau et al., 2019) qui révèle que les réseaux de neurones sont sensibles aux attaques adversaires. Nous mettons en évidence des cas d'utilisation concrets spécifiques pour souligner l'importance de ces attaques dans des situations réelles, à savoir la qualité et la sécurité des aliments, les capteurs des véhicules et la consommation d'électricité. Nos résultats montrent que les réseaux profonds pour les données de séries temporelles sont vulnérables aux attaques adverses comme leurs homologues en vision par ordinateur. Par conséquent, ce travail met en lumière la nécessité de se protéger contre de telles attaques, en particulier lorsque l'apprentissage profond est utilisé pour les applications sensibles de CST. Nous montrons également que les séries temporelles adverses apprises à l'aide d'une architecture de réseau peuvent être transférées à différentes architectures. Nous discutons ensuite de certains mécanismes pour empêcher ces attaques tout en renforçant la robustesse des modèles aux attaques adversaires. Enfin, dans un esprit de régularisation des DNNs, nous montrons comment ces séries perturbées peuvent être exploitées afin d'améliorer la capacité de généralisation d'un modèle d'apprentissage en profondeur: une technique appelée entraînement adversaire (Xie et al., 2020).

Chapitre 3: InceptionTime: Recherche d'AlexNet pour la classification de séries temporelles

Les industries allant des soins de santé (Forestier et al., 2018; Lee et al., 2018; Ismail Fawaz et al., 2019c) et de la sécurité sociale (Yi et al., 2018) à la reconnaissance de l'activité humaine (Yuan et al., 2018) et à la télédétection (Pelletier, Webb, and Petitjean, 2019), produisent toutes des séries temporelles d'une échelle jamais vue auparavant — à la fois en termes de longueur et de quantité de séries. Cette croissance signifie également une dépendance accrue à l'égard de la classification automatique

de ces données séquentielles et, idéalement, des algorithmes capables de le faire à grande échelle.

Dans les chapitres précédents, nous avons montré en quoi ces problèmes de CST diffèrent considérablement de l'apprentissage supervisé traditionnel pour les données structurées, en ce que les algorithmes doivent être capables de gérer et d'exploiter les informations temporelles présentes dans le signal. Il est facile d'établir des parallèles entre ce scénario et des problèmes de vision par ordinateur tels que la classification d'images et la localisation d'objets, où les algorithmes efficaces apprennent des informations spatiales contenues dans une image. En termes simples, le problème de classification des séries temporelles est essentiellement la même classe de problèmes, juste avec une dimension de moins. Pourtant, malgré cette similitude, les algorithmes d'état de l'art actuels des deux domaines partagent peu de ressemblance (Ismail Fawaz et al., 2019d).

Le deep learning a une longue histoire (en termes d'apprentissage automatique) en vision par ordinateur (LeCun et al., 1998) mais sa popularité a explosé avec AlexNet (Krizhevsky, Sutskever, and Hinton, 2012), après quoi il a été incontestablement la classe d'algorithmes avec le plus de réussite (LeCun, Bengio, and Hinton, 2015). Inversement, l'apprentissage en profondeur n'a commencé à gagner en popularité que récemment parmi les chercheurs en exploration de données temporelles (Ismail Fawaz et al., 2019d). Ceci est souligné par le fait que ResNet, qui est actuellement considéré comme l'architecture de réseau neurone d'état de l'art pour la CST lorsqu'elle est évaluée sur l'archive UCR/UEA (Dau et al., 2019), a été initialement proposé simplement comme modèle de base pour la tâche sous-jacent (Wang, Yan, and Oates, 2017). Compte tenu des similitudes dans les données, il est facile de suggérer qu'il y a beaucoup d'amélioration potentielle pour l'apprentissage profond dans la CST. Dans le chapitre précédent, nous avons montré comment il est possible d'améliorer la précision d'une architecture d'apprentissage profond donnée, en utilisant diverses techniques de régularisation telles que l'ensembling, l'apprentissage par transfert, l'augmentation des données et l'apprentissage adversaire. Cependant, nous pensons qu'il y a encore place à l'amélioration en termes d'architecture de réseau, qui peut être considérée comme une tâche orthogonale aux différentes méthodes de régularisation des DNNs.

Dans ce chapitre, nous franchissons une étape importante vers la recherche de l'équivalent d'AlexNet pour la CST en présentant InceptionTime — un nouvel ensemble d'apprentissage profond pour la CST. InceptionTime atteint la précision d'état de l'art lorsqu'il est évalué sur l'archive UCR/UEA (actuellement le plus grand référentiel publiquement disponible pour la CST (Dau et al., 2019)) et passe mieux à l'échelle dû à son coût computationnel plus faible.

InceptionTime est un ensemble de cinq modèles d'apprentissage profond pour la CST, chacun est créé en cascadant plusieurs modules Inception (Szegedy et al., 2015). Chaque classifieur individuel (modèle) aura exactement la même architecture mais avec une initialisation des poids différente et aléatoire. L'idée centrale d'un module Inception est d'appliquer simultanément plusieurs filtres à une série temporelle en entrée. Le module comprend des filtres de longueurs variables qui, comme nous le montrerons, permettent au réseau d'extraire automatiquement les caractéristiques pertinentes des séries temporelles longues et courtes. En fait, InceptionTime suit ici l'idée d'ensemble présentée dans le chapitre précédent.

Après avoir présenté InceptionTime et ses résultats, nous effectuons une analyse des hyperparamètres architecturaux des réseaux de neurones profonds — profondeur, longueur du filtre, nombre de filtres — et les caractéristiques du module Inception — le bottleneck et les connexions résiduelles, afin de mieux comprendre

pourquoi ce modèle connaît un tel succès. En fait, nous construisons des réseaux avec des filtres plus grands que ceux utilisés pour les tâches de vision par ordinateur, profitant directement du fait que les séries temporelles présentent une dimension de moins que les images.

En conclusion, l'apprentissage profond pour la classification des séries temporelles est toujours en retard par rapport aux réseaux de neurones pour la reconnaissance d'images en termes d'études expérimentales et de conceptions architecturales. Dans ce chapitre, nous comblons cette lacune en introduisant InceptionTime, inspiré par le récent succès des réseaux basés sur Inception pour diverses tâches de vision par ordinateur. Nous avons regroupé ces réseaux pour produire de nouveaux résultats d'état de l'art pour la CST sur les 85 jeux de données du UCR/UEA archive. Notre approche est hautement évolutive, deux ordres de grandeur plus rapide que les modèles d'état de l'art actuels tels que HIVE-COTE. L'ampleur de cette accélération est intéressante dans un contexte Big Data ainsi que pour des séries temporelles plus longues avec un taux d'échantillonnage élevé. Nous étudions en outre les effets sur la précision globale de divers hyperparamètres des architectures CNN. Pour ceux-ci, nous allons bien au-delà des pratiques standard pour les données d'images et la conception de réseaux avec de longs filtres. Nous les examinons en utilisant un ensemble de données simulées et encadrons notre étude en termes de définition du "receptive field" d'un CNN pour la CST.

Chapitre 4: Analyse de séries temporelles pour la formation chirurgicale

Au cours des cent dernières années, la méthodologie d'enseignement classique de "voir un, faire un, enseigner un" a dominé les systèmes d'enseignement chirurgical dans le monde. Avec l'avènement de salle d'opération 2.0, l'enregistrement des vidéos, des données cinématiques et de nombreux autres types de données au cours d'une opération chirurgicale est devenu une tâche facile, permettant ainsi aux systèmes d'intelligence artificielle d'être déployés et utilisés dans la pratique chirurgicale et médicale. Récemment, il a été démontré que les données des capteurs de mouvement (par exemple cinématique) ainsi que les vidéos chirurgicales fournissent une structure de coaching permettant aux stagiaires débutants d'apprendre des chirurgiens expérimentés en rejouant ces vidéos et/ou trajectoires cinématiques. Dans ce chapitre, nous abordons deux problèmes présents dans le programme actuel de formation chirurgicale.

Évaluation des compétences chirurgicales: L'idée principale de la méthodologie d'enseignement du Dr William Halsted est que l'étudiant pourrait devenir un chirurgien expérimenté en observant et en participant à des chirurgies encadrées (Polavarapu et al., 2013). Ces techniques de formation, bien que largement utilisées, sont dépourvues d'une méthode objective d'évaluation des compétences chirurgicales (Kassahun et al., 2016). L'évaluation standard des compétences chirurgicales est actuellement basée sur des listes de contrôle remplies par un expert observant la tâche chirurgicale (Ahmidi et al., 2017). Afin de prédire le niveau de compétence d'un stagiaire sans utiliser le jugement d'un chirurgien expert, l'OSATS a été proposé et est actuellement adopté comme pratique clinique standard (Niitsu et al., 2013). Hélas, ce type de notation observationnelle souffre encore de plusieurs facteurs externes et subjectifs tels que la fiabilité inter-évaluateurs, le processus de développement et le biais de la liste de contrôle et de l'évaluateur (Hatala et al.,

2015). D'autres études ont démontré une corrélation entre les compétences techniques d'un chirurgien et les résultats postopératoires (Bridgewater et al., 2003).

Cette dernière approche souffre du fait que les suites d'une intervention chirurgicale dépendent des attributs physiologiques du patient (Kassahun et al., 2016). De plus, l'obtention de ce type de données est très ardue, ce qui rend ces techniques d'évaluation des compétences difficiles à mettre en œuvre pour la formation chirurgicale. Les progrès récents en robotique chirurgicale tels que le système chirurgical *da Vinci* (Intuitive Surgical Sunnyvale, 2018) ont permis l'enregistrement de données vidéo et cinématiques de diverses tâches chirurgicales. Par conséquent, un substitut des listes de contrôle et des approches basées sur les résultats, est de générer, à partir de ces cinématiques, des GMFs tels que la vitesse de la tâche chirurgicale, l'achèvement du temps, la fluidité du mouvement, la courbure et d'autres caractéristiques holistiques (Zia and Essa, 2018; Kassahun et al., 2016). Bien que la plupart de ces techniques soient efficaces, il n'est pas évident de voir comment elles pourraient être utilisées pour soutenir le stagiaire avec une rétroaction détaillée et constructive, afin d'aller au-delà d'une classification naïve dans un niveau de compétence (c.-à-d. expert, intermédiaire, etc.) . Cela est problématique car le retour d'expérience sur la pratique médicale permet aux chirurgiens d'atteindre des niveaux de compétence plus élevés tout en améliorant leurs performances (Islam et al., 2016). Dernièrement, un domaine intitulé *Surgical Data Science* (Maier-Hein et al., 2017) est apparu motivé par l'accès croissant à une énorme quantité de données complexes qui concernent le personnel, le patient et les capteurs pour capturer la procédure et les données relatives au patient telles que les variables cinématiques et les images (Gao et al., 2014). Au lieu d'extraire des GMFs, les enquêtes récentes ont tendance à décomposer manuellement les tâches chirurgicales en segments plus fins appelés "gestes", avant d'entraîner le modèle, et enfin à estimer les performances des stagiaires en fonction de leur évaluation au cours de ces gestes individuels (Tao et al., 2012). Même si ces méthodes ont obtenu des résultats prometteurs et précis en termes d'évaluation des compétences chirurgicales, elles nécessitent d'étiqueter une énorme quantité de gestes avant de former l'estimateur (Tao et al., 2012). Nous avons souligné deux limites majeures dans les techniques actuelles existantes qui estiment le niveau de compétence des chirurgiens à partir de leurs variables cinématiques correspondantes: premièrement, l'absence d'un résultat interprétable de la prédiction de compétence qui peut être utilisé par les stagiaires pour atteindre des niveaux de compétence chirurgicale plus élevés; deuxièmement, l'exigence de limites de gestes prédéfinies par les annotateurs qui est sujette à la fiabilité inter-annotateurs et qui prend du temps à préparer (Vedula et al., 2016). Dans cette première partie du chapitre, nous concevons une nouvelle architecture basée sur les FCNs, dédiée à l'évaluation des compétences chirurgicales. En utilisant des noyaux unidimensionnels sur les séries temporelles cinématiques, nous évitons d'avoir à extraire de gestes peu fiables et sensibles. La structure hiérarchique originale de notre modèle nous permet de capturer des informations globales spécifiques au niveau de compétence chirurgicale, ainsi que de représenter les gestes dans des caractéristiques latentes de bas niveau. De plus, pour fournir une rétroaction interprétable, au lieu d'utiliser une couche dense comme la plupart des architectures traditionnelles d'apprentissage profond, nous plaçons une couche GAP qui nous permet de profiter de la carte d'activation de classe, proposée à l'origine par Zhou et al., 2016, pour localiser quelle partie de l'exercice a eu un impact sur la décision du modèle lors de l'évaluation du niveau de compétence d'un chirurgien. En utilisant une configuration expérimentale standard sur le plus grand ensemble de données publiques pour l'analyse des données robotiques chirurgicales: le JHU-ISI Gesture and Skill Assessment Working Set (Gao

et al., 2014), nous montrons la précision de notre modèle FCN. Notre principale contribution est de démontrer que l'apprentissage en profondeur peut être mis à profit pour comprendre les structures complexes et latentes lors de la classification des compétences chirurgicales et de la prévision du score OSATS d'une chirurgie, d'autant plus qu'il y a encore beaucoup à apprendre sur ce qui constitue exactement une compétence chirurgicale (Kassahun et al., 2016).

Alignement des vidéos chirurgicales: Les éducateurs ont toujours cherché des moyens innovants d'améliorer le taux d'apprentissage des apprenants. Alors que les cours classiques sont encore les plus utilisés, les ressources multimédias sont de plus en plus adoptées (Smith and Ransbottom, 2000), en particulier dans les cours en ligne ouverts et massifs (Means et al., 2009). Dans ce contexte, les vidéos ont été considérées comme particulièrement intéressantes car elles peuvent combiner les images, textes, graphiques, audios et animations. Le domaine médical ne fait pas exception et l'utilisation de ressources vidéo est intensivement adoptée dans le programme médical (Masic, 2008), en particulier dans le contexte de la formation chirurgicale (Kneebone et al., 2002). L'avènement de la chirurgie robotique stimule également cette tendance, car les robots chirurgicaux, comme le Da Vinci (Intuitive Surgical Sunnyvale, 2018), enregistrent généralement des flux vidéo pendant l'intervention. Par conséquent, une grande quantité de données vidéo a été enregistrée au cours des dix dernières années (Rapp et al., 2016). Cette nouvelle source de données représente une opportunité sans précédent pour les jeunes chirurgiens d'améliorer leurs connaissances et leurs compétences (Gao et al., 2014). En outre, la vidéo peut également être un outil pour les chirurgiens seniors pendant les périodes d'enseignement pour évaluer les compétences des stagiaires. En fait, une étude récente de Mota et al., 2018 a montré que les résidents passent plus de temps à regarder des vidéos que des spécialistes, soulignant la nécessité pour les jeunes chirurgiens de profiter pleinement de cet outil. Dans Herrera-Almario et al., 2016, les auteurs ont montré que les scores obtenus pour la tâche nœuds ainsi que leurs temps de réalisation des exercices se sont considérablement améliorés pour les sujets qui ont regardé les vidéos de leur propre performance.

Cependant, lorsque les stagiaires sont prêts à évaluer leurs progrès au cours de plusieurs essais de la même tâche chirurgicale en revoyant simultanément leurs vidéos chirurgicales enregistrées, le fait que les vidéos soient désynchronisées rend la comparaison entre les différents essais très difficile, voire impossible. Ce problème se rencontre dans de nombreuses études de cas réels, car les experts accomplissent en moyenne les tâches chirurgicales en moins de temps que les chirurgiens débutants (McNatt and Smith, 2001). Ainsi, lorsque les stagiaires améliorent leurs compétences, leur fournir une rétroaction qui identifie la raison de l'amélioration des compétences chirurgicales devient problématique car les vidéos enregistrées présentent une durée différente et ne sont pas parfaitement alignées. Bien que la synchronisation des vidéos ait été le centre d'intérêt de plusieurs sites de recherche en vision par ordinateur, les contributions se concentrent généralement sur un cas particulier où plusieurs vidéos enregistrées simultanément (avec des caractéristiques différentes telles que les angles de vue et les facteurs de zoom) sont traitées (Wolf and Zomet, 2002; Wedge, Kovesi, and Huynh, 2005; Padua et al., 2010). Un autre type de synchronisation de multiples vidéos utilise des fonctionnalités conçues à la main (telles que des trajectoires de points d'intérêt) à partir des vidéos (Wang et al., 2014; Evangelidis and Bauckhage, 2011), ce qui rend l'approche très sensible à la qualité des caractéristiques extraites. Ce type de techniques était très efficace car les vidéos brutes étaient la seule source d'information disponible, alors que dans notre cas,

l'utilisation de systèmes chirurgicaux robotisés permet de capturer un type de données supplémentaire: les variables cinématiques telles que les coordonnées cartésiennes x, y, z des effecteurs terminaux du Da Vinci (Gao et al., 2014). Dans cette deuxième partie du chapitre, nous proposons de tirer parti de l'aspect séquentiel des données cinématiques enregistrées par le système chirurgical Da Vinci, afin de synchroniser leurs images vidéo correspondantes en alignant les données de séries temporelles. Lors de l'alignement de deux séries temporelles, l'algorithme standard est DTW (Sakoe and Chiba, 1978) que nous avons en effet utilisé pour aligner deux vidéos. Cependant, lors de l'alignement de plusieurs séquences, cette dernière technique ne se généralise pas de manière simple et réalisable par calcul (Petitjean et al., 2014). Par conséquent, pour la synchronisation de multiples vidéos, nous proposons d'aligner leurs séries temporelles correspondantes avec une série temporelle moyenne, calculée en utilisant l'algorithme DBA. Ce processus est appelé NLTS et a été initialement proposé pour trouver l'alignement multiple d'un ensemble de gestes chirurgicaux discrétisés (Forestier et al., 2014), que nous étendons dans ce travail à des données cinématiques numériques continues.

En conclusion, dans ce chapitre, nous avons abordé deux problèmes différents liés aux compétences chirurgicales. Tout d'abord, en concevant un FCN, nous avons pu obtenir des résultats d'état de l'art pour l'évaluation des compétences chirurgicales (classification et régression). Ainsi, nous avons pu atténuer l'effet de boîte noire des DNNs, en utilisant la technique CAM afin de mettre en évidence ce qui permet d'identifier l'expérience du chirurgien. Le deuxième problème lié aux compétences chirurgicales était dû au fait que les vidéos de formation chirurgicale n'étaient pas synchronisées, ce qui rend difficile pour les stagiaires de comprendre et de comparer les vidéos entre différents chirurgiens avec différents niveaux de compétence. Nous avons abordé ce dernier problème en proposant l'utilisation de NLTS afin d'aligner et de synchroniser plusieurs vidéos simultanément. Ces deux projets étaient orthogonaux dans le sens où ils pouvaient également se compléter: la synchronisation de la vidéo et des séries temporelles pourrait être une étape de prétraitement qui améliorerait les modèles d'évaluation des compétences chirurgicales.

Introduction

Time series data are omnipresent in many practical data science applications ranging from health care (Gao et al., 2014) and stock market predictions (Anghinoni et al., 2018) to social media analysis (Xu, Chen, and Mao, 2018) and human activity recognition (Xi et al., 2018). In fact, any type of numerical acquisition of data with some notion of ordering will generate time series, making this type of data very common among data mining problems (Längkvist, Karlsson, and Loutfi, 2014). Compared to traditional tabular data, each time series can be represented (under the tabular format) as a row with each attribute corresponding to one time stamp (numerical acquisition). However, analyzing time series data differs significantly from its tabular counterpart, as harnessing the temporal information is usually very important for the underlying task we are trying to solve (Bagnall et al., 2017). A concrete example of time series data in health care would be the acquisition of ECG heart signals (Rajan and Thiagarajan, 2018). In stock market analysis, a time series element would correspond to the value of a stock at a given time stamp (Anghinoni et al., 2018). In human activity recognition, the given Cartesian position of a hand in 3D space would constitute an element of a time series (Ignatov, 2018).

Since 2006, time series analysis has been considered one of the most challenging problems in data mining (Yang and Wu, 2006), and in a more recent poll it has been shown that 48% of data expert had analyzed time series data during their career, ahead of text and images (Neamtu et al., 2018). Under the time series analysis umbrella, there exists many orthogonal tasks, that can be grouped into four main categories:

Forecasting consists of training a model using some historical time series data, with the goal to predict the future observations of this time series (Hyndman and Athanasopoulos, 2018). Weather forecasting is one of the most common applications, where the training data consists of old observations, and the task is to predict the future weather behavior (Taylor, McSharry, and Buizza, 2009). In finance, predicting the value of stock using its historical values is one very common application (Kim, 2003). Further examples and details of time series forecasting an be found in Hyndman and Athanasopoulos, 2018.

Anomaly detection is a very special task of time series analysis. Unlike forecasting, the goal is not to predict the future, but rather to determine if a given time series observation is normal or not (Blázquez-García et al., 2020). This task is also known as time series outlier detection. One of the most common application is called predictive maintenance, such as predicting anomalies in advance in order to prevent potential failures (Rabatel, Bringay, and Poncelet, 2011). The reader is referred to this excellent review by Blázquez-García et al., 2020 on time series anomaly/outlier detection.

Clustering is probably one of the most widely studied problems in unsupervised learning. The problem can be defined as follows: given a set of data points, partition them into a set of groups which are as similar as possible (Aggarwal, 2014). For time series data, traditional clustering approaches can provide a baseline, however many

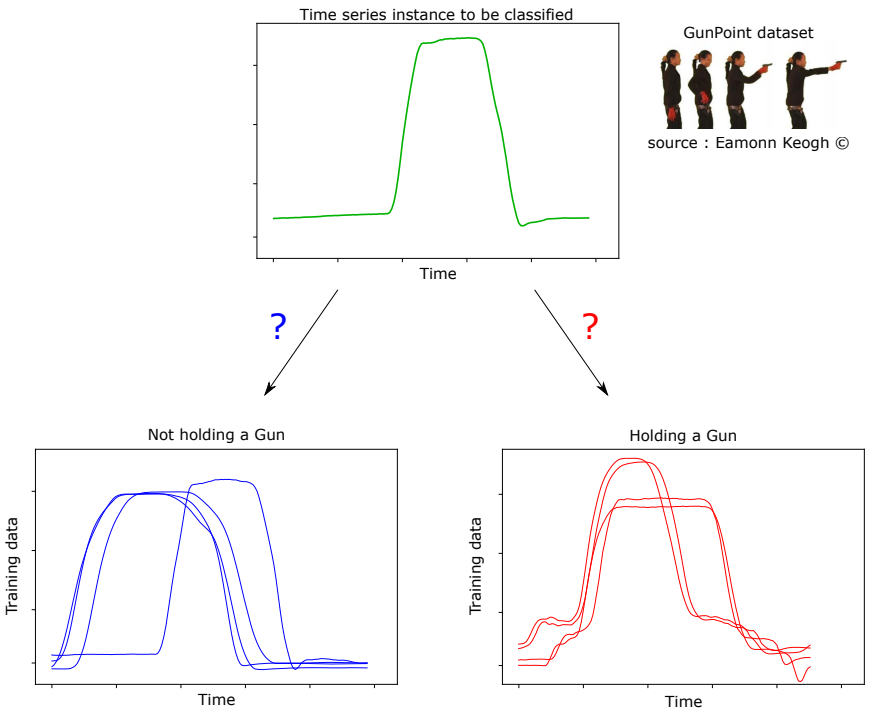


FIGURE 1: An example illustrating the task of classifying an input time series from the GunPointAgeSpan dataset (Dau et al., 2019), where the goal is to classify whether or not a person is holding a gun (a time series corresponds to the hand's x coordinate of a person holding or not a gun)

researchers found that mining the temporal information provided by the time series data can be crucial for many tasks (Petitjean and Gançarski, 2012). Applications range from discovering daily patterns of sales in marketing databases (Sanwlani and Vijayalakshmi, 2013) to finding particular behaviors of solar magnetic wind in scientific databases (Pravilovic et al., 2014). For more details on time series clustering, we refer the interested reader to this recent survey by Aghabozorgi, Shirkhorshidi, and Wah, 2015.

Classification consists of predicting the correct class of a given data point using a labeled training set. For TSC, this data point is by itself a whole time series, and the task consists of predicting its correct label. This problem is encountered in various data mining fields such as patient risk identification in health care (Ma, Xiao, and Wang, 2018), malware detection in cyber security (Tobiyama et al., 2016), food safety evaluation in agriculture and livestock (Nawrocka and Lamorska, 2013). Similar to unsupervised learning, traditional classification approaches can provide a basic baseline for solving this underlying TSC task. However over the past two decades, research has shown that designing algorithms that can exploit the temporal information is a need in order to achieve high classification accuracy (Bagnall et al., 2017). Figure 1 illustrates an example of the TSC task.

In this thesis, we chose to focus on the latter TSC problem. The reason behind choosing this field of time series analysis was motivated by our interest in a very particular problem: surgical skills evaluation from kinematic data. This specific task can be cast as a TSC problem: given an input time series (kinematic data registered through time) predict the correct label (skill level of the surgeon performing the surgery). The reader can find more details on this problem in Chapter 4. For

solving this surgical skills evaluation problem, we started looking into the state-of-the-art TSC algorithms of that epoch. At the beginning of this thesis in 2017, we found that most TSC approaches were inspired by traditional machine learning algorithms such as NN classifiers coupled with a bespoke distance - such as DTW instead of ED. Some algorithms were inspired by traditional text mining approaches such as the Time Series Bag-Of-Features developed by Baydogan, Runger, and Tuv, 2013. Other TSC methods were inspired by Fourier analysis from signal processing such as BOSS (proposed by Schäfer, 2015). One type of classifier focused on extracting discriminative subsequence from the time series called Shapelets, which are later used for classifying the input time series (Hills et al., 2014). Meanwhile in 2017, the computer vision community has already achieved tremendous human level performance with DNNs, followed by the much more recent success of deep learning for various NLP tasks. However, we have noticed that the TSC community have not considered DNNs as potential classifiers of time series data, which is evident in the great TSC bake-off paper (Bagnall et al., 2017). This is very surprising as given the success of deep learning with image classification problems, coupled with the intrinsic similarity between 2D patterns in images and 1D patterns in series, one should consider the potential of deep learning for TSC problems. We therefore started investigating and benchmarking the recent work proposing the use of DNNs for classifying time series data, which is the main focus of Chapter 1 (Ismail Fawaz et al., 2019d).

Following this thorough review of the recent advances in deep learning architectures for TSC, we started looking into the different techniques to improve the accuracy of a given neural network model. This type of technique is also known as regularization, which enables us to improve the generalization capabilities of a given machine learning model. These methods range from transfer learning (Ismail Fawaz et al., 2018d) and ensembling (Ismail Fawaz et al., 2019e) to data augmentation (Ismail Fawaz et al., 2018b) and adversarial training (Ismail Fawaz et al., 2019b). The latter techniques were the main focus of Chapter 2 of this thesis.

Having identified the current state-of-the-art architectures for TSC in Chapter 1, followed by the main techniques on how to improve the generalization capability of a given deep learning model in Chapter 2, we took a further step into designing a new type of neural network architecture for TSC based on the famous Inception module proposed by Szegedy et al., 2015. Although we achieved similar results to other non deep learning based approaches for TSC, the main focus of Chapter 3 was to motivate the use of the Inception module, with a focus on its running time, a severe bottleneck of the current state-of-the-art algorithm for TSC (Bagnall et al., 2017).

Finally, with the recent advances in deep learning for TSC being presented in the first three chapters, we turned our attention in Chapter 4 to our initial motivation: evaluating surgical skills from kinematic data using DNNs. We also focused on achieving state-of-the-art results while providing interpretability of our deep learning model, allowing us to leverage the high accuracy from DNNs while mitigating their black-box effect.

In order to have a thorough and fair experimental evaluation of all approaches, we used the whole UCR/UEA archive (Dau et al., 2019) which contained 85 univariate time series datasets at that time. Other than the fact of being publicly available, the choice of validating on the UCR/UEA archive is motivated by having datasets from different domains which have been broken down into seven different categories (Image Outline, Sensor Readings, Motion Capture, Spectrographs, ECG, Electric Devices and Simulated Data) in Bagnall et al., 2017. In fact, the UCR/UEA

Paper	Method(s)	Data (in paper)	Data (on GitHub)	Chap.	GitHub
(Ismail Fawaz et al., 2019d)	9 methods	UCR/UEA (85)	UCR (128)	1	dl-4-tsc
(Ismail Fawaz et al., 2018d)	FCN	UCR (85)	UCR (85)	2	bigdata18
(Ismail Fawaz et al., 2019e)	6 methods	UCR/UEA (85)	UCR/UEA (85)	2	ijcnn19ensemble
(Ismail Fawaz et al., 2018b)	ResNet	UCR (85)	UCR (85)	2	aaltd18
(Ismail Fawaz et al., 2019b)	ResNet	UCR (85)	UCR (85)	2	ijcnn19attacks
(Ismail Fawaz et al., 2020)	InceptionTime	UCR/UEA (85)	UCR (128)	3	InceptionTime
(Ismail Fawaz et al., 2018c)	FCN	JIGSAWS	JIGSAWS	4	miccai18
(Ismail Fawaz et al., 2019a)	FCN	JIGSAWS	JIGSAWS	4	ijcars19
(Ismail Fawaz et al., 2019c)	NLTS	JIGSAWS	JIGSAWS	4	aime19

TABLE 1: List of papers with the corresponding public datasets used as well as the companion GitHub repository.

archive has evolved over the years. In 2015 the benchmark contained 44 datasets, then in 2017 UCR and UEA collaborated and provided the community with an even larger archive containing 85 datasets, which represents the version that we have used in this work. In 2018, Dau et al., 2019 published the most recent version of the archive containing 128 datasets. We did our best to be consistent in the datasets used for our experiments to make our results easily comparable to other state-of-the-art methods. The evolution of the UCR during this thesis can bring some sort of confusion in the datasets used in our different papers. Therefore Table 1 has the objective to clarify what has been used when our papers were published and what is now available on the companion GitHub pages. In any case, the source code for each paper is available to the community and can be easily reused to follow further evolution of the archive or other challenges. We are aware of the limitations of using the UCR/UEA archive as a sole reference to compare methods and we share the thoughts about this issue discussed by the authors in Dau et al., 2019. However, we believe that having a way to objectively compare the methods between them is very important and we are thankful to all the people involved in making the UCR/UEA archive publicly available. As for the experiments regarding the surgical evaluation skills, we have used the publicly available JIGSAWS dataset published in Gao et al., 2014.

Chapter 1

The state of the art for time series classification

1.1 Introduction

During the last two decades, TSC has been considered as one of the most challenging problems in data mining (Yang and Wu, 2006; Esling and Agon, 2012). With the increase of temporal data availability (Silva et al., 2018), hundreds of TSC algorithms have been proposed since 2015 (Bagnall et al., 2017). Due to their natural temporal ordering, time series data are present in almost every task that requires some sort of human cognitive process (Längkvist, Karlsson, and Loutfi, 2014). In fact, any classification problem, using data that is registered taking into account some notion of ordering, can be cast as a TSC problem (Cristian Borges Gamboa, 2017). Time series are encountered in many real-world applications ranging from health care (Gogolou et al., 2018) and human activity recognition (Wang et al., 2018; Mathis, Ismail Fawaz, and Khamis, 2020) to acoustic scene classification (Nwe, Dat, and Ma, 2017) and cyber security (Susto, Cenedese, and Terzi, 2018). In addition, the diversity of the datasets' types in the UCR/UEA archive (Dau et al., 2019; Bagnall et al., 2017) (the largest repository of time series datasets) shows the different applications of the TSC problem.

Given the need to accurately classify time series data, researchers have proposed hundreds of methods to solve this task (Bagnall et al., 2017). One of the most popular and traditional TSC approaches is the use of an NN classifier coupled with a distance function (Lines and Bagnall, 2015). Particularly, DTW when used with an NN classifier has been shown to be a very strong baseline (Bagnall et al., 2017). Lines and Bagnall, 2015 compared several distance measures - such as TWE (Marteau, 2009) and MSM (Stefan, Athitsos, and Das, 2013) - showing that there is no single distance measure that significantly outperforms DTW. They also showed that ensembling the individual NN classifiers (with different distance measures) outperforms all of the ensemble's individual components. Hence, recent contributions have focused on developing ensembling methods that significantly outperforms the NN-DTW (Bagnall et al., 2016; Hills et al., 2014; Bostrom and Bagnall, 2015; Lines, Taylor, and Bagnall, 2016; Schäfer, 2015; Kate, 2016; Deng et al., 2013; Baydogan, Runger, and Tuv, 2013). These approaches use either an ensemble of decision trees (random forest) (Baydogan, Runger, and Tuv, 2013; Deng et al., 2013) or an ensemble of different types of discriminant classifiers (SVM, NN with several distances) on one or several feature spaces (Bagnall et al., 2016; Bostrom and Bagnall, 2015; Schäfer, 2015; Kate, 2016). Most of these approaches significantly outperform the NN-DTW (Bagnall et al., 2017) and share one common property, which is the data transformation phase where time series are transformed into a new feature space

(for example using shapelets transform (Bostrom and Bagnall, 2015) or DTW features (Kate, 2016)). This notion motivated the development of an ensemble of 35 classifiers named COTE (Bagnall et al., 2016) that does not only ensemble different classifiers over the same transformation, but instead ensembles different classifiers over different time series representations. Lines, Taylor, and Bagnall, 2016; Lines, Taylor, and Bagnall, 2018 extended COTE with HIVE-COTE which has been shown to achieve a significant improvement over COTE by leveraging a new hierarchical structure with probabilistic voting, including two new classifiers and two additional representation transformation domains. HIVE-COTE is currently considered the state-of-the-art algorithm for time series classification (Bagnall et al., 2017) when evaluated over the 85 datasets from the UCR/UEA archive.

To achieve its high accuracy, HIVE-COTE becomes hugely computationally intensive and impractical to run on a real big data mining problem (Bagnall et al., 2017). The approach requires training 37 classifiers as well as cross-validating each hyperparameter of these algorithms, which makes the approach infeasible to train in some situations (Lucas et al., 2018). To emphasize on this infeasibility, note that one of these 37 classifiers is the ST (Hills et al., 2014) whose time complexity is $O(n^2 \cdot l^4)$ with n being the number of time series in the dataset and l being the length of a time series. Adding to the training time's complexity is the high *classification* time of one of the 37 classifiers: the nearest neighbor which needs to scan the training set before taking a decision at test time. Therefore since the nearest neighbor constitutes an essential component of HIVE-COTE, its deployment in a real-time setting is still limited if not impractical. Finally, adding to the huge runtime of HIVE-COTE, the decision taken by 37 classifiers cannot be interpreted easily by domain experts, since researchers already struggle with understanding the decisions taken by an individual classifier. We should note that recently, Bagnall et al., 2020 proposed a new version of HIVE-COTE that is substantially faster, showing the importance of having a scalable TSC algorithm.

After having established the current state-of-the-art of non deep classifiers for TSC (Bagnall et al., 2017), we discuss the success of deep learning (LeCun, Bengio, and Hinton, 2015) in various classification tasks which motivated the recent utilization of deep learning models for TSC (Wang, Yan, and Oates, 2017). Deep CNNs have revolutionized the field of computer vision (Krizhevsky, Sutskever, and Hinton, 2012). For example, in 2015, CNNs were used to reach human level performance in image recognition tasks (Szegedy et al., 2015). Following the success of DNNs in computer vision, a huge amount of research proposed several DNN architectures to solve many NLP tasks such as machine translation (Sutskever, Vinyals, and Le, 2014; Bahdanau, Cho, and Bengio, 2015), learning word embeddings (Mikolov et al., 2013; Mikolov et al., 2013) and document classification (Le and Mikolov, 2014; Goldberg, 2016). DNNs also had a huge impact on the speech recognition community (Hinton et al., 2012; Sainath et al., 2013). Interestingly, we should note that the intrinsic similarity between the NLP and speech recognition tasks is due to the sequential aspect of the data which is also one of the main characteristics of time series data.

In this context, this chapter targets the following open questions: *What is the current state-of-the-art DNN for TSC? Is there a current DNN approach that reaches state-of-the-art performance for TSC and is less complex than HIVE-COTE? What type of DNN architectures works best for the TSC task? How does the random initialization affect the performance of deep learning classifiers?* And finally: *Could the black-box effect of DNNs be avoided to provide interpretability?* Given that the latter questions have not been addressed by the TSC community, it is surprising how a small amount of papers have considered DNNs to be a potential accurate classifier of time series data (Lines,

Taylor, and Bagnall, 2018). In fact, a recent empirical study (Bagnall et al., 2017) evaluated 18 TSC algorithms on 85 time series datasets, none of which was a deep learning model. This shows how much the community lacks of an overview of the current performance of deep learning models for solving the TSC problem (Lines, Taylor, and Bagnall, 2018).

In this chapter, we performed an empirical comparative study of the most recent deep learning approaches for TSC. With the rise of GPUs, we show how deep architectures can be trained efficiently to learn hidden discriminative features from raw time series in an end-to-end manner. Similarly to Bagnall et al., 2017, in order to have a fair comparison between the tested approaches, we developed a common framework in Python, Keras (Chollet, 2015) and Tensorflow (Abadi et al., 2015) to train the deep learning models on a cluster of more than 60 GPUs.

In addition to the univariate datasets' evaluation, we tested the approaches on 12 MTS datasets (Baydogan, 2015). The multivariate evaluation shows another benefit of deep learning models, which is the ability to handle the curse of dimensionality (Bellman, 2010; Keogh and Mueen, 2017) by leveraging different degrees of smoothness in compositional function (Poggio et al., 2017) as well as the parallel computations of the GPUs (Lu et al., 2015).

As for comparing the classifiers over multiple datasets, we followed the recommendations in Demšar, 2006 and used the Friedman test (Friedman, 1940) to reject the null hypothesis. Once we have established that a statistical difference exists within the classifiers' performance, we followed the pairwise post-hoc analysis recommended by Benavoli, Corani, and Mangili, 2016 where the average rank comparison is replaced by a Wilcoxon signed-rank test (Wilcoxon, 1945) with Holm's alpha correction (Holm, 1979; Garcia and Herrera, 2008).

In this study, we have trained about 1 billion parameters across 97 univariate and multivariate time series datasets. Despite the fact that a huge number of parameters risks overfitting (Zhang et al., 2017) the relatively small train set in the UCR/UEA archive, our experiments showed that not only DNNs are able to significantly outperform the NN-DTW, but are also able to achieve results that are *not significantly* different than COTE and HIVE-COTE using a deep residual network architecture (He et al., 2016; Wang, Yan, and Oates, 2017). Finally, we analyze how poor random initializations can have a significant effect on a DNN's performance.

The rest of the chapter is structured as follows. In Section 1.2, we provide some background materials concerning the main types of architectures that have been proposed for TSC. In Section 1.3, the tested architectures are individually presented in details. We describe our experimental open source framework in Section 1.4. The corresponding results and the discussions are presented in Section 1.5. In Section 1.6, we describe in detail a couple of methods that mitigate the black-box effect of the deep learning models. Finally, we present a conclusion in Section 1.7 to summarize our findings and discuss future directions.

The main contributions presented in this chapter can be summarized as follows:

- We explain with practical examples, how deep learning can be adapted to one dimensional time series data.
- We propose a unified taxonomy that regroups the recent applications of DNNs for TSC in various domains under two main categories: generative and discriminative models.
- We detail the architecture of nine end-to-end deep learning models designed specifically for TSC.

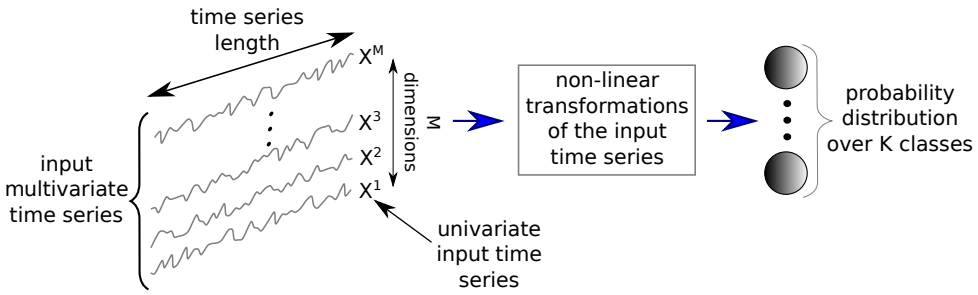


FIGURE 1.1: A unified deep learning framework for time series classification.

- We evaluate these models on the univariate UCR/UEA archive benchmark and 12 MTS classification datasets.
- We provide the community with an open source deep learning framework for TSC in which we have implemented all nine approaches.
- We investigate the use of CAM in order to reduce DNNs' black-box effect and explain the different decisions taken by various models.

1.2 Background

In this section, we start by introducing the necessary definitions for ease of understanding. We then follow by an extensive theoretical background on training DNNs for the TSC task. Finally we present our proposed taxonomy of the different DNNs with examples of their application in various real world data mining problems.

1.2.1 Time series classification

Before introducing the different types of neural networks architectures, we go through some formal definitions for TSC.

Definition 1. A univariate time series $X = [x_1, x_2, \ldots, x_T]$ is an ordered set of real values. The length of X is equal to the number of real values T.

Definition 2. An M-dimensional MTS, $X = [X^1, X^2, \ldots, X^M]$ consists of M different univariate time series with $X^i \in \mathbb{R}^T$.

Definition 3. A dataset $D = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\}$ is a collection of pairs (X_i, Y_i) where X_i could either be a univariate or multivariate time series with Y_i as its corresponding one-hot label vector. For a dataset containing K classes, the one-hot label vector Y_i is a vector of length K where each element $j \in [1, K]$ is equal to 1 if the class of X_i is j and 0 otherwise.

The task of TSC consists of training a classifier on a dataset D in order to map from the space of possible inputs to a probability distribution over the class variable values (labels).

1.2.2 Deep learning approaches for time series classification

In this chapter, we focus on reviewing various approaches tackling the TSC task (Bagnall et al., 2017) using DNNs, which are considered complex machine learning models (LeCun, Bengio, and Hinton, 2015). A general deep learning framework

for TSC is depicted in Figure 1.1. These networks are designed to learn hierarchical representations of the data. A deep neural network is a composition of L parametric functions referred to as layers where each layer is considered a representation of the input domain (Papernot and McDaniel, 2018). One layer l_i, such as $i \in 1 \dots L$, contains neurons, which are small units that compute one element of the layer's output. The layer l_i takes as input the output of its previous layer l_{i-1} and applies a non-linearity (such as the sigmoid function) to compute its own output. The behavior of these non-linear transformations is controlled by a set of parameters θ_i for each layer. In the context of DNNs, these parameters are called weights which link the input of the previous layer to the output of the current layer. Hence, given an input x, a neural network performs the following computations to predict the class:

$$f_L(\theta_L, x) = f_{L-1}(\theta_{L-1}, f_{L-2}(\theta_{L-2}, \dots, f_1(\theta_1, x))) \tag{1.1}$$

where f_i corresponds to the non-linearity applied at layer l_i. For simplicity, we will omit the vector of parameters θ and use $f(x)$ instead of $f(\theta, x)$. This process is also referred to as *feed-forward* propagation in the deep learning literature.

During training, the network is presented with a certain number of known input-output (for example a dataset D). First, the weights are initialized randomly (LeCun et al., 1998), although a robust alternative would be to take a pre-trained model on a source dataset and fine-tune it on the target dataset (Pan and Yang, 2010). This process is known as transfer learning which we do not study empirically, rather we discuss the transferability of each model with respect to the architecture in Section 1.3. After the weight's initialization, a forward pass through the model is applied: using the function f the output of an input x is computed. The output is a vector whose components are the estimated probabilities of x belonging to each class. The model's prediction loss is computed using a cost function, for example the negative log likelihood. Then, using gradient descent (LeCun et al., 1998), the weights are updated in a backward pass to propagate the error. Thus, by iteratively taking a forward pass followed by backpropagation, the model's parameters are updated in a way that minimizes the loss on the training data.

During testing, the probabilistic classifier (the model) is tested on unseen data which is also referred to as the inference phase: a forward pass on this unseen input followed by a class prediction. The prediction corresponds to the class whose probability is maximum. To measure the performance of the model on the test data (generalization), we adopted the accuracy measure (similarly to Bagnall et al., 2017). One advantage of DNNs over non-probabilistic classifiers (such as NN-DTW) is that a probabilistic decision is taken by the network (Large, Lines, and Bagnall, 2017), thus allowing to measure the confidence of a certain prediction given by an algorithm.

Although there exist many types of DNNs, in this review we focus on three main DNN architectures used for the TSC task: MLP, CNN and ESN. These three types of architectures were chosen since they are widely adopted for end-to-end deep learning (LeCun, Bengio, and Hinton, 2015) models for TSC.

Multi Layer Perceptrons

An MLP constitutes the simplest and most traditional architecture for deep learning models. This form of architecture is also known as an FC network since the neurons in layer l_i are connected to every neuron in layer l_{i-1} with $i \in [1, L]$. These connections are modeled by the weights in a neural network. A general form of applying a

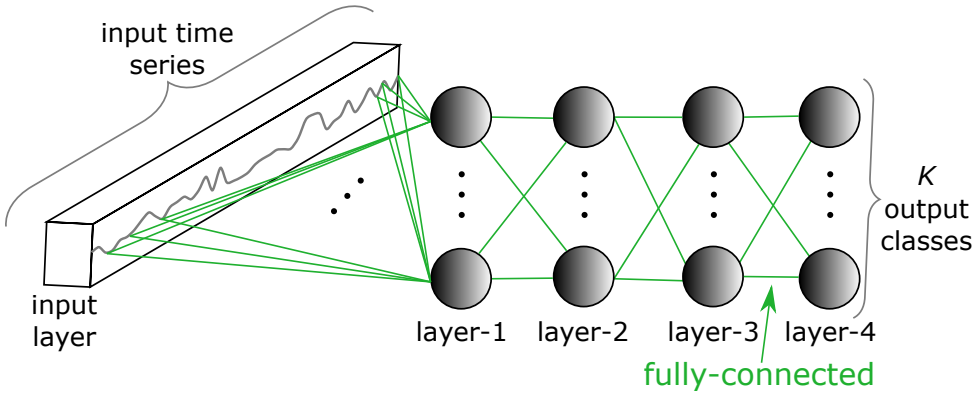


FIGURE 1.2: Multilayer perceptron for time series classification.

non-linearity to an input time series X can be seen in the following equation:

$$A_{l_i} = f(\omega_{l_i} * X + b) \tag{1.2}$$

with ω_{l_i} being the set of weights with length and number of dimensions identical to X's, b the bias term and A_{l_i} the activation of the neurons in layer l_i. Note that the number of neurons in a layer is considered a hyperparameter.

One impediment from adopting MLPs for time series data is that they do not exhibit any spatial invariance, which can be seen in Figure 1.2. In other words, each time stamp has its own weight and the temporal information is lost: meaning time series elements are treated independently from each other. For example the set of weights w_d of neuron d contains $T \times M$ values denoting the weight of each time stamp t for each dimension of the M-dimensional input MTS of length T. Then by cascading the layers we obtain a computation graph similar to equation 1.1.

For TSC, the final layer is usually a discriminative layer that takes as input the activation of the previous layer and gives a probability distribution over the class variables in the dataset. Most deep learning approaches for TSC employ a softmax layer which corresponds to an FC layer with softmax as activation function f and a number of neurons equal to the number of classes in the dataset. Three main useful properties motivate the use of the softmax activation function: the sum of probabilities is guaranteed to be equal to 1, the function is differentiable and it is an adaptation of logistic regression to the multinomial case. The result of a softmax function can be defined as follows:

$$\hat{Y}_j(X) = \frac{e^{A_{L-1} * \omega_j + b_j}}{\sum_{k=1}^{K} e^{A_{L-1} * \omega_k + b_k}} \tag{1.3}$$

with $\hat{Y}_j$ denoting the probability of X having the class Y equal to class j out of K classes in the dataset. The set of weights w_j (and the corresponding bias b_j) for each class j are linked to each previous activation in layer l_{L-1}.

The weights in equations (1.2) and (1.3) should be learned automatically using an optimization algorithm that minimizes an objective cost function. In order to approximate the error of a certain given value of the weights, a differentiable cost (or loss) function that quantifies this error should be defined. The most used loss function in DNNs for the classification task is the categorical cross entropy as defined

in the following equation:

$$L(X) = -\sum_{j=1}^{K} Y_j \log \hat{Y}_j \tag{1.4}$$

with L denoting the loss or cost when classifying the input time series X. Similarly, the average loss when classifying the whole training set of D can be defined using the following equation:

$$J(\Omega) = \frac{1}{N} \sum_{n=1}^{N} L(X_n) \tag{1.5}$$

with Ω denoting the set of weights to be learned by the network (in this case the weights w from equations 1.2 and 1.3). The loss function is minimized to learn the weights in Ω using a gradient descent method which is defined using the following equation:

$$\omega = \omega - \alpha \frac{\partial J}{\partial \omega} \mid \forall\, \omega \in \Omega \tag{1.6}$$

with α denoting the learning rate of the optimization algorithm. By subtracting the partial derivative, the model is actually auto-tuning the parameters ω in order to reach a local minimum (or a saddle point) of J in case of a non-linear classifier (which is almost always the case for a DNN). We should note that when the partial derivative cannot be directly computed with respect to a certain parameter ω, the chain rule of derivative is employed which is in fact the main idea behind the backpropagation algorithm (LeCun et al., 1998).

Convolutional Neural Networks

Since AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) won the ImageNet competition in 2012, deep CNNs have seen a lot of successful applications in many different domains (LeCun, Bengio, and Hinton, 2015) such as reaching human level performance in image recognition problems (Szegedy et al., 2015) as well as different natural language processing tasks (Sutskever, Vinyals, and Le, 2014; Bahdanau, Cho, and Bengio, 2015). Motivated by the success of these CNN architectures in these various domains, researchers have started adopting them for time series analysis (Cristian Borges Gamboa, 2017).

A convolution can be seen as applying and sliding a filter over the time series. Unlike images, the filters exhibit only one dimension (time) instead of two dimensions (width and height). The filter can also be seen as a generic non-linear transformation of a time series. Concretely, if we are convoluting (multiplying) a filter of length 3 with a univariate time series, by setting the filter values to be equal to $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, the convolution will result in applying a moving average with a sliding window of length 3. A general form of applying the convolution for a centered time stamp t is given in the following equation:

$$C_t = f(\omega * X_{t-l/2:t+l/2} + b) \mid \forall\, t \in [1, T] \tag{1.7}$$

where C denotes the result of a convolution (dot product $*$) applied on a univariate time series X of length T with a filter ω of length l, a bias parameter b and a final non-linear function f such as the ReLU. The result of a convolution (one filter) on an input time series X can be considered as another univariate time series C that underwent a filtering process. Thus, applying several filters on a time series will result in a multivariate time series whose dimensions are equal to the number of

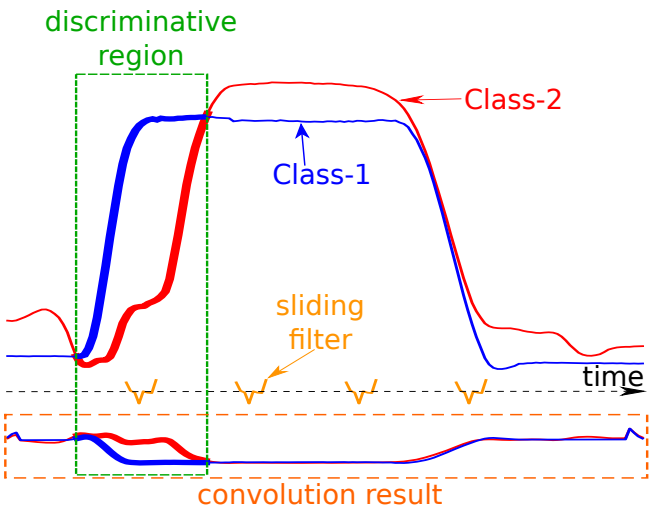


FIGURE 1.3: The result of a applying a learned discriminative convolution on the GunPoint dataset.

filters used. An intuition behind applying several filters on an input time series would be to learn multiple discriminative features useful for the classification task.

Unlike MLPs, the same convolution (the same filter values w and b) will be used to find the result for all time stamps $t \in [1, T]$. This is a very powerful property (called weight sharing) of the CNNs which enables them to learn filters that are invariant across the time dimension.

When considering an MTS as input to a convolutional layer, the filter no longer has one dimension (time) but also has dimensions that are equal to the number of dimensions of the input MTS. Thus, the filter can be considered to be multivariate itself.

Finally, instead of setting manually the values of the filter ω, these values should be learned automatically since they depend highly on the targeted dataset. For example, one dataset would have the optimal filter to be equal to $[1, 2, 2]$ whereas another dataset would have an optimal filter equal to $[2, 0, -1]$. By *optimal* we mean a filter whose application will enable the classifier to easily discriminate between the dataset classes (see Figure 1.3). In order to learn automatically a discriminative filter, the convolution should be followed by a discriminative classifier, which is usually preceded by a *pooling* operation that can either be *local* or *global*.

Local pooling such as *average* or *max* pooling takes an input time series and reduces its length T by aggregating over a sliding window of the time series. For example if the sliding window's length is equal to 3 the resulting pooled time series will have a length equal to $\frac{T}{3}$ - this is only true if the stride is equal to the sliding window's length. With a global pooling operation, the time series will be aggregated over the whole time dimension resulting in a single real value. In other words, this is similar to applying a local pooling with a sliding window's length equal to the length of the input time series. Usually a global aggregation is adopted to reduce drastically the number of parameters in a model thus decreasing the risk of overfitting while enabling the use of CAM to explain the model's decision (Zhou et al., 2016).

In addition to pooling layers, some deep learning architectures include normalization layers to help the network converge quickly. For time series data, the batch normalization operation is performed over each channel therefore preventing the internal covariate shift across one mini-batch training of time series (Ioffe and Szegedy,

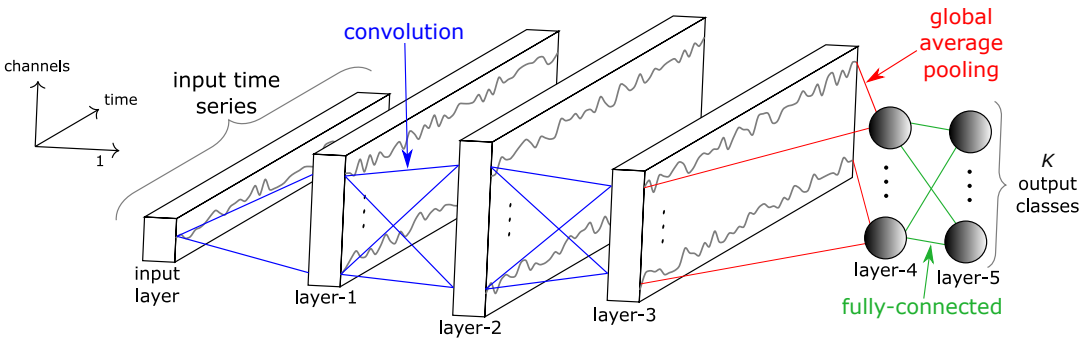


FIGURE 1.4: Fully Convolutional Neural Network architecture.

2015). Another type of normalization was proposed by Ulyanov, Vedaldi, and Lempitsky, 2016 to normalize each instance instead of a per batch basis, thus learning the mean and standard deviation of each training instance for each layer via gradient descent. The latter approach is called instance normalization and mimics learning the z-normalization parameters for the time series training data.

The final discriminative layer takes the representation of the input time series (the result of the convolutions) and give a probability distribution over the class variables in the dataset. Usually, this layer is comprised of a softmax operation similarly to the MLPs. Note that for some approaches, we would have an additional non-linear FC layer before the final softmax layer which increases the number of parameters in a network. Finally in order to train and learn the parameters of a deep CNN, the process is identical to training an MLP: a feed-forward pass followed by backpropagation (LeCun et al., 1998). An example of a CNN architecture for TSC with three convolutional layers is illustrated in Figure 1.4.

Echo State Networks

Another popular type of architectures for deep learning models is the RNN. Apart from time series forecasting, we found that these neural networks were rarely applied for time series classification which is mainly due to three factors: (1) the type of this architecture is designed mainly to predict an output for each element (time stamp) in the time series (Längkvist, Karlsson, and Loutfi, 2014); (2) RNNs typically suffer from the vanishing gradient problem due to training on long time series (Pascanu, Mikolov, and Bengio, 2012); (3) RNNs are considered hard to train and parallelize which led the researchers to avoid using them for computational reasons (Pascanu, Mikolov, and Bengio, 2013).

Given the aforementioned limitations, a relatively recent type of recurrent architecture was proposed for time series: ESNs (Gallicchio and Micheli, 2017). ESNs were first invented by Jaeger and Haas, 2004 for time series prediction in wireless communication channels. They were designed to mitigate the challenges of RNNs by eliminating the need to compute the gradient for the hidden layers which reduces the training time of these neural networks thus avoiding the vanishing gradient problem. These hidden layers are initialized randomly and constitutes the *reservoir*: the core of an ESN which is a sparsely connected random RNN. Each neuron in the reservoir will create its own nonlinear activation of the incoming signal. The inter-connected weights inside the reservoir and the input weights are not learned via gradient descent, only the output weights are tuned using a learning algorithm such as logistic regression or Ridge classifier (Hoerl and Kennard, 1970).

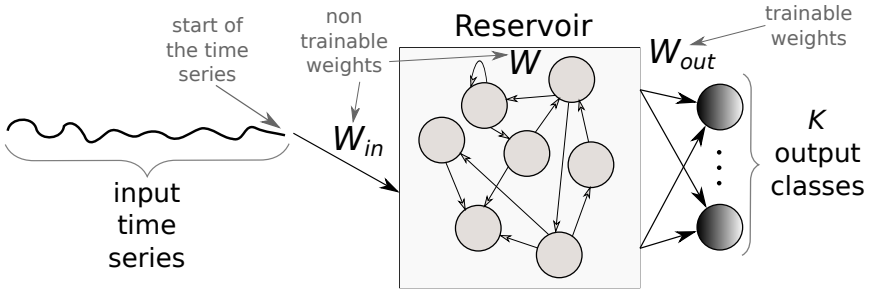


FIGURE 1.5: An Echo State Network architecture for time series classification.

To better understand the mechanism of these networks, consider an ESN with input dimensionality M, neurons in the reservoir N_r and an output dimensionality K equal to the number of classes in the dataset. Let $X(t) \in \mathbb{R}^M$, $I(t) \in \mathbb{R}^{N_r}$ and $\hat{Y}(t) \in \mathbb{R}^K$ denote the vectors of the input M-dimensional MTS, the internal (or hidden) state and the output unit activity for time t respectively. Further let $W_{in} \in \mathbb{R}^{N_r \times M}$ and $W \in \mathbb{R}^{N_r \times N_r}$ and $W_{out} \in \mathbb{R}^{C \times N_r}$ denote respectively the weight matrices for the input time series, the internal connections and the output connections as seen in Figure 1.5. The internal unit activity $I(t)$ at time t is updated using the internal state at time step $t-1$ and the input time series element at time t. Formally the hidden state can be computed using the following recurrence:

$$I(t) = f(W_{in}X(t) + WI(t-1)) \mid \forall\, t \in [1, T] \tag{1.8}$$

with f denoting an activation function of the neurons, a common choice is $tanh(\cdot)$ applied element-wise (Tanisaro and Heidemann, 2016). The output can be computed according to the following equation:

$$\hat{Y}(t) = W_{out}I(t) \tag{1.9}$$

thus classifying each time series element $X(t)$. Note that ESNs depend highly on the initial values of the reservoir that should satisfy a pre-determined hyperparameter: the spectral radius. Figure 1.5 shows an example of an ESN with a univariate input time series to be classified into K classes.

Finally, we should note that for all types of DNNs, a set of techniques was proposed by the deep learning community to enhance neural networks' generalization capabilities. Regularization methods such as $l2$-norm weight decay (Bishop, 2006) or Dropout (Srivastava et al., 2014) aim at reducing overfitting by limiting the activation of the neurons. Another popular technique is data augmentation, which tackles the problem of overfitting a small dataset by increasing the number of training instances (Baird, 1992). This method consists in cropping, rotating and blurring images which have been shown to improve the DNNs' performance for computer vision tasks (Zhang et al., 2017). Although two approaches in this survey include a data augmentation technique, the study of its impact on TSC is currently limited (Ismail Fawaz et al., 2018b). Finally we should note that Chapter 2 of this thesis is dedicated to study several regularization techniques of DNNs for the underlying TSC task.

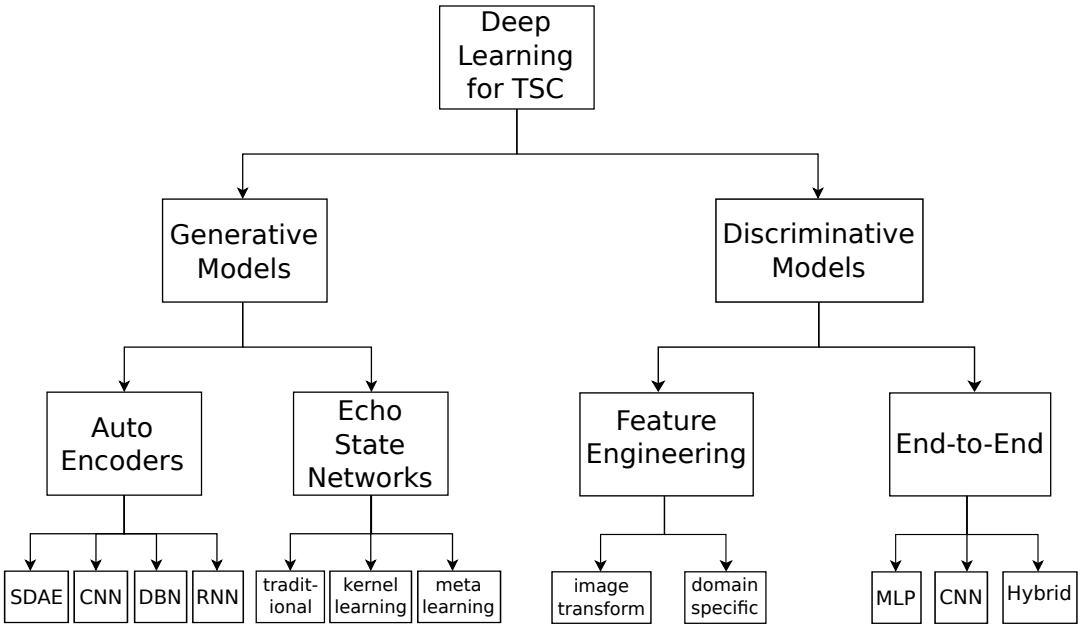


FIGURE 1.6: An overview of the different deep learning approaches for time series classification.

1.2.3 Generative or discriminative approaches

Deep learning approaches for TSC can be separated into two main categories: the *generative* and the *discriminative* models (as proposed in Längkvist, Karlsson, and Loutfi, 2014). We further separate these two groups into sub-groups which are detailed in the following subsections and illustrated in Figure 1.6.

Generative models

Generative models usually exhibit an unsupervised training step that precedes the learning phase of the classifier (Längkvist, Karlsson, and Loutfi, 2014). This type of network has been referred to as *Model-based* classifiers in the TSC community (Bagnall et al., 2017). Some of these generative non deep learning approaches include auto-regressive models (Bagnall and Janacek, 2014), hidden Markov models (Kotsifakos and Papapetrou, 2014) and kernel models (Chen et al., 2013).

For all generative approaches, the goal is to find a good representation of time series prior to training a classifier (Längkvist, Karlsson, and Loutfi, 2014). Usually, to model the time series, classifiers are preceded by an unsupervised pre-training phase such as SDAEs (Bengio et al., 2013; Hu, Zhang, and Zhou, 2016). A generative CNN-based model was proposed in Wang et al., 2016b; Mittelman, 2015 where the authors introduced a deconvolutional operation followed by an upsampling technique that helps in reconstructing a multivariate time series. DBNs were also used to model the latent features in an unsupervised manner which are then leveraged to classify univariate and multivariate time series (Wang et al., 2017; Banerjee et al., 2017). In Mehdiyev et al., 2017; Malhotra et al., 2018; Rajan and Thiagarajan, 2018, an RNN auto-encoder was designed to first generate the time series then using the learned latent representation, they trained a classifier (such as SVM or Random Forest) on top of these representations to predict the class of a given input time series.

Other studies such as in Aswolinskiy, Reinhart, and Steil, 2017; Bianchi et al., 2018; Chouikhi, Ammar, and Alimi, 2018; Ma et al., 2016 used self-predict modeling

for time series classification where ESNs were first used to re-construct the time series and then the learned representation in the reservoir space was utilized for classification. We refer to this type of architecture by traditional ESNs in Figure 1.6. Other ESN-based approaches (Chen et al., 2015; Chen et al., 2013; Che et al., 2017b) define a kernel over the learned representation followed by an SVM or an MLP classifier. In Gong et al., 2018; Wang, Wang, and Liu, 2016, a meta-learning evolutionary-based algorithm was proposed to construct an optimal ESN architecture for univariate and multivariate time series. For more details concerning generative ESN models for TSC, we refer the interested reader to a recent empirical study (Aswolinskiy, Reinhart, and Steil, 2016) that compared classification in reservoir and model-space for both multivariate and univariate time series.

Discriminative models

A discriminative deep learning model is a classifier (or regressor) that directly learns the mapping between the raw input of a time series (or its hand engineered features) and outputs a probability distribution over the class variables in a dataset. Several discriminative deep learning architectures have been proposed to solve the TSC task, but we found that this type of model could be further sub-divided into two groups: (1) deep learning models with hand engineered features and (2) *end-to-end* deep learning models.

The most frequently encountered and computer vision inspired feature extraction method for hand engineering approaches is the transformation of time series into images using specific imaging methods such as Gramian fields (Wang and Oates, 2015b; Wang and Oates, 2015a), recurrence plots (Hatami, Gavet, and Debayle, 2017; Tripathy and Acharya, 2018) and Markov transition fields (Wang and Oates, 2015). Unlike image transformation, other feature extraction methods are not domain agnostic. These features are first hand-engineered using some domain knowledge, then fed to a deep learning discriminative classifier. For example in Uemura et al., 2018, several features (such as the velocity) were extracted from sensor data placed on a surgeon's hand in order to determine the skill level during surgical training. In fact, most of the deep learning approaches for TSC with some hand engineered features are present in human activity recognition tasks (Ignatov, 2018). For more details on the different applications of deep learning for human motion detection using mobile and wearable sensor networks, we refer the interested reader to a recent survey by Nweke et al., 2018 where deep learning approaches (with or without hand engineered features) were thoroughly described specifically for the human activity recognition task.

In contrast to feature engineering, *end-to-end* deep learning aims to incorporate the feature learning process while fine-tuning the discriminative classifier (Nweke et al., 2018). Since this type of deep learning approach is domain agnostic and does not include any domain specific pre-processing steps, we decided to further separate these end-to-end approaches using their neural network architectures.

In Wang, Yan, and Oates, 2017; Geng and Luo, 2018, an MLP was designed to learn from scratch a discriminative time series classifier. The problem with an MLP approach is that temporal information is lost and the features learned are no longer time-invariant. This is where CNNs are most useful, by learning spatially invariant filters (or features) from raw input time series (Wang, Yan, and Oates, 2017). During our study, we found that CNN is the most widely applied architecture for the TSC problem, which is probably due to their robustness and the relatively small amount of training time compared to their counterpart architectures such as RNNs

or MLPs. Several variants of CNNs have been proposed and validated on a subset of the UCR/UEA archive (Dau et al., 2019; Bagnall et al., 2017) such as ResNets (Wang, Yan, and Oates, 2017; Geng and Luo, 2018) which add linear shortcut connections for the convolutional layers potentially enhancing the model's accuracy (He et al., 2016). In Le Guennec, Malinowski, and Tavenard, 2016; Cui, Chen, and Chen, 2016; Wang, Yan, and Oates, 2017; Zhao et al., 2017, traditional CNNs were also validated on the UCR/UEA archive. More recently in Wang et al., 2018, the architectures proposed in Wang, Yan, and Oates, 2017 were modified to leverage a filter initialization technique based on the Daubechies 4 Wavelet values (Rowe and Abbott, 1995). Outside of the UCR/UEA archive, deep learning has reached state-of-the-art performance on several datasets in different domains (Längkvist, Karlsson, and Loutfi, 2014). For spatio-temporal series forecasting problems, such as meteorology and oceanography, DNNs were proposed in Ziat et al., 2017. Strodthoff and Strodthoff, 2019 proposed to detect myocardial infractions from electrocardiography data using deep CNNs. For human activity recognition from wearable sensors, deep learning is replacing the feature engineering approaches (Nweke et al., 2018) where features are no longer hand-designed but rather learned by deep learning models trained through backpropagation. One other type of time series data is present in Electronic Health Records, where a recent generative adversarial network with a CNN (Che et al., 2017a) was trained for risk prediction based on patients historical medical records. In Ismail Fawaz et al., 2018c, CNNs were designed to reach state-of-the-art performance for surgical skills identification. Liu, Hsaio, and Tu, 2018 leveraged a CNN model for multivariate and lag-feature characteristics in order to achieve state-of-the-art accuracy on the Prognostics and Health Management 2015 challenge data. Finally, a recent review of deep learning for physiological signals classification revealed that CNNs were the most popular architecture (Faust et al., 2018) for the considered task. We mention one final type of hybrid architectures that showed promising results for the TSC task on the UCR/UEA archive datasets, where mainly CNNs were combined with other types of architectures such as Gated Recurrent Units (Lin and Runger, 2018) and the attention mechanism (Serrà, Pascual, and Karatzoglou, 2018). The reader may have noticed that CNNs appear under Auto Encoders as well as under End-to-End learning in Figure 1.6. This can be explained by the fact that CNNs when trained as Auto Encoders have a complete different objective function than CNNs that are trained in an end-to-end fashion.

Now that we have presented the taxonomy for grouping DNNs for TSC, we introduce in the following section the different approaches that we have included in our experimental evaluation. We also explain the motivations behind the selection of these algorithms.

1.3 Benchmarking deep learning for time series classification

In this section, we start by explaining the reasons behind choosing discriminative end-to-end approaches for this empirical evaluation. We then describe in detail the nine different deep learning architectures with their corresponding advantages and drawbacks.

1.3.1 Why discriminative end-to-end approaches ?

As previously mentioned in Section 1.2, the main characteristic of a generative model is fitting a time series self-predictor whose latent representation is later fed into an

off-the-shelf classifier such as Random Forest or SVM. Although these models do sometimes capture the trend of a time series, we decided to leave these generative approaches out of our experimental evaluation for the following reasons:

- This type of method is mainly proposed for tasks other than classification or as part of a larger classification scheme (Bagnall et al., 2017);
- The informal consensus in the literature is that generative models are usually less accurate than direct discriminative models (Bagnall et al., 2017; Nguyen, Gsponer, and Ifrim, 2017);
- The implementation of these models is usually more complicated than for discriminative models since it introduces an additional step of fitting a time series generator - this has been considered a barrier with most approaches whose code was not publicly available such as Gong et al., 2018; Che et al., 2017b; Chouikhi, Ammar, and Alimi, 2018; Wang et al., 2017;
- The accuracy of these models depends highly on the chosen off-the-shelf classifier which is sometimes not even a neural network classifier (Rajan and Thiagarajan, 2018).

Given the aforementioned limitations for generative models, we decided to limit our experimental evaluation to discriminative deep learning models for TSC. In addition to restricting the study to discriminative models, we decided to only consider end-to-end approaches, thus further leaving classifiers that incorporate feature engineering out of our empirical evaluation. We made this choice because we believe that the main goal of deep learning approaches is to remove the bias due to manually designed features (Ordóñez and Roggen, 2016), thus enabling the network to learn the most discriminant useful features for the classification task. This has also been the consensus in the human activity recognition literature, where the accuracy of deep learning methods depends highly on the quality of the extracted features (Nweke et al., 2018). Finally, since our goal is to provide an empirical study of domain agnostic deep learning approaches for any TSC task, we found that it is best to compare models that do not incorporate any domain knowledge into their approach.

As for why we chose the nine approaches (described in the next section), it is first because among all the discriminative end-to-end deep learning models for TSC, we wanted to cover a wide range of architectures such as CNNs, Fully CNNs, MLPs, ResNets, ESNs, etc. Second, since we cannot cover an empirical study of all approaches validated in all TSC domains, we decided to only include approaches that were validated on the whole (or a subset of) the univariate time series UCR/UEA archive (Dau et al., 2019; Bagnall et al., 2017) and/or on the MTS archive (Baydogan, 2015). Finally, we chose to work with approaches that do not try to solve a sub task of the TSC problem such as in Geng and Luo, 2018 where CNNs were modified to classify imbalanced time series datasets. To justify this choice, we emphasize that imbalanced TSC problems can be solved using several techniques such as data augmentation (Ismail Fawaz et al., 2018b) and modifying the class weights (Geng and Luo, 2018). However, any deep learning algorithm can benefit from this type of modification. Therefore if we did include modifications for solving imbalanced TSC tasks, it would be much harder to determine if it is the choice of the deep learning classifier or the modification itself that improved the accuracy of the model. Another sub task that has been at the center of recent studies is early time series classification (Wang et al., 2016a) where deep CNNs were modified to include an early classification of time series. More recently, a deep reinforcement learning approach was

also proposed for the early TSC task (Martinez et al., 2018). For further details, we refer the interested reader to a recent survey on deep learning for *early* time series classification (Santos and Kern, 2017).

1.3.2 Compared approaches

After having presented an overview over the recent deep learning approaches for time series classification, we present the nine architectures that we have chosen to compare in this chapter.

Multi Layer Perceptron

The MLP, which depicted in Figure 1.2, is considered the most traditional form of DNNs proposed in Wang, Yan, and Oates, 2017 as a baseline architecture for TSC. The network contains 4 layers in total where each one is fully connected to the output of its previous layer. The final layer is a softmax classifier, which is fully connected to its previous layer's output and contains a number of neurons equal to the number of classes in a dataset. All three hidden FC layers are composed of 500 neurons with ReLU as the activation function. Each layer is preceded by a dropout operation (Srivastava et al., 2014) with a rate equal to 0.1, 0.2, 0.2 and 0.3 for respectively the first, second, third and fourth layer. Dropout is one form of regularization that helps in preventing overfitting (Srivastava et al., 2014). The dropout rate indicates the percentage of neurons that are deactivated (set to zero) in a feed forward pass during training.

MLP does not have any layer whose number of parameters is invariant across time series of different lengths (denoted by *#invar* in Table 1.1) which means that the transferability of the network is not trivial: the number of parameters (weights) of the network depends directly on the length of the input time series.

Fully Convolutional Neural Network

FCNs, illustrated in Figure 1.4, were first proposed in Wang, Yan, and Oates, 2017 for classifying univariate time series and validated on 44 datasets from the UCR/UEA archive. FCNs are mainly convolutional networks that do not contain any local pooling layers which means that the length of a time series is kept unchanged throughout the convolutions. In addition, one of the main characteristics of this architecture is the replacement of the traditional final FC layer with a GAP layer which reduces drastically the number of parameters in a neural network while enabling the use of the CAM (Zhou et al., 2016) that highlights which parts of the input time series contributed the most to a certain classification.

The architecture proposed in Wang, Yan, and Oates, 2017 is first composed of three convolutional blocks where each block contains three operations: a convolution followed by a batch normalization (Ioffe and Szegedy, 2015) whose result is fed to a ReLU activation function. The result of the third convolutional block is averaged over the whole time dimension which corresponds to the GAP layer. Finally, a traditional softmax classifier is fully connected to the GAP layer's output.

All convolutions have a stride equal to 1 with a zero padding to preserve the exact length of the time series after the convolution. The first convolution contains 128 filters with a filter length equal to 8, followed by a second convolution of 256 filters with a filter length equal to 5 which in turn is fed to a third and final convolutional layer composed of 128 filters, each one with a length equal to 3.

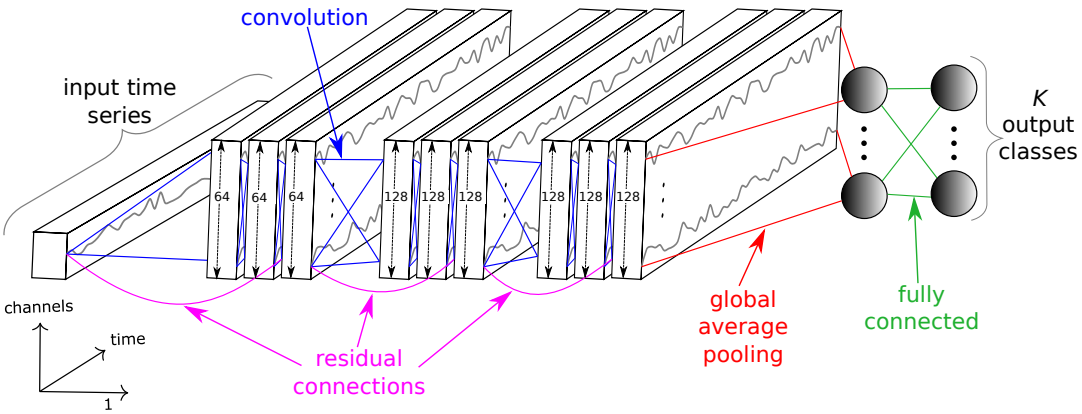


FIGURE 1.7: The Residual Network's architecture for time series classification.

We can see that FCN does not hold any pooling nor a regularization operation. In addition, one of the advantages of FCNs is the invariance (denoted by *#invar* in Table 1.1) in the number of parameters for 4 layers (out of 5) across time series of different lengths. This invariance (due to using GAP) enables the use of a transfer learning approach where one can train a model on a certain source dataset and then fine-tune it on the target dataset (Ismail Fawaz et al., 2018d).

Residual Network

The third and final proposed architecture in Wang, Yan, and Oates, 2017 is a relatively deep ResNet. For TSC, this is the deepest architecture with 11 layers of which the first 9 layers are convolutional followed by a GAP layer that averages the time series across the time dimension. The main characteristic of ResNet is the shortcut residual connection between consecutive convolutional layers. In fact, the difference with the usual convolutions (such as in FCNs) is that a linear shortcut is added to link the output of a residual block to its input thus enabling the flow of the gradient directly through these connections, which makes training a DNN much easier by reducing the vanishing gradient effect (He et al., 2016).

The network is composed of three residual blocks followed by a GAP layer and a final softmax classifier whose number of neurons is equal to the number of classes in a dataset. Each residual block is first composed of three convolutions whose output is added to the residual block's input and then fed to the next layer. The number of filters for all convolutions is fixed to 64, with the ReLU activation function that is preceded by a batch normalization operation. In each residual block, the filter's length is set to 8, 5 and 3 respectively for the first, second and third convolution.

Similarly to the FCN model, the layers (except the final one) in the ResNet architecture have an invariant number of parameters across different datasets. That being said, we can easily pre-train a model on a source dataset, then transfer and fine-tune it on a target dataset without having to modify the hidden layers of the network. As we have previously mentioned and since this type of transfer learning approach can give an advantage for certain types of architecture, we leave the exploration of this area of research for future work. The ResNet architecture proposed by Wang, Yan, and Oates, 2017 is depicted in Figure 1.7.

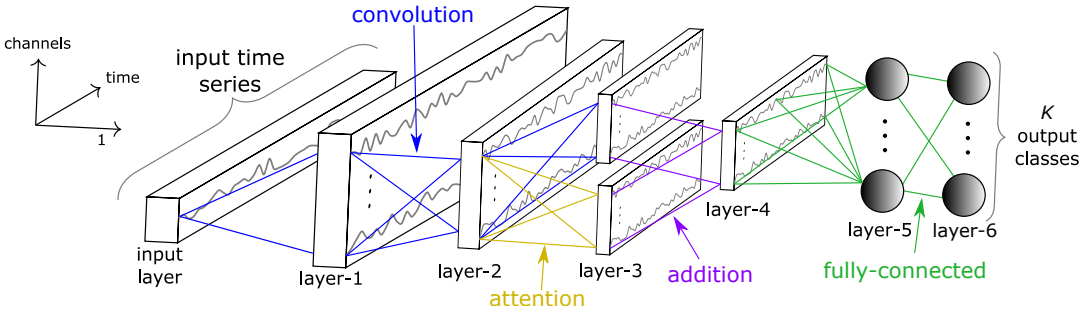


FIGURE 1.8: Encoder's architecture for time series classification.

Encoder

Originally proposed by Serrà, Pascual, and Karatzoglou, 2018, Encoder is a hybrid deep CNN whose architecture is inspired by FCN (Wang, Yan, and Oates, 2017) with a main difference where the GAP layer is replaced with an attention layer. In Serrà, Pascual, and Karatzoglou, 2018, two variants of Encoder were proposed: the first approach was to train the model from scratch in an end-to-end fashion on a target dataset while the second one was to pre-train this same architecture on a source dataset and then fine-tune it on a target dataset. The latter approach reached higher accuracy thus benefiting from the transfer learning technique. On the other hand, since almost all approaches can benefit to certain degree from a transfer learning method, we decided to implement only the end-to-end approach (training from scratch) which already showed high performance in the author's original paper.

Similarly to FCN, the first three layers are convolutional with some relatively small modifications. The first convolution is composed of 128 filters of length 5; the second convolution is composed of 256 filters of length 11; the third convolution is composed of 512 filters of length 21. Each convolution is followed by an instance normalization operation (Ulyanov, Vedaldi, and Lempitsky, 2016) whose output is fed to the PReLU (He et al., 2015) activation function. The output of PReLU is followed by a dropout operation (with a rate equal to 0.2) and a final max pooling of length 2. The third convolutional layer is fed to an attention mechanism (Bahdanau, Cho, and Bengio, 2015) that enables the network to learn which parts of the time series (in the time domain) are important for a certain classification. More precisely, to implement this technique, the input MTS is multiplied with a second MTS of the same length and number of channels, except that the latter has gone through a softmax function. Each element in the second MTS will act as a weight for the first MTS, thus enabling the network to learn the importance of each element (time stamp). Finally, a traditional softmax classifier is fully connected to the latter layer with a number of neurons equal to the number of classes in the dataset.

In addition to replacing the GAP layer with the attention layer, Encoder differs from FCN in three main core changes: (1) the PReLU activation function where an additional parameter is added for each filter to enable learning the slope of the function, (2) the dropout regularization technique and (3) the max pooling operation. One final note is that the careful design of Encoder's attention mechanism enabled the invariance across all layers which encouraged the authors to implement a transfer learning approach. Figure 1.8 illustrates Encoder's architecture for TSC.

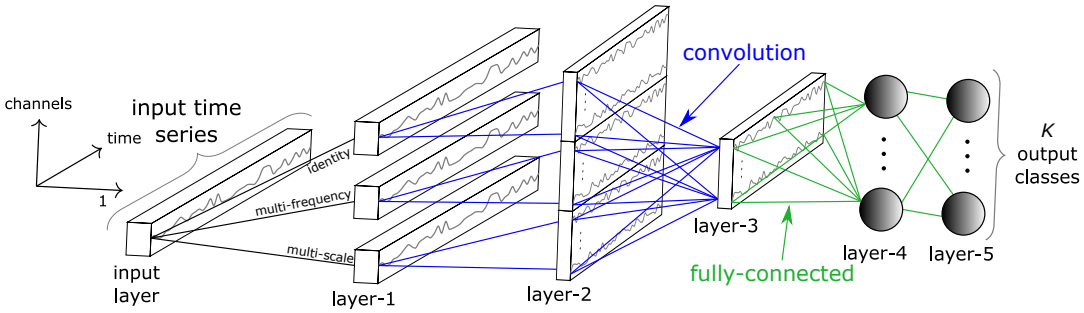


FIGURE 1.9: MCNN's architecture for time series classification.

Multi-scale Convolutional Neural Network

Originally proposed by Cui, Chen, and Chen, 2016, MCNN is the earliest approach to validate an end-to-end deep learning architecture on the UCR Archive. MCNN's architecture is very similar to a traditional CNN model: with two convolutions (and max pooling) followed by an FC layer and a final softmax layer. On the other hand, this approach is very complex with its heavy data pre-processing step. Cui, Chen, and Chen, 2016 were the first to introduce the WS method as a data augmentation technique. WS slides a window over the input time series and extract subsequences, thus training the network on the extracted subsequences instead of the raw input time series. Following the extraction of a subsequence from an input time series using the WS method, a transformation stage is used. More precisely, prior to any training, the subsequence will undergo three transformations: (1) identity mapping; (2) down-sampling and (3) smoothing; thus, transforming a univariate input time series into a multivariate input time series. This heavy pre-processing would question the end-to-end label of this approach, but since their method is generic enough we incorporated it into our developed framework.

For the first transformation, the input subsequence is left unchanged and the raw subsequence will be used as an input for an independent first convolution. The down-sampling technique (second transformation) will result in shorter subsequences with different lengths which will then undergo another independent convolutions in parallel to the first convolution. As for the smoothing technique (third transformation), the result is a smoothed subsequence whose length is equal to the input raw subsequence which will also be fed to an independent convolution in parallel to the first and the second convolutions.

The output of each convolution in the first convolutional stage is concatenated to form the input of the subsequent convolutional layer. Following this second layer, an FC layer is deployed with 256 neurons using the sigmoid activation function. Finally, the usual softmax classifier is used with a number of neurons equal to the number of classes in the dataset.

Note that each convolution in this network uses 256 filters with the sigmoid as an activation function, followed by a max pooling operation. Two architecture hyperparameters are cross-validated, using a grid search on an unseen split from the training set: the filter length and the pooling factor which determines the pooling size for the max pooling operation. The total number of layers in this network is 4, out of which only the first two convolutional layers are invariant (transferable). Finally, since the WS method is also used at test time, the class of an input time series is determined by a majority vote over the extracted subsequences' predicted labels. Figure 1.9 shows the MCNN's architecture for TSC.

Time Le-Net

T-LeNet was originally proposed by Le Guennec, Malinowski, and Tavenard, 2016 and inspired by the great performance of LeNet's architecture for the document recognition task (LeCun et al., 1998). This model can be considered as a traditional CNN with two convolutions followed by an FC layer and a final softmax classifier. There are two main differences with the FCNs: (1) an FC layer and (2) local max-pooling operations. Unlike GAP, local pooling introduces invariance to small perturbations in the activation map (the result of a convolution) by taking the maximum value in a local pooling window. Therefore for a pool size equal to 2, the pooling operation will halve the length of a time series by taking the maximum value between each two time steps.

For both convolutions, the ReLU activation function is used with a filter length equal to 5. For the first convolution, 5 filters are used and followed by a max pooling of length equal to 2. The second convolution uses 20 filters followed by a max pooling of length equal to 4. Thus, for an input time series of length l, the resulting output of these two convolutions will divide the length of the time series by $8 = 4 \times 2$. The convolutional blocks are followed by a non-linear fully connected layer which is composed of 500 neurons, each one using the ReLU activation function. Finally, similarly to all previous architectures, the number of neurons in the final softmax classifier is equal to the number of classes in a dataset.

Unlike ResNet and FCN, this approach does not have much invariant layers (2 out of 4) due to the use of an FC layer instead of a GAP layer, thus increasing drastically the number of parameters needed to be trained which also depends on the length of the input time series. Thus, the transferability of this network is limited to the first two convolutions whose number of parameters depends solely on the number and length of the chosen filters.

We should note that t-LeNet is one of the approaches adopting a data augmentation technique to prevent overfitting especially for the relatively small time series datasets in the UCR/UEA archive. Their approach uses two data augmentation techniques: WS and WW. The former method is identical to MCNN's data augmentation technique originally proposed in Cui, Chen, and Chen, 2016. As for the second data augmentation technique, WW employs a warping technique that squeezes or dilates the time series. In order to deal with multi-length time series the WS method is adopted to ensure that subsequences of the same length are extracted for training the network. Therefore, a given input time series of length l is first dilated ($\times 2$) then squeezed ($\times \frac{1}{2}$) resulting in three time series of length l, $2l$ and $\frac{1}{2}l$ that are fed to WS to extract equal length subsequences for training. Note that in their original paper (Le Guennec, Malinowski, and Tavenard, 2016), WS' length is set to $0.9l$. Finally similarly to MCNN, since the WS method is also used at test time, a majority vote over the extracted subsequences' predicted labels is applied. Figure 1.10 shows the details of T-LeNet's architecture.

Multi Channel Deep Convolutional Neural Network

MCDCNN was originally proposed and validated on two multivariate time series datasets (Zheng et al., 2014; Zheng et al., 2016). The proposed architecture is mainly a traditional deep CNN with one modification for MTS data: the convolutions are applied independently (in parallel) on each dimension (or channel) of the input MTS.

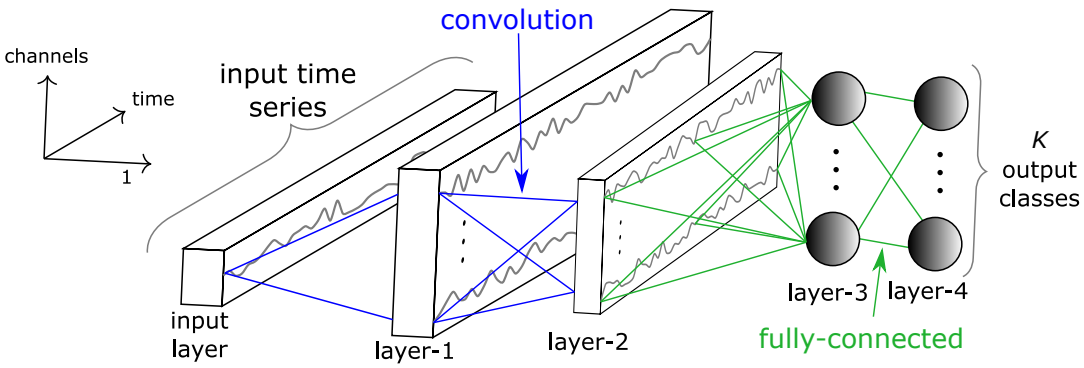


FIGURE 1.10: T-LeNet's architecture for time series classification.

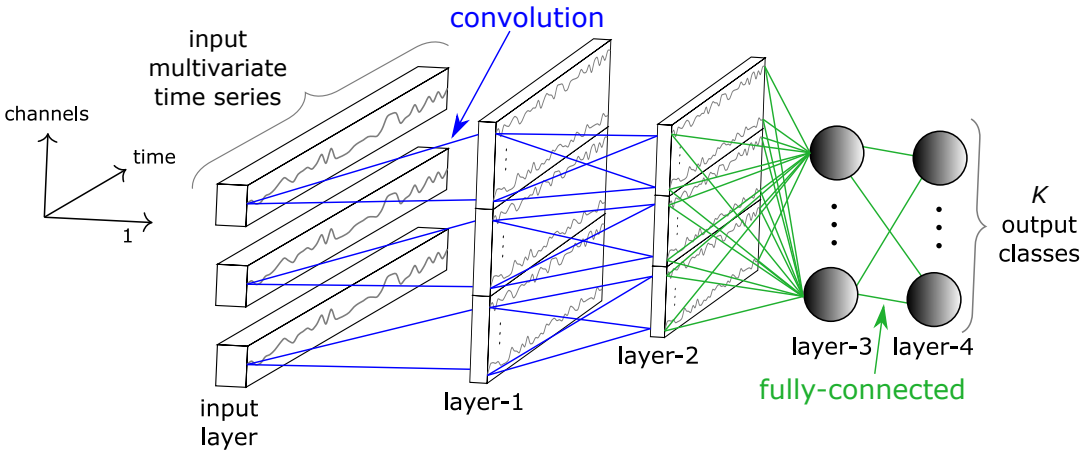


FIGURE 1.11: MCDCNN's architecture for time series classification.

Each dimension for an input MTS will go through two convolutional stages with 8 filters of length 5 with ReLU as the activation function. Each convolution is followed by a max pooling operation of length 2. The output of the second convolutional stage for all dimensions is concatenated over the channels axis and then fed to an FC layer with 732 neurons with ReLU as the activation function. Finally, the softmax classifier is used with a number of neurons equal to the number of classes in the dataset. By using an FC layer before the softmax classifier, the transferability of this network is limited to the first and second convolutional layers. MCDCNN's architecture is depicted in Figure 1.11.

Time Convolutional Neural Network

Time-CNN approach was originally proposed by Zhao et al., 2017 for both univariate and multivariate TSC. There are three main differences compared to the previously described networks. The first characteristic of Time-CNN is the use of the MSE instead of the traditional categorical cross-entropy loss function, which has been used by all the deep learning approaches we have mentioned so far. Hence, instead of a softmax classifier, the final layer is a traditional FC layer with sigmoid as the activation function, which does not guarantee a sum of probabilities equal to 1. Another difference to traditional CNNs is the use of a local *average* pooling operation instead of local *max* pooling. In addition, unlike MCDCNN, for MTS data they apply one convolution for all the dimensions of a multivariate classification task. Another unique characteristic of this architecture is that the final classifier is fully connected

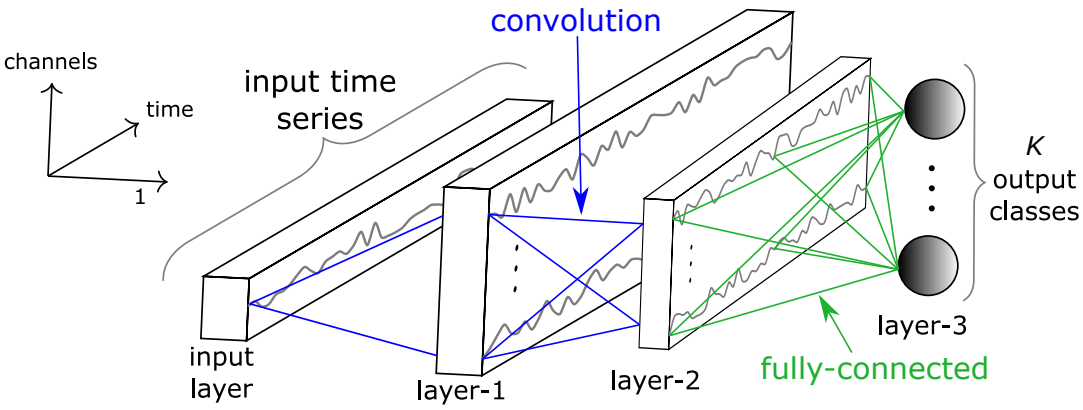


FIGURE 1.12: Time-CNN's architecture for time series classification.

directly to the output of the second convolution, which removes completely the GAP layer without replacing it with an FC non-linear layer.

The network is composed of two consecutive convolutional layers with respectively 6 and 12 filters followed by a local average pooling operation of length 3. The convolutions adopt the sigmoid as the activation function. The network's output consists of an FC layer with a number of neurons equal to the number of classes in the dataset. Figure 1.12 shows the relatively shallow architecture of Time-CNN.

Time Warping Invariant Echo State Network

TWIESN (Tanisaro and Heidemann, 2016) is the only non-convolutional *recurrent* architecture tested and re-implemented in our study. Although ESNs were originally proposed for time series forecasting, Tanisaro and Heidemann, 2016 proposed a variant of ESNs that uses directly the raw input time series and predicts a probability distribution over the class variables.

In fact, for each element (time stamp) in an input time series, the reservoir space is used to project this element into a higher dimensional space. Thus, for a univariate time series, the element is projected into a space whose dimensions are inferred from the size of the reservoir. Then for each element, a Ridge classifier (Hoerl and Kennard, 1970) is trained to predict the class of each time series element. During test time, for each element of an input test time series, the already trained Ridge classifier will output a probability distribution over the classes in a dataset. Then the a posteriori probability for each class is averaged over all time series elements, thus assigning for each input test time series the label for which the averaged probability is maximum. Following the original paper of Tanisaro and Heidemann, 2016, using a grid-search on an unseen split (20%) from the training set, we optimized TWIESN's three hyperparameters: the reservoir's size, sparsity and spectral radius. An example of TWIESN's architecture is depicted in Figure 1.5.

1.3.3 Hyperparameters

Tables 1.1 and 1.2 show respectively the architecture and the optimization hyperparameters for all the described approaches except for TWIESN, since its hyperparameters are not compatible with the eight other algorithms' hyperparameters. We should add that for all the other deep learning classifiers (with TWIESN omitted), a model checkpoint procedure was performed either on the training set or a validation set (split from the training set). Which means that if the model is trained for 1000

Methods	Architecture							
	#layers	#conv	#invar	normalize	pooling	feature	activate	regularize
MLP	4	0	0	none	none	FC	ReLU	dropout
FCN	5	3	4	batch	none	GAP	ReLU	none
ResNet	11	9	10	batch	none	GAP	ReLU	none
Encoder	5	3	4	instance	max	Att	PReLU	dropout
MCNN	4	2	2	none	max	FC	sigmoid	none
t-LeNet	4	2	2	none	max	FC	ReLU	none
MCDCNN	4	2	2	none	max	FC	ReLU	none
Time-CNN	3	2	2	none	avg	Conv	sigmoid	none

TABLE 1.1: Architecture's hyperparameters for the deep learning approaches.

Methods	Optimization						
	algorithm	valid	loss	epochs	batch	learning rate	decay
MLP	AdaDelta	train	entropy	5000	16	1.0	0.0
FCN	Adam	train	entropy	2000	16	0.001	0.0
ResNet	Adam	train	entropy	1500	16	0.001	0.0
Encoder	Adam	train	entropy	100	12	0.00001	0.0
MCNN	Adam	$\text{split}_{20\%}$	entropy	200	256	0.1	0.0
t-LeNet	Adam	train	entropy	1000	256	0.01	0.005
MCDCNN	SGD	$\text{split}_{33\%}$	entropy	120	16	0.01	0.0005
Time-CNN	Adam	train	mse	2000	16	0.001	0.0

TABLE 1.2: Optimization's hyperparameters for the deep learning approaches.

epochs, the best one on the validation set (or the train set) loss will be chosen for evaluation. This characteristic is included in Table 1.2 under the "valid" column. In addition to the model checkpoint procedure, we should note that all deep learning models in Table 1.1 were initialized randomly using Glorot's uniform initialization method (Glorot and Bengio, 2010). All models were optimized using a variant of SGD such as Adam (Kingma and Ba, 2015) and AdaDelta (Zeiler, 2012). We should add that for FCN, ResNet and MLP proposed in Wang, Yan, and Oates, 2017, the learning rate was reduced by a factor of 0.5 each time the model's training loss has not improved for 50 consecutive epochs (with a minimum value equal to 0.0001). One final note is that we have no way of controlling the fact that those described architectures might have been overfitted for the UCR/UEA archive and designed empirically to achieve a high performance, which is always a risk when comparing classifiers on a benchmark (Bagnall et al., 2017). We therefore think that challenges where only the training data is publicly available and the testing data are held by the challenge organizer for evaluation might help in mitigating this problem.

1.4 Experimental setup

We first start by presenting the datasets' properties we have adopted in this empirical study. We then describe in details our developed open-source framework of deep learning for time series classification.

1.4.1 Datasets

Univariate archive

In order to have a thorough and fair experimental evaluation of all approaches, we tested each algorithm on the whole UCR/UEA archive (Dau et al., 2019) which contains 85 univariate time series datasets. The datasets possess different varying characteristics such as the length of the series which has a minimum value of 24 for the ItalyPowerDemand dataset and a maximum equal to 2,709 for the HandOutLines dataset. One important characteristic that could impact the DNNs' accuracy is the size of the training set which varies between 16 and 8926 for respectively DiatomSizeReduction and ElectricDevices datasets. We should note that twenty datasets contains a relatively small training set (50 or fewer instances) which surprisingly was not an impediment for obtaining high accuracy when applying a very deep architecture such as ResNet. Furthermore, the number of classes varies between 2 (for 31 datasets) and 60 (for the ShapesAll dataset). Note that the time series in this archive are already z-normalized (Bagnall et al., 2017).

Other than the fact of being publicly available, the choice of validating on the UCR/UEA archive is motivated by having datasets from different domains which have been broken down into seven different categories (Image Outline, Sensor Readings, Motion Capture, Spectrographs, ECG, Electric Devices and Simulated Data) in Bagnall et al., 2017. Further statistics, which we do not repeat for brevity, were conducted on the UCR/UEA archive in Bagnall et al., 2017.

Multivariate archive

We also evaluated all deep learning models on Baydogan's archive (Baydogan, 2015) that contains 13 MTS classification datasets. For memory usage limitations over a single GPU, we left the MTS dataset Performance Measurement System out of our experimentations. This archive also exhibits datasets with different characteristics such as the length of the time series which, unlike the UCR/UEA archive, varies among the same dataset. This is due to the fact that the datasets in the UCR/UEA archive are already re-scaled to have an equal length among one dataset (Bagnall et al., 2017).

In order to solve the problem of unequal length time series in the MTS archive we decided to linearly interpolate the time series of each dimension for every given MTS, thus each time series will have a length equal to the longest time series' length. This form of pre-processing has also been used by Ratanamahatana and Keogh, 2005 to show that the length of a time series is not an issue for TSC problems. This step is very important for deep learning models whose architecture depends on the length of the input time series (such as an MLP) and for parallel computation over the GPUs. We did not z-normalize any time series, but we emphasize that this traditional pre-processing step (Bagnall et al., 2017) should be further studied for univariate as well as multivariate data, especially since normalization is known to have a huge effect on DNNs' learning capabilities (Zhang et al., 2017). Note that this process is only true for the MTS datasets whereas for the univariate benchmark, the time series are already z-normalized. Since the data is pre-processed using the same technique for all nine classifiers, we can safely say, to some extent, that the accuracy improvement of certain models can be solely attributed to the model itself. Table 1.3 shows the different characteristics of each MTS dataset used in our experiments.

Dataset	old length	new length	classes	dimensions	train	test
ArabicDigits	4-93	93	10	13	6600	2200
AUSLAN	45-136	136	95	22	1140	1425
CharacterTrajectories	109-205	205	20	3	300	2558
CMUsubject16	127-580	580	2	62	29	29
ECG	39-152	152	2	2	100	100
JapaneseVowels	7-29	29	9	12	270	370
KickVsPunch	274-841	841	2	62	16	10
Libras	45-45	45	15	2	180	180
Outflow	50-997	997	2	4	803	534
UWave	315-315	315	8	3	200	4278
Wafer	104-198	198	2	6	298	896
WalkVsRun	128-1918	1919	2	62	28	16

TABLE 1.3: The multivariate time series classification archive.

1.4.2 Experiments

For each dataset in both archives (97 datasets in total), we have trained the nine deep learning models (presented in the previous section) with 10 different runs each. Each run uses the same original train/test split in the archive but with a different random weight initialization, which enables us to take the mean accuracy over the 10 runs in order to reduce the bias due to the weights' initial values. In total, we have performed 8730 experiments for the 85 univariate and 12 multivariate TSC datasets. Thus, given the huge number of models that needed to be trained, we ran our experiments on a cluster of 60 GPUs. These GPUs were a mix of four types of Nvidia graphic cards: GTX 1080 Ti, Tesla K20, K40 and K80. The total sequential running time was approximately 100 days, that is if the computation has been done on a single GPU. However, by leveraging the cluster of 60 GPUs, we managed to obtain the results in less than one month. We implemented our framework using the open source deep learning library Keras (Chollet, 2015) with the Tensorflow (Abadi et al., 2015) back-end[1].

Following Lucas et al., 2018; Forestier et al., 2017b; Petitjean et al., 2016; Grabocka et al., 2014 we used the mean accuracy measure averaged over the 10 runs on the test set. When comparing with the state-of-the-art results published in Bagnall et al., 2017 we averaged the accuracy using the median test error. Following the recommendation in Demšar, 2006 we used the Friedman test (Friedman, 1940) to reject the null hypothesis. Then we performed the pairwise post-hoc analysis recommended by Benavoli, Corani, and Mangili, 2016 where the average rank comparison is replaced by a Wilcoxon signed-rank test (Wilcoxon, 1945) with Holm's alpha (5%) correction (Holm, 1979; Garcia and Herrera, 2008). To visualize this type of comparison we used a critical difference diagram proposed by Demšar, 2006, where a thick horizontal line shows a group of classifiers (a clique) that are not-significantly different in terms of accuracy.

1.5 Results

In this section, we present the accuracies for each one of the nine approaches. All accuracies are absolute and not relative to each other that is if we claim algorithm A is 5% better than algorithm B, this means that the average accuracy is 0.05 higher for algorithm A than B.

[1]The implementations are available on https://github.com/hfawaz/dl-4-tsc

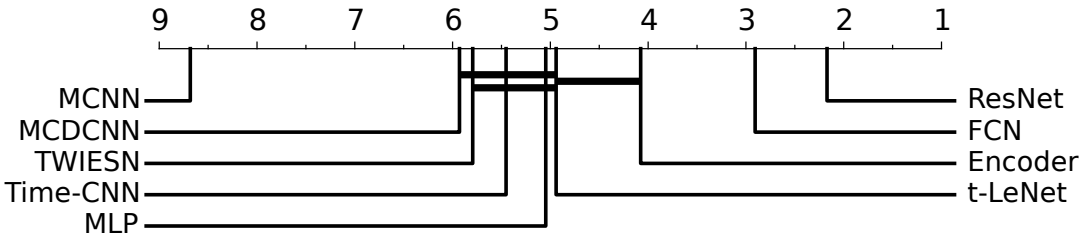


FIGURE 1.13: Critical difference diagram showing pairwise statistical difference comparison of nine deep learning classifiers on the univariate UCR/UEA time series classification archive.

1.5.1 Results for univariate time series

We provide on the companion GitHub repository the raw accuracies over the 10 runs for the nine deep learning models we have tested on the 85 univariate time series datasets: the UCR/UEA archive (Dau et al., 2019). The corresponding critical difference diagram is shown in Figure 1.13. The ResNet significantly outperforms the other approaches with an average rank of almost 2. ResNet wins on 34 problems out of 85 and significantly outperforms the FCN architecture. This is in contrast to the original paper's results where FCN was found to outperform ResNet on 18 out of 44 datasets, which shows the importance of validating on a larger archive in order to have a robust statistical significance.

We believe that the success of ResNet is highly due to its deep flexible architecture. First of all, our findings are in agreement with the deep learning for computer vision literature where deeper neural networks are much more successful than shallower architectures (He et al., 2016). In fact, in a space of 4 years, neural networks went from 7 layers in AlexNet 2012 (Krizhevsky, Sutskever, and Hinton, 2012) to *1000* layers for ResNet 2016 (He et al., 2016). These types of deep architectures generally need a huge amount of data in order to generalize well on unseen examples (He et al., 2016). Although the datasets used in our experiments are relatively small compared to the billions of labeled images (such as ImageNet (Russakovsky et al., 2015) and OpenImages (Krasin et al., 2017) challenges), the deepest networks did reach competitive accuracies on the UCR/UEA archive benchmark.

We give two potential reasons for this high generalization capabilities of deep CNNs on the TSC tasks. First, having seen the success of convolutions in classification tasks that require learning features that are spatially invariant in a *two* dimensional space (such as width and height in images), it is only natural to think that discovering patterns in a *one* dimensional space (time) should be an easier task for CNNs thus requiring less data to learn from. The other more direct reason behind the high accuracies of deep CNNs on time series data is its success in other sequential data such as speech recognition (Hinton et al., 2012) and sentence classification (Kim, 2014) where text and audio, similarly to time series data, exhibit a natural temporal ordering.

MCNN yielded very low accuracy compared to t-LeNet. The main common idea between both of these approaches is extracting subsequences to augment the training data. Therefore the model learns to classify a time series from a shorter subsequence instead of the whole one, then with a majority voting scheme the time series at test time are assigned a class label. The poor performance of MCNN compared to t-LeNet suggest that the architecture itself is not very well optimized for the underlying task. In addition to MCNN's WS technique, t-LeNet uses the WW data augmentation method. This suggest that t-LeNet benefited even more from the

warping time series data augmentation that shows significant improvement compared to MCNN's simple window slicing technique. Nevertheless, more architecture independent experiments are needed to further verify these findings regarding data augmentation (Ismail Fawaz et al., 2018b).

Although MCDCNN and Time-CNN were originally proposed to classify MTS datasets, we have evaluated them on the univariate UCR/UEA archive. The MCDCNN did not manage to beat any of the cl assifiers except for the ECG5000 dataset which is already a dataset where almost all approaches reached the highest accuracy. This low performance is probably due to the non-linear FC layer that replaces the GAP pooling of the best performing algorithms (FCN and ResNet). This FC layer reduces the effect of learning time invariant features which explains why MLP, Time-CNN and MCDCNN exhibit very similar performance.

One approach that shows relatively high accuracy is Encoder (Serrà, Pascual, and Karatzoglou, 2018). The statistical test indicates a significant difference between Encoder, FCN and ResNet. FCN wins on 17 datasets whereases Encoder wins only on 9 which suggests the superiority of the GAP layer compared to Encoder's attention mechanism.

1.5.2 Comparing with state-of-the-art approaches

In this subsection, we compared ResNet (the most accurate DNN of our study) with the current state-of-the-art classifiers evaluated on the UCR/UEA archive in the great time series classification bake off (Bagnall et al., 2017). Note that our empirical study strongly suggests to use ResNet instead of any other deep learning algorithm - it is the most accurate one with similar runtime to FCN (the second most accurate DNN). Finally, since ResNet's results were averaged over ten different random initializations, we chose to take one iteration of ResNet (the median) and compare it to other state-of-the-art algorithms that were executed once over the original train/test split provided by the UCR/UEA archive.

Out of the 18 classifiers evaluated by Bagnall et al., 2017, we have chosen the four best performing algorithms: (1) EE proposed by Lines and Bagnall, 2015 is an ensemble of nearest neighbor classifiers with 11 different time series similarity measures; (2) BOSS published in Schäfer, 2015 forms a discriminative bag of words by discretizing the time series using a Discrete Fourier Transform and then building a nearest neighbor classifier with a bespoke distance measure; (3) ST developed by Hills et al., 2014 extracts discriminative subsequences (shapelets) and builds a new representation of the time series that is fed to an ensemble of 8 classifiers; (4) COTE proposed by Bagnall et al., 2017 is basically a weighted ensemble of 35 TSC algorithms including EE and ST. We also include HIVE-COTE (proposed by Lines, Taylor, and Bagnall, 2018) which improves significantly COTE's performance by leveraging a hierarchical voting system as well as adding two new classifiers and two additional transformation domains. In addition to these five state-of-the-art classifiers, we have included the classic nearest neighbor coupled with DTW and a warping window set through cross-validation on the training set (denoted by NN-DTW-WW), since it is still one of the most popular methods for classifying time series data (Bagnall et al., 2017). Finally, we added a recent approach named PF which is similar to Random Forest but replaces the attribute based splitting criteria by a random similarity measure chosen out of EE's elastic distances (Lucas et al., 2018). Note that we did not implement any of the non-deep TSC algorithms. We used the results provided by Bagnall et al., 2017 and the other corresponding papers to construct the critical difference diagram in Figure 1.14.

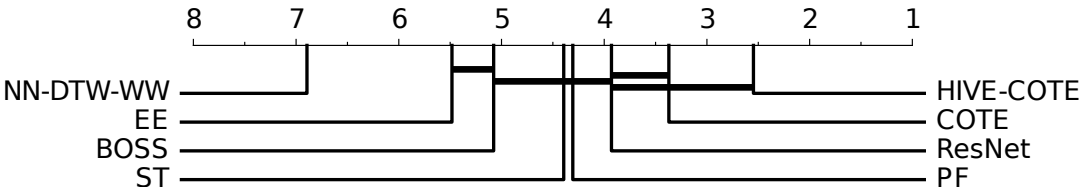


FIGURE 1.14: Critical difference diagram showing pairwise statistical difference comparison of state-of-the-art classifiers on the univariate UCR/UEA time series classification archive.

Figure 1.14 shows the critical difference diagram over the UEA benchmark with ResNet added to the pool of six classifiers. As we have previously mentioned, the state-of-the-art classifiers are compared to ResNet's median accuracy over the test set. Nevertheless, we generated the ten different average ranks for each iteration of ResNet and observed that the ranking of the compared classifiers is stable for the ten different random initializations of ResNet. The statistical test failed to find any significant difference between COTE/HIVE-COTE and ResNet which is the only TSC algorithm that was able to reach similar performance to COTE. Note that for the ten different random initializations of ResNet, the pairwise statistical test always failed to find any significance between ResNet and COTE/HIVE-COTE. PF, ST, BOSS and ResNet showed similar performances according to the Wilcoxon signed-rank test, but the fact that ResNet is not significantly different than COTE suggests that more datasets would give a better insight into these performances (Demšar, 2006). NN-DTW-WW and EE showed the lowest average rank suggesting that these methods are no longer competitive with current state-of-the-art algorithms for TSC. It is worthwhile noting that cliques formed by the Wilcoxon Signed Rank Test with Holm's alpha correction do not necessary reflect the rank order (Lines, Taylor, and Bagnall, 2018). For example, if we have three classifiers (C_1, C_2, C_3) with average ranks $(C_1 > C_2 > C_3)$, one can still encounter a case where C_1 is not significantly worse than C_2 and C_3 with C_2 and C_3 being significantly different. In our experiments, when comparing to state-of-the-art algorithms, we have encountered this problem with (ResNet>COTE>HIVE-COTE). Therefore we should emphasize that HIVE-COTE and COTE are *significantly* different when performing the pairwise statistical test.

Although HIVE-COTE is still the most accurate classifier (when evaluated on the UCR/UEA archive) its use in a real data mining application is limited due to its huge training time complexity which is $O(N^2 \cdot T^4)$ corresponding to the training time of one of its individual classifiers ST. However, we should note that the recent work of Bostrom and Bagnall, 2015 showed that it is possible to use a random sampling approach to decrease significantly the running time of ST (HIVE-COTE's choke-point) without any loss of accuracy. On the other hand, DNNs offer this type of scalability evidenced by its revolution in the field of computer vision when applied to images, which are thousand times larger than time series data (Russakovsky et al., 2015). In addition to the huge training time, HIVE-COTE's *classification* time is bounded by a linear scan of the training set due to employing a nearest neighbor classifier, whereas the trivial GPU parallelization of DNNs provides instant classification. Finally we should note that unlike HIVE-COTE, ResNet's hyperparameters were not tuned for each dataset but rather the same architecture was used for the whole benchmark suggesting further investigation of these hyperparameters should improve DNNs' accuracy for TSC. These results should give an insight of deep learning for TSC therefore encouraging researchers to consider DNNs as robust real time

classifiers for time series data.

The need of a fair comparison

In this paragraph, we highlight the fairness of the comparison to other machine learning TSC algorithms. Since we did not train nor test any of the state-of-the-art non deep learning algorithms, it is possible that we allowed much more training time for the described DNNs. For example, for a lazy machine learning algorithm such as NN-DTW, training time is zero when allowing maximum warping whereas it has been shown that judicially setting the warping window Dau et al., 2017 can lead to a significant increase in accuracy. Therefore, we believe that allowing a much more thorough search of DTW's warping window would lead to a fairer comparison between deep learning approaches and other state-of-the-art TSC algorithms. In addition to cross-validating NN-DTW's hyper-parameters, we can imagine spending more time on data pre-processing and cleansing (e.g. smoothing the input time series) in order to improve the accuracy of NN-DTW (Höppner, 2016; Dau et al., 2019). Ultimately, in order to obtain a fair comparison between deep learning and current state-of-the-art algorithms for TSC, we think that the time spent on optimizing a network's weights should be also spent on optimizing non deep learning based classifiers especially lazy learning algorithms such as the K nearest neighbor coupled with any similarity measure.

1.5.3 Results for multivariate time series

We provide on our companion repository[2] the detailed performance of the nine deep learning classifiers for 10 different random initializations over the 12 MTS classification datasets (Baydogan, 2015). Although Time-CNN and MCDCNN are the only architectures originally proposed for MTS data, they were outperformed by the three deep CNNs (ResNet, FCN and Encoder), which shows the superiority of these approaches on the MTS classification task. The corresponding critical difference diagram is depicted in Figure 1.15, where the statistical test failed to find any significant difference between the nine classifiers which is mainly due to the small number of datasets compared to their univariate counterpart. Therefore, we illustrated in Figure 1.16 the critical difference diagram when both archives are combined (evaluation on 97 datasets in total). At first glance, we can notice that when adding the MTS datasets to the evaluation, the critical difference diagram in Figure 1.16 is not significantly different than the one in Figure 1.13 (where only the univariate UCR/UEA archive was taken into consideration). This is probably due to the fact that the algorithms' performance over the 12 MTS datasets is negligible to a certain degree when compared to the performance over the 85 univariate datasets. These observations reinforces the need to have an equally large MTS classification archive in order to evaluate hybrid univariate/multivariate time series classifiers. The rest of the analysis is dedicated to studying the effect of the datasets' characteristics on the algorithms' performance.

1.5.4 What can the dataset's characteristics tell us about the best architecture?

The first dataset characteristic we have investigated is the problem's domain. Table 1.4 shows the algorithms' performance with respect to the dataset's theme.

[2] www.github.com/hfawaz/dl-4-tsc

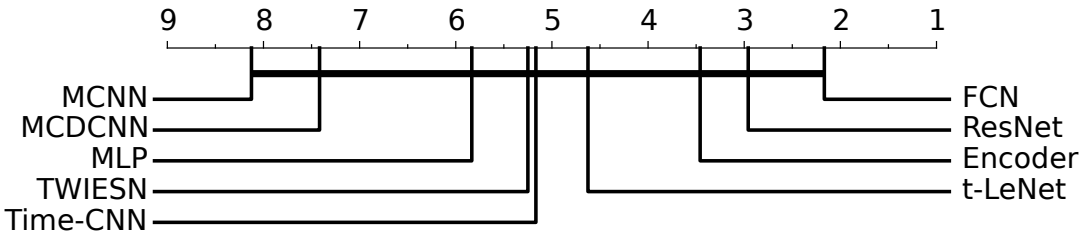


FIGURE 1.15: Critical difference diagram showing pairwise statistical difference comparison of nine deep learning classifiers on the multivariate time series classification archive.

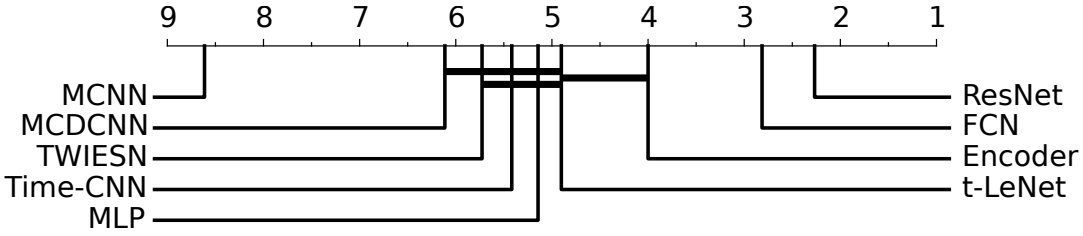


FIGURE 1.16: Critical difference diagram showing pairwise statistical difference comparison of nine deep learning classifiers on both univariate and multivariate time series classification archives.

These themes were first defined in Bagnall et al., 2017. Again, we can clearly see the dominance of ResNet as the best performing approach across different domains. One exception is the ECG datasets (7 in total) where ResNet was drastically beaten by the FCN model in 71.4% of ECG datasets. However, given the small sample size (only 7 datasets), we cannot conclude that FCN will almost always outperform the ResNet model for ECG datasets (Bagnall et al., 2017).

The second characteristic which we have studied is the time series length. Similar to the findings for non deep learning models in Bagnall et al., 2017, the time series length does not give information on deep learning approaches' performance. Table 1.5 shows the average rank of each DNN over the univariate datasets grouped by the datasets' lengths. One might expect that the relatively short filters (3) might affect the performance of ResNet and FCN since longer patterns cannot be captured by short filters. However, since increasing the number of convolutional layers will increase the path length viewed by the CNN model (Vaswani et al., 2017), ResNet and FCN managed to outperform other approaches whose filter length is longer (21) such as Encoder. For the recurrent TWIESN algorithm, we were expecting a poor

Themes (#)	MLP	FCN	ResNet	Encoder	MCNN	t-LeNet	MCDCNN	Time-CNN	TWIESN
DEVICE (6)	0.0	50.0	**83.3**	0.0	0.0	16.7	0.0	0.0	0.0
ECG (7)	14.3	**71.4**	28.6	42.9	0.0	0.0	14.3	0.0	0.0
IMAGE (29)	6.9	34.5	**48.3**	10.3	0.0	10.3	3.4	10.3	0.0
MOTION (14)	0.0	21.4	**57.1**	0.0	0.0	42.9	0.0	0.0	0.0
SENSOR (16)	6.2	31.2	**68.8**	31.2	6.2	37.5	6.2	0.0	12.5
SIMULATED (6)	0.0	33.3	**83.3**	33.3	0.0	33.3	0.0	0.0	0.0
SPECTRO (7)	14.3	14.3	**71.4**	0.0	0.0	0.0	0.0	28.6	28.6

TABLE 1.4: Deep learning algorithms' performance grouped by themes. Each entry is the percentage of dataset themes an algorithm is most accurate for. Bold indicates the best model.

Length	MLP	FCN	ResNet	Encoder	MCNN	t-LeNet	MCDCNN	Time-CNN	TWIESN
<81	5.57	3.5	**2.64**	2.93	8.93	6.14	3.29	3.86	5.86
81-250	4.53	**1.74**	1.84	3.74	8.74	5.05	5.74	4.89	5.95
251-450	4.32	3.05	**1.86**	3.68	8.82	4.73	6.55	5.14	5.41
451-700	5.23	2.92	**2.0**	4.31	7.77	4.62	6.31	5.38	4.77
701-1000	5.4	2.1	**1.7**	4.6	8.4	2.9	6.1	6.8	5.5
>1000	3.71	3.0	**1.57**	4.0	8.29	3.29	5.71	6.43	7.0

TABLE 1.5: Deep learning algorithms' average ranks grouped by the datasets' length. Bold indicates the best model.

Train size	MLP	FCN	ResNet	Encoder	MCNN	t-LeNet	MCDCNN	Time-CNN	TWIESN
<100	4.64	2.21	**1.93**	4.57	8.5	5.0	6.71	4.39	5.32
100-399	5.21	2.5	**1.86**	3.64	8.29	4.71	5.93	6.25	4.71
400-799	4.73	3.07	**2.33**	3.67	8.93	6.0	3.67	4.33	6.0
>799	4.29	3.64	**1.86**	2.71	8.86	2.57	5.21	5.71	7.79

TABLE 1.6: Deep learning algorithms' average ranks grouped by the training sizes. Bold indicates the best model.

accuracy for very long time series since a recurrent model may "forget" a useful information present in the early elements of a long time series. However, TWIESN did reach competitive accuracies on several long time series datasets such as reaching a 96.8% accuracy on Meat whose time series length is equal to 448. This would suggest that ESNs can solve the vanishing gradient problem especially when learning from long time series.

A third important characteristic is the training size of datasets and how it affects a DNN's performance. Table 1.6 shows the average rank for each classifier grouped by the train set's size. Again, ResNet and FCN still dominate with not much of a difference. However we found one very interesting dataset: DiatomSizeReduction. ResNet and FCN achieved the worst accuracy (30%) on this dataset while Time-CNN reached the best accuracy (95%). Interestingly, DiatomSizeReduction is the smallest datasets in the UCR/UEA archive (with 16 training instances), which suggests that ResNet and FCN are easily overfitting this dataset. This suggestion is also supported by the fact that Time-CNN is the smallest model: it contains a very small number of parameters by design with only 18 filters compared to the 512 filters of FCN. This simple architecture of Time-CNN renders overfitting the dataset much harder. Therefore, we conclude that the small number of filters in Time-CNN is the main reason behind its success on small datasets, however this shallow architecture is unable to capture the variability in larger time series datasets which is modeled efficiently by the FCN and ResNet architectures. One final observation that is in agreement with the deep learning literature is that in order to achieve high accuracies while training a DNN, a large training set is needed. Figure 1.17 shows the effect of the training size on ResNet's accuracy for the TwoPatterns dataset: the accuracy increases significantly when adding more training instances until it reaches 100% for 75% of the training data.

Finally, we should note that the number of classes in a dataset - although it yielded some variability in the results for the recent TSC experimental study conducted by Bagnall et al., 2017 - did not show any significance when comparing the classifiers based on this characteristic. In fact, most DNNs architectures, with the categorical cross-entropy as their cost function, employ mainly the same classifier:

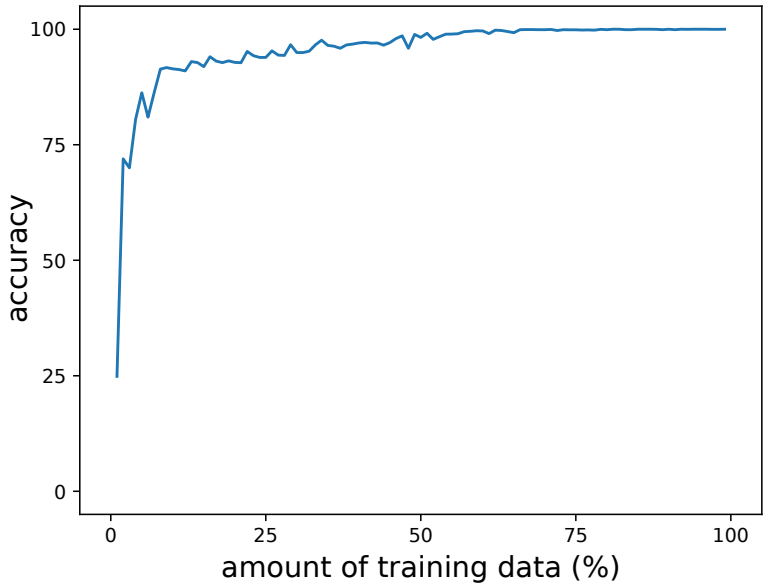


FIGURE 1.17: ResNet's accuracy variation with respect to the amount of training instances in the TwoPatterns dataset.

softmax which is basically designed for multi-class classification.

Overall, our results show that, on average, ResNet is the best architecture with FCN and Encoder following as second and third respectively. ResNet performed very well in general except for the ECG datasets where it was outperformed by FCN. MCNN was significantly worse than all the other approaches, with t-LeNet showing a superiority with the additional WW technique. We found small variance between the approaches that replace the GAP layer with an FC dense layer (MCDCNN, CNN) which also showed similar performance to TWIESN, MLP and t-LeNet.

1.5.5 Effect of random initializations

The initialization of deep neural networks has received a significant amount of interest from many researchers in the field (LeCun, Bengio, and Hinton, 2015). These advancement have contributed to a better understanding and initialization of deep learning models in order to maximize the quality of non-optimal solutions found by the gradient descent algorithm (Glorot and Bengio, 2010). Nevertheless, we observed in our experiments, that DNNs for TSC suffer from a significant decrease (increase) in accuracy when initialized with bad (good) random weights. Therefore, we study in this section, how random initializations can affect the performance of ResNet and FCN on the whole benchmark in a best and worst case scenario.

Figure 1.18 shows the accuracy plot of ResNet versus FCN on the 85 univariate time series datasets when aggregated over the 10 random initializations using three different functions: the minimum, median and maximum. When first observing Figure 1.18 one can easily conclude that ResNet has a better performance than FCN across most of the datasets regardless of the aggregation method. This is in agreement with the critical difference diagram as well as the analysis conducted in the previous subsections, where ResNet was shown to achieve higher performance on most datasets with different characteristics. A deeper look into the *minimum* aggregation (red points in Figure 1.18) shows that FCN's performance is less stable compared to ResNet's. In other words, the weight's initial value can easily decrease the accuracy of FCN whereas ResNet maintained a relatively high accuracy when

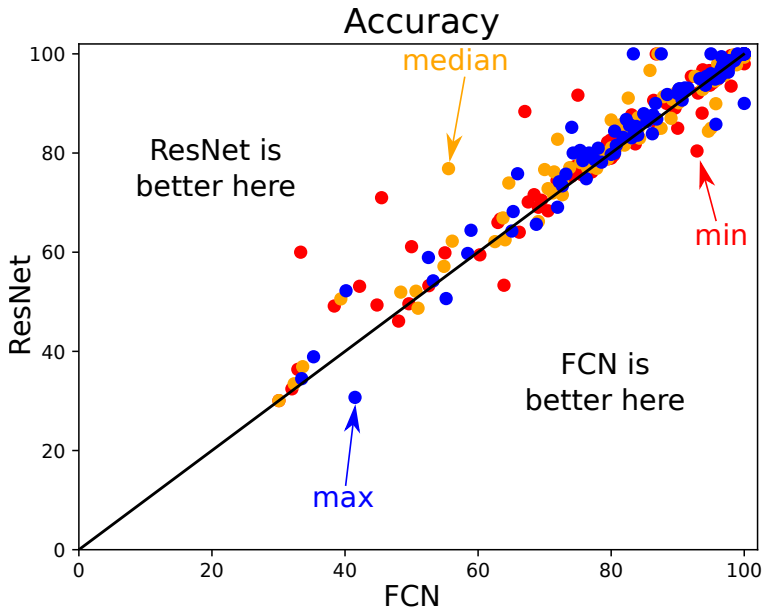


FIGURE 1.18: Accuracy of ResNet versus FCN over the UCR/UEA archive when three different aggregations are taken: the minimum, median and maximum.

taking the worst initial weight values. This is also in agreement with the average standard deviation of ResNet (1.48) which is less than FCN's (1.70). These observations would encourage a practitioner to avoid using a complex deep learning model since its accuracy may be unstable. Nevertheless we think that investigating different weight initialization techniques such as leveraging the weights of a pre-trained neural network would yield better and much more stable results (Ismail Fawaz et al., 2018d) or even simply leveraging this high variance by ensembling DNNs (Ismail Fawaz et al., 2019e).

1.6 Visualization

In this section, we start by investigating the use of Class Activation Map to provide an interpretable feedback that highlights the reason for a certain decision taken by the classifier. We then propose another visualization technique which is based on Multi-Dimensional Scaling (Kruskal and Wish, 1978) to understand the latent representation that is learned by the DNNs.

1.6.1 Class Activation Map

We investigate the use of CAM which was first introduced by Zhou et al., 2016 to highlight the parts of an image that contributed the most for a given class identification. Wang, Yan, and Oates, 2017 later introduced a one-dimensional CAM with an application to TSC. This method explains the classification of a certain deep learning model by highlighting the subsequences that contributed the most to a certain classification. Figure 1.19 and 1.20 show the results of applying CAM respectively on GunPoint and Meat datasets. Note that employing the CAM is only possible for the approaches with a GAP layer preceding the softmax classifier (Zhou et al., 2016). Therefore, we only considered in this section the ResNet and FCN models, who also achieved the best accuracies overall. Note that Wang, Yan, and Oates, 2017 was the

only paper to propose an interpretable analysis of TSC with a DNN. We should emphasize that this is a very important research area which is usually neglected for the sake of improving accuracy: only 2 out of the 9 approaches provided a method that explains the decision taken by a deep learning model. In this section, we start by presenting the CAM method from a mathematical point of view and follow it with two interesting case studies on Meat and GunPoint datasets.

By employing a Global Average Pooling layer, ResNet and FCN benefit from the CAM method (Zhou et al., 2016), which makes it possible to identify which regions of an input time series constitute the reason for a certain classification. Formally, let $A(t)$ be the result of the last convolutional layer which is an MTS with M variables. $A_m(t)$ is the univariate time series for the variable $m \in [1, M]$, which is in fact the result of applying the m^{th} filter. Now let w_m^c be the weight between the m^{th} filter and the output neuron of class c. Since a GAP layer is used then the input to the neuron of class c (z_c) can be computed by the following equation:

$$z_c = \sum_m w_m^c \sum_t A_m(t) \tag{1.10}$$

The second sum constitutes the averaged time series over the whole time dimension but with the denominator omitted for simplicity. The input z_c can be also written by the following equation:

$$z_c = \sum_t \sum_m w_m^c A_m(t) \tag{1.11}$$

Finally the Class Activation Map (CAM_c) that explains the classification as label c is given in the following equation:

$$CAM_c(t) = \sum_m w_m^c A_m(t) \tag{1.12}$$

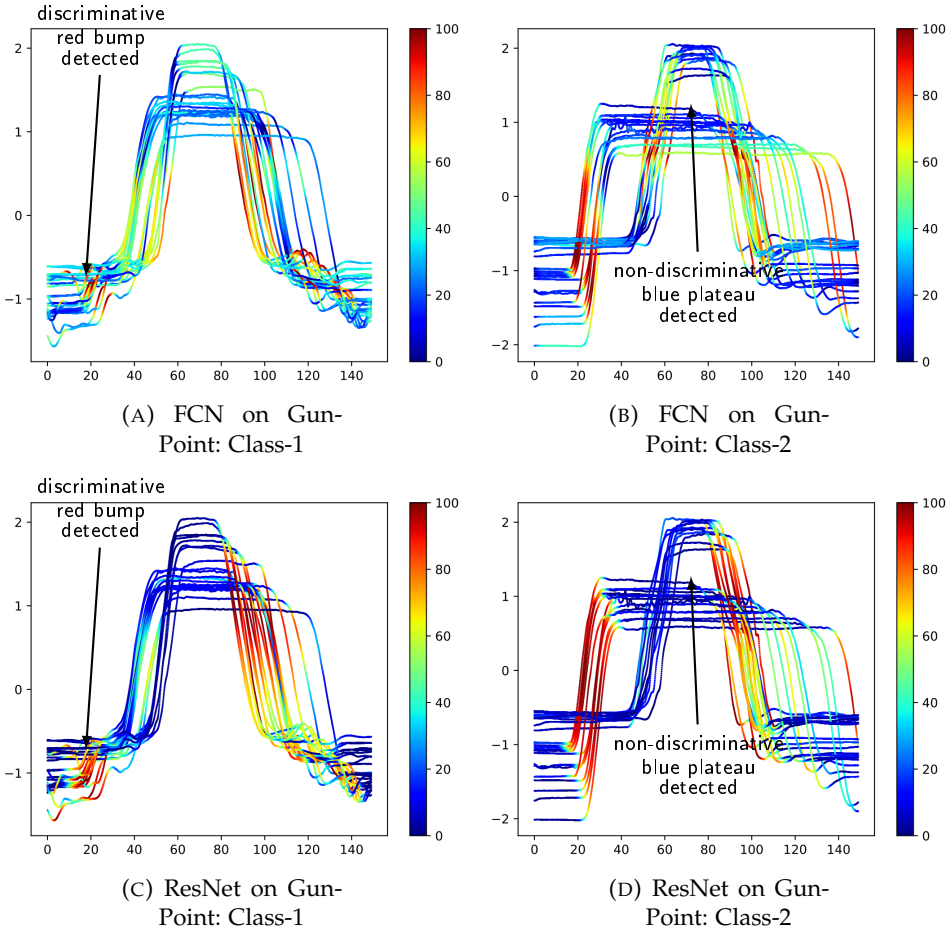
CAM is actually a univariate time series where each element (at time stamp $t \in [1, T]$) is equal to the weighted sum of the M data points at t, with the weights being learned by the neural network.

GunPoint dataset

The GunPoint dataset was first introduced by Ratanamahatana and Keogh, 2005 as a TSC problem. This dataset involves one male and one female actor performing two actions (Gun-Draw and Point) which makes it a binary classification problem. For Gun-Draw (Class-1 in Figure 1.19), the actors have first their hands by their sides, then draw a replicate gun from hip-mounted holster, point it towards the target for one second, then finally place the gun in the holster and their hands to their initial position. Similarly to Gun-Draw, for Point (Class-2 in Figure 1.19) the actors follow the same steps but instead of pointing a gun they point their index finger. For each task, the centroid of the actor's right hands on both X and Y axes were tracked and seemed to be very correlated, therefore the dataset contains only one univariate time series: the X-axis.

We chose to start by visualizing the CAM for GunPoint for three main reasons. First, it is easy to visualize unlike other noisy datasets. Second, both FCN and ResNet models achieved almost 100% accuracy on this dataset which will help us to verify if both models are reaching the same decision for the same reasons. Finally, it contains only two classes which allow us to analyze the data much more easily.

Figure 1.19 shows the CAM's result when applied on each time series from both classes in the training set while classifying using the FCN model (Figure 1.19a



(A) FCN on GunPoint: Class-1

(B) FCN on GunPoint: Class-2

(C) ResNet on GunPoint: Class-1

(D) ResNet on GunPoint: Class-2

FIGURE 1.19: Highlighting with the Class Activation Map the contribution of each time series region for both classes in GunPoint when using the FCN and ResNet classifiers. Red corresponds to high contribution and blue to almost no contribution to the correct class identification (smoothed for visual clarity and best viewed in color).

and 1.19b) and the ResNet model (Figure 1.19c and 1.19d). At first glance, we can clearly see how both DNNs are neglecting the plateau non-discriminative regions of the time series when taking the classification decision. It is depicted by the blue flat parts of the time series which indicates no contribution to the classifier's decision. As for the highly discriminative regions (the red and yellow regions) both models were able to select the same parts of the time series which correspond to the points with high derivatives. Actually, the first most distinctive part of class-1 discovered by both classifiers is almost the same: the little red bump in the bottom left of Figure 1.19a and 1.19c. Finally, another interesting observation is the ability of CNNs to localize a given discriminative shape regardless where it appears in the time series, which is evidence for CNNs' capability of learning time-invariant warped features.

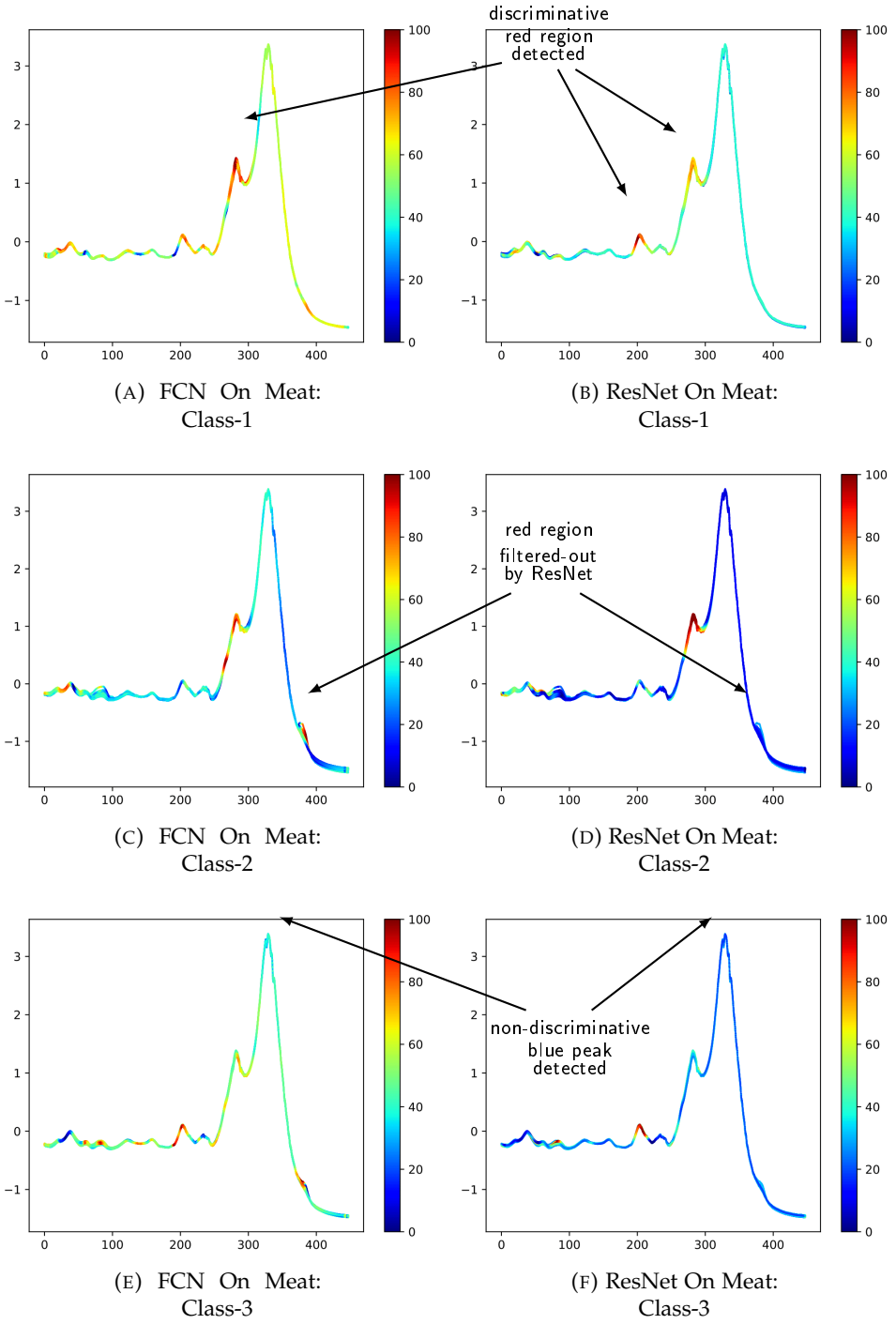
An interesting observation would be to compare the discriminative regions identified by a deep learning model with the most discriminative shapelets extracted by other shapelet-based approaches. This observation would also be backed up by the mathematical proof provided by Cui, Chen, and Chen, 2016, that showed how the learned filters in a CNN can be considered a generic form of shapelets extracted by the learning shapelets algorithm (Grabocka et al., 2014). Ye and Keogh, 2011 identified that the most important shapelet for the Gun/NoGun classification occurs when the actor's arm is lowered (about 120 on the horizontal axis in Figure 1.19). Hills et al., 2014 introduced a shapelet transformation based approach that discovered shapelets that are similar to the ones identified by Ye and Keogh, 2011. For ResNet and FCN, the part where the actor lowers his/her arm (bottom right of Figure 1.19) seems to be also identified as potential discriminative regions for some time series. On the other hand, the part where the actor raises his/her arm seems to be also a discriminative part of the data which suggests that the deep learning algorithms are identifying more "shapelets". We should note that this observation cannot confirm which classifier extracted the most discriminative subsequences especially because all algorithms achieved similar accuracy on GunPoint dataset. Perhaps a bigger dataset might provide a deeper insight into the interpretability of these machine learning models. Finally, we stress that the shapelet transformation classifier (Hills et al., 2014) is an ensemble approach, which makes unclear how the shapelets affect the decision taken by the individual classifiers whereas for an end-to-end deep learning model we can directly explain the classification by using the Class Activation Map.

Meat dataset

Although the previous case study on GunPoint yielded interesting results in terms of showing that both models are localizing meaningful features, it failed to show the difference between the two most accurate deep learning classifiers: ResNet and FCN. Therefore we decided to further analyze the CAM's result for the two models on the Meat dataset.

Meat is a food spectrograph dataset which are usually used in chemometrics to classify food types, a task that has obvious applications in food safety and quality assurance. There are three classes in this dataset: Chicken, Pork and Turkey corresponding respectively to classes 1, 2 and 3 in Figure 1.20. Al-Jowder, Kemsley, and Wilson, 1997 described how the data is acquired from 60 independent samples using Fourier transform infrared spectroscopy with attenuated total reflectance sampling.

Similarly to GunPoint, this dataset is easy to visualize and does not contain very noisy time series. In addition, with only three classes, the visualization is possible to understand and analyze. Finally, unlike for the GunPoint dataset, the two



(A) FCN On Meat: Class-1

(B) ResNet On Meat: Class-1

(C) FCN On Meat: Class-2

(D) ResNet On Meat: Class-2

(E) FCN On Meat: Class-3

(F) ResNet On Meat: Class-3

FIGURE 1.20: Highlighting with the Class Activation Map the contribution of each time series region for the three classes in Meat when using the FCN and ResNet classifiers. Red corresponds to high contribution and blue to almost no contribution to the correct class identification (smoothed for visual clarity and best viewed in color).

approaches ResNet and FCN reached significantly different results on Meat with respectively 97% and 83% accuracy.

Figure 1.20 enables the comparison between FCN's CAM (left) and ResNet's CAM (right). We first observe that ResNet is much more firm when it comes to highlighting the regions. In other words, FCN's CAM contains much more smoother regions with cyan, green and yellow regions, whereas ResNet's CAM contains more dark red and blue subsequences showing that ResNet can filter out non-discriminative and discriminative regions with a higher confidence than FCN, which probably explains why FCN is less accurate than ResNet on this dataset. Another interesting observation is related to the red subsequence highlighted by FCN's CAM for class 2 and 3 at the bottom right of Figure 1.20c and 1.20e. By visually investigating this part of the time series, we clearly see that it is a non-discriminative part since the time series of both classes exhibit this bump. This subsequence is therefore filtered-out by the ResNet model which can be seen by the blue color in the bottom right of Figure 1.20d and 1.20f. These results suggest that ResNet's superiority over FCN is mainly due to the former's ability to filter-out non-distinctive regions of the time series. We attribute this ability to the main characteristic of ResNet which is composed of the residual connections between the convolutional blocks that enable the model to *learn to skip* unnecessary convolutions by dint of its shortcut links (He et al., 2016).

1.6.2 Multi-Dimensional Scaling

We propose the use of MDS (Kruskal and Wish, 1978) with the objective to gain some insights on the spatial distribution of the input time series belonging to different classes in the dataset. MDS uses a pairwise distance matrix as input and aims at placing each object in a N-dimensional space such as the between-object distances are preserved as well as possible. Using the ED on a set of input time series belonging to the test set, it is then possible to create a similarity matrix and apply MDS to display the set into a two dimensional space. This straightforward approach supposes that the ED is able to strongly separate the raw time series, which is usually not the case evident by the low accuracy of the nearest neighbor when coupled with ED (Bagnall et al., 2017).

On the other hand, we propose to apply this MDS method to visualize the set of time series with its latent representation learned by the network. Usually in a deep neural network, we have several hidden layers and one can find several latent representation of the dataset. But since we are aiming at visualizing the class specific latent space, we chose to use the last latent representation of a DNN (the one directly before the softmax classifier), which is known to be a class specific layer (Yosinski et al., 2014b). We decided to apply this method only on ResNet and FCN for two reasons: (1) when evaluated on the UCR/UEA archive they reached the highest ranks; (2) they both employ a GAP layer before the softmax layer making the number of latent features invariant to the time series length.

To better explain this process, for each input time series, the last convolution (for ResNet and FCN) outputs a multivariate time series whose dimensions are equal to the number of filters (128) in the last convolution, then the GAP layer averages the latter 128-dimensional multivariate time series over the time dimension resulting in a vector of 128 real values over which the ED is computed. As we worked with the ED, we used metric MDS (Kruskal and Wish, 1978) that minimizes a cost function

called *Stress* which is a residual sum of squares:

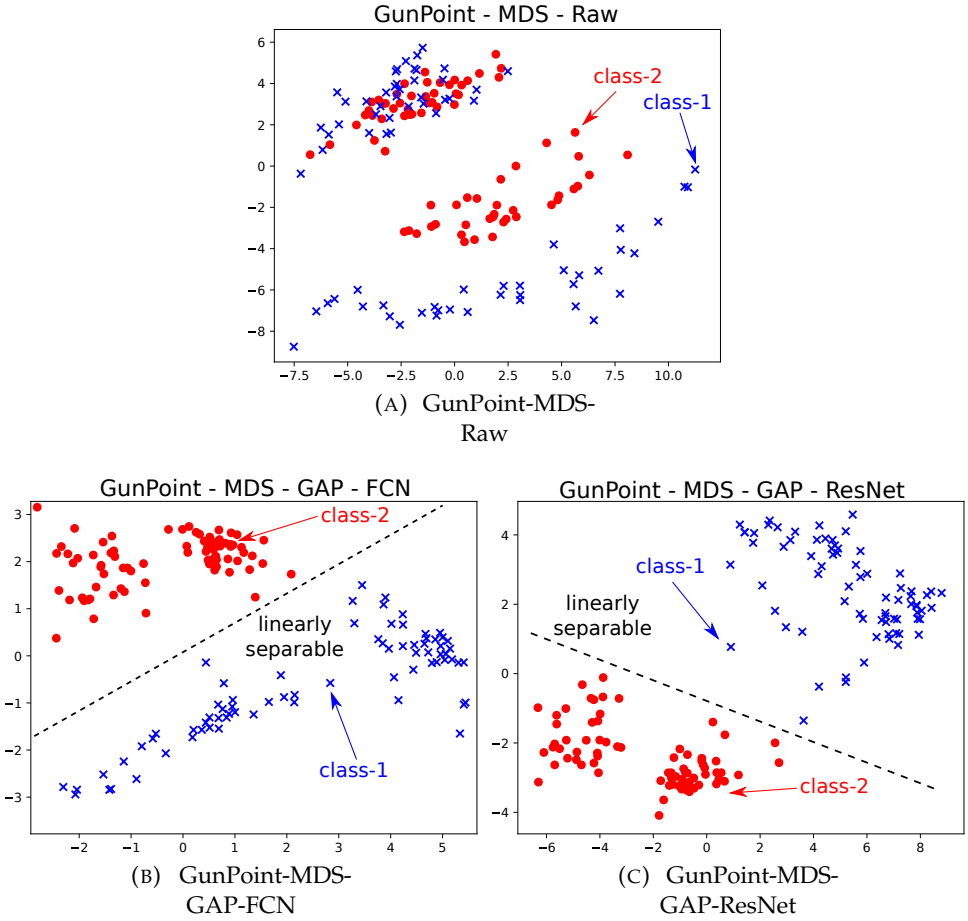
$$Stress_D(X_1, \dots, X_N) = \left(\frac{\sum_{i,j} \left(d_{ij} - \|x_i - x_j\| \right)^2}{\sum_{i,j} d_{ij}^2} \right)^{1/2} \tag{1.13}$$

where d_{ij} is the ED between the GAP vectors of time series X_i and X_j. Obviously, one has to be careful about the interpretation of MDS output, as the data space is highly simplified (each time series X_i is represented as a single data point x_i).

We should note that there exists many visualization algorithms that would provide some insights about the data. One of the most popular techniques is called t-SNE (Maaten and Hinton, 2008). This method projects high-dimensional data by giving each data point (or in our case a time series or its representation) a location in a two or three dimensional map. It is an optimized derivative of Stochastic Neighbor Embedding (Hinton and Roweis, 2003). Although t-SNE has been used by many researchers, we decided to go with the MDS algorithm when analyzing time series representation. The reason behind this choice is that MDS is based on the pairwise distance matrix between the time series which is a very common way to cluster and analyze time series data (Petitjean et al., 2016), whereas t-SNE uses an iterative stochastic optimization process in order to preserve local structure at the expense of global structure in which we are interested.

Figure 1.21 shows three MDS plots for the GunPoint dataset using: (1) the raw input time series (Figure 1.21a); (2) the learned latent features from the GAP layer for FCN (Figure 1.21b); and (3) the learned latent features from the GAP layer for ResNet (Figure 1.21c). We can easily observe in Figure 1.21a that when using the raw input data and projecting it into a 2D space, the two classes are not linearly separable. On the other hand, in both Figures 1.21b and 1.21c, by applying MDS on the latent representation learned by the network, one can easily separate the set of time series belonging to the two classes. We note that both deep learning models (FCN and ResNet) managed to project the data from GunPoint into a linearly separable space which explains why both models performed equally very well on this dataset with almost 100% accuracy.

Although the visualization of MDS on GunPoint yielded some interesting results, it failed to pinpoint the difference between the two deep learning models FCN and ResNet. Therefore we decided to analyze another dataset where the accuracy of both models differed by almost 15%. Figure 1.22 shows three MDS plots for the Wine dataset using: (1) the raw input time series (Figure 1.22a); (2) the learned latent features from the GAP layer for FCN (Figure 1.22b); and (3) the learned latent features from the GAP layer for ResNet (Figure 1.22c). At first glimpse of Figure 1.22, the reader can conclude that all projections, even when using the learned representation, are not linearly separable which is evident by the relatively low accuracy of both models FCN and ResNet which is equal respectively to 58.7% and 74.4%. A thorough observation shows us that the learned hidden representation of ResNet (Figure 1.22c) separates the data from both classes in a much clearer way than the FCN (Figure 1.22b). In other words, FCN's learned representation has too many data points close to the decision boundary whereas ResNet's hidden features enables projecting data points further away from the decision boundary. This observation could explain why ResNet achieves a better performance than FCN on the Wine dataset.

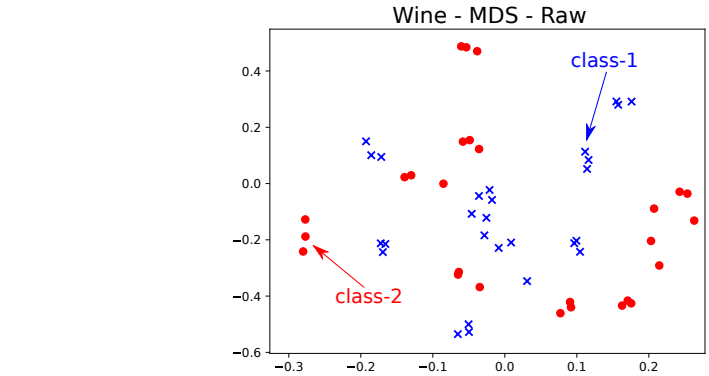


(A) GunPoint-MDS-Raw

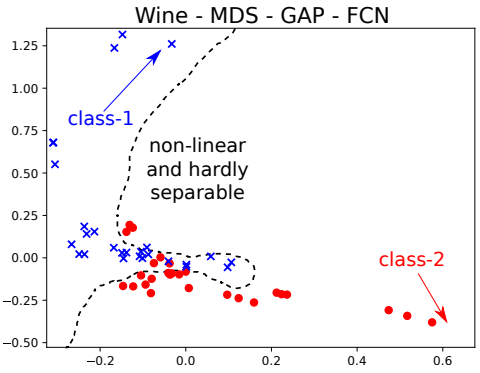
(B) GunPoint-MDS-GAP-FCN

(C) GunPoint-MDS-GAP-ResNet

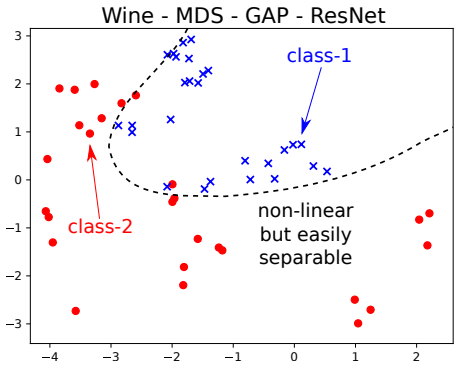
FIGURE 1.21: Multi-Dimensional Scaling (MDS) applied on GunPoint for: (top) the raw input time series; (bottom) the learned features from the Global Average Pooling (GAP) layer for FCN (left) and ResNet (right) - (best viewed in color). This figure shows how the ResNet and FCN are projecting the time series from a non-linearly separable 2D space (when using the raw input), into a linearly separable 2D space (when using the latent representation).



(A) Wine-MDS-Raw



(B) Wine-MDS-GAP-FCN



(C) Wine-MDS-GAP-ResNet

FIGURE 1.22: Multi-Dimensional Scaling (MDS) applied on Wine for: (top) the raw input time series; (bottom) the learned features from the Global Average Pooling (GAP) layer for FCN (left) and ResNet (right) - (best viewed in color). This figure shows how ResNet, unlike FCN, is able to project the data into an easily separable space when using the learned features from the GAP layer (Color figure online).

1.7 Conclusion

In this chapter, we presented the largest empirical study of DNNs for TSC. We described the most recent successful deep learning approaches for TSC in many different domains such as human activity recognition and sleep stage identification. Under a unified taxonomy, we explained how DNNs are separated into two main categories of generative and discriminative models. We re-implemented nine recently published end-to-end deep learning classifiers in a unique framework which we make publicly available to the community. Our results show that end-to-end deep learning can achieve the current state-of-the-art performance for TSC with architectures such as Fully Convolutional Neural Networks and deep Residual Networks. Finally, we showed how the black-box effect of deep models which renders them uninterpretable, can be mitigated with a Class Activation Map visualization that highlights which parts of the input time series, contributed the most to a certain class identification.

Although we have conducted an extensive experimental evaluation, deep learning for time series classification, unlike for computer vision and NLP tasks, still lacks a thorough study of data augmentation (Ismail Fawaz et al., 2018b; Forestier et al., 2017b) and transfer learning (Ismail Fawaz et al., 2018d; Serrà, Pascual, and Karatzoglou, 2018). In addition, the time series community would benefit from an extension of this empirical study that compares in addition to accuracy, the training and testing time of these deep learning models. Furthermore, we think that the effect of z-normalization (and other normalization methods) on the learning capabilities of DNNs should also be thoroughly explored. In our future work, we aim to investigate and answer the aforementioned limitations by conducting more extensive experiments especially on multivariate time series datasets. In order to achieve all of these goals, one important challenge for the TSC community is to provide one large generic *labeled* dataset similar to the large images database in computer vision such as ImageNet (Russakovsky et al., 2015) that contains 1000 classes.

In conclusion, with data mining repositories becoming more frequent, leveraging deeper architectures that can learn automatically from annotated data in an end-to-end fashion, makes deep learning a very enticing approach. In this chapter, we demonstrated the potential of deep neural networks for TSC, nevertheless these complex machine learning models can still benefit from many regularization techniques, which is the main focus of the following chapter.

Chapter 2

Regularizing deep neural networks

2.1 Introduction

Deep learning models usually have significantly more trainable parameters than the number of training instances. Nevertheless, in the previous chapter, we showed how these artificial neural networks are able to achieve good generalization capabilities compared to traditional TSC algorithms. Yet most of these models require some sort of regularization in order to ensure a small generalization error (i.e. a small difference between the training and testing error). In this chapter, we present four main techniques for regularizing DNNs for TSC: (1) transfer learning (Ismail Fawaz et al., 2018d); (2) ensembling (Ismail Fawaz et al., 2019e); (3) data augmentation (Ismail Fawaz et al., 2018b); and adversarial training (Ismail Fawaz et al., 2019b); Our work reveals that these techniques improve the accuracy of the previously described deep learning models such as FCN and ResNet.

The chapter is divided into four main sections, each one describing in details the regularization technique. First we start by describing how transfer learning can further improve the performance of DNNs for the TSC task. We then showcase our findings regarding how ensembling deep learning models can significantly improve the performance of these time series classifiers. In the following section, we present how adopting a data augmentation method can help in improving the performance of neural networks for TSC. Finally, we define and investigate how DNNs are vulnerable to adversarial examples, and explore the possibility of leveraging those examples to regularize deep learning time series classifiers.

2.2 Transfer learning

CNNs have recently been shown to significantly outperform the nearest neighbor approach coupled with the DTW algorithm (NN-DTW) on the UCR/UEA archive benchmark (Dau et al., 2019) for the TSC problem (Wang, Yan, and Oates, 2017). CNNs were not only able to beat the NN-DTW baseline, but in the previous chapter, we showed how they were also able to reach results that are not significantly different than COTE (Bagnall et al., 2015) - which is an ensemble of 35 classifiers. However, despite the high performance of these CNNs, deep learning models are still prone to overfitting. One example where these neural networks fail to generalize is when the training set of the time series dataset is very small. We attribute this huge difference in accuracy to the overfitting phenomena, which is still an open area of research in the deep learning community (Zhang et al., 2017). This problem is known to be mitigated using several regularization techniques such as transfer learning (Yosinski et al., 2014a), where a model trained on a source task is then fine-tuned on a target dataset. For example in Figure 2.1, we trained a model on the

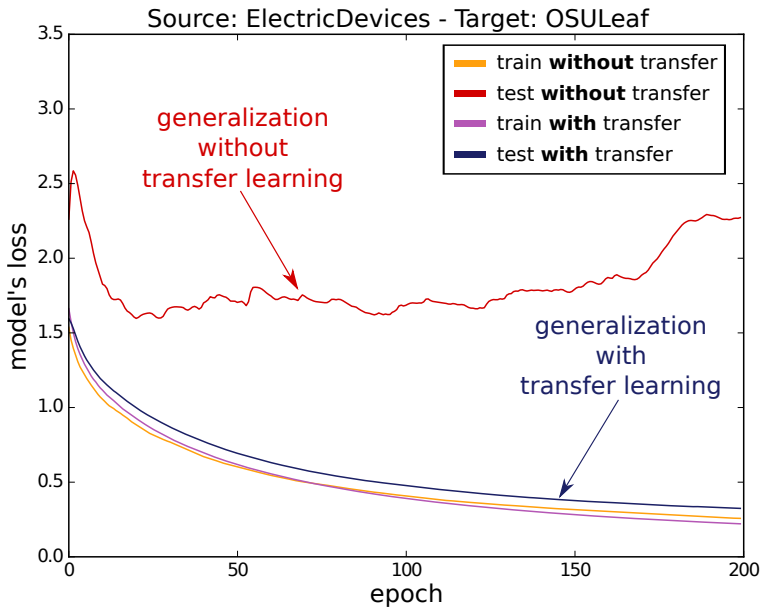


FIGURE 2.1: Evolution of model's loss (train and test) with and without the transfer learning method using ElectricDevices as source and OSULeaf as target datasets. (Best viewed in color).

ElectricDevices dataset (Dau et al., 2019) and then fine-tuned this same model on the OSULeaf dataset (Dau et al., 2019), which significantly improved the network's generalization capability.

Transfer learning is currently used in almost every deep learning model when the target dataset does not contain enough labeled data (Yosinski et al., 2014a). Despite its recent success in computer vision (Csurka, 2017), transfer learning has been rarely applied to deep learning models for time series data. One of the reasons for this absence is probably the lack of one big general purpose dataset similar to ImageNet (Russakovsky et al., 2015) or OpenImages (Krasin et al., 2017) but for time series. Furthermore, it is only recently that deep learning was proven to work well for TSC (Cui, Chen, and Chen, 2016) and there is still much to be explored in building deep neural networks for mining time series data (Cristian Borges Gamboa, 2017).

Since transferring deep learning models, between the UCR/UEA archive datasets (Dau et al., 2019), have not been thoroughly studied, we decided to investigate this area of research with the ultimate goal to determine in advance which dataset transfers could benefit the CNNs and improve their TSC accuracy.

The intuition behind the transfer learning approach for time series data is also partially inspired by the observation of Cui, Chen, and Chen, 2016, where the authors showed that shapelets (Ye and Keogh, 2009) (or subsequences) learned by the learning shapelets approach (Grabocka et al., 2014) are related to the filters (or kernels) learned by the CNNs. We hypothesize that these learned subsequences might not be specific to one dataset and could occur in other unseen datasets with un/related classification tasks. Another observation for why transfer learning should work for time series data is its recent success in computer vision tasks (Csurka, 2017). Indeed, since time series data contain one temporal dimension (time) compared to two dimensions for images (width and height), it is only natural to think that if filters can successfully be transferred on images (Yosinski et al., 2014a), they should also be transferable across time series datasets.

To evaluate the potential of transfer learning for TSC, we performed experiments

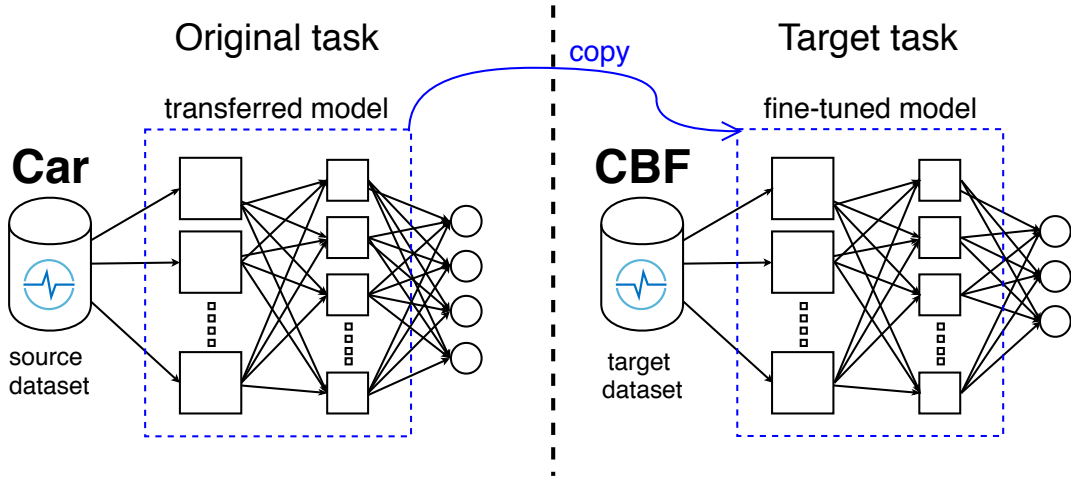


FIGURE 2.2: General deep learning training process with transfer learning for time series classification. In this example, a model is first pre-trained on Car (source dataset) and then the corresponding weights are fine-tuned on CBF (target dataset).

where each pair of datasets in the UCR/UEA archive was tested twice: we pre-trained a model for each dataset, then transferred and fine-tuned it on all the other datasets (a total of more than 7140 trained models). Figure 2.2 illustrates the architecture of our proposed framework of transfer learning for TSC on two datasets. The obtained results show that time series do exhibit some low level features that could be used in a transfer learning approach. They also show that using transfer learning reduces the training time by reducing the number of epochs needed for the network to converge on the train set.

Motivated by the consensus that transferring models between similar datasets improves the classifier's accuracy (Weiss, Khoshgoftaar, and Wang, 2016), we used the DTW algorithm as an inter-datasets similarity measure in order to quantify the relationship between the source and target datasets in our transfer learning framework. Our experiments show that DTW can be used to predict the best source dataset for a given target dataset. Our method can thus identify which datasets should be considered for transfer learning given a new TSC problem.

2.2.1 Background and related work

Before getting into the details of the recent applications for transfer learning, we give a formal definition of the latter (Weiss, Khoshgoftaar, and Wang, 2016).

Definition 1. Transfer learning for deep neural networks, is the process of first training a base network on a source dataset and task, and then transfer the learned features (the network's weights) to a second network to be trained on a target dataset and task. Throughout this section, we will refer to *source* dataset as the dataset we are transferring the pre-trained model *from*, and to *target* dataset as the dataset we are transferring the pre-trained model *to*.

Now that we have established the necessary definition, we will dive into the recent applications of transfer learning for time series data mining tasks. In fact, transfer learning is sometimes confused with the domain adaptation approach (Pan and Yang, 2010; Long et al., 2015). The main difference with the latter method is

that the model is jointly trained on the source and target datasets (Weiss, Khoshgoftaar, and Wang, 2016). The goal of using the target instances during training, is to minimize the discrepancy between the source's and target's instances. In Arief-Ang, Salim, and Hamilton, 2017, a domain adaptation approach was proposed to predict human indoor occupancy based on the carbon dioxide concentration in the room. In Kasteren, Englebienne, and Kröse, 2008, hidden Markov models' generative capabilities were used in a domain adaptation approach to recognize human activities based on a sensor network.

For time series anomaly detection, a transfer learning approach was used to determine which time series should be transferred from the source to the target dataset to be used with an NN-DTW classifier (Vercruyssen, Meert, and Davis, 2017). Similarly, Spiegel, 2016 developed a method to transfer specific training examples from the source dataset to the target dataset and hence compute the dissimilarity matrix using the new training set. As for time series forecasting, a transfer learning approach for an auto-encoder was employed to predict the wind-speed in a farm (Hu, Zhang, and Zhou, 2016). The authors proposed first to train a model on the historical wind-speed data of an old farm and fine-tune it using the data of a new farm. In Banerjee et al., 2017 restricted Boltzmann machines were first pre-trained for acoustic phoneme recognition and then fine-tuned for post-traumatic stress disorder diagnosis.

Perhaps the recent work in Serrà, Pascual, and Karatzoglou, 2018 is the closest to ours in terms of using transfer learning to improve the accuracy of deep neural networks for TSC. In this work, the authors designed a CNN with an attention mechanism to encode the time series in a supervised manner. Before fine-tuning a model on a target dataset, the model is first jointly pre-trained on several source datasets with themes (Bagnall et al., 2017) that are different from the target dataset's theme which limits the choice of the source dataset to only one. Additionally, unlike Serrà, Pascual, and Karatzoglou, 2018, we take a pre-designed deep learning model without modifying it nor adding regularizers. This enabled us to solely attribute the improvement in accuracy to the transfer learning feature, which we describe in details in the following section.

2.2.2 Method

In this subsection, we present our proposed method of transfer learning for TSC. The adopted neural network architecture was FCN, for the simple reason of being three times faster than ResNet while being ranked the second most accurate model in the previous chapter. We then thoroughly explain how we adapted the network for the transfer learning process. Finally, we present our DTW based method that enabled us to compute the inter-datasets similarities, which we later use to guide the transfer learning process.

Network adaptation

After training FCN on the 85 datasets in the archive, we obtain 85 different neural networks. The only difference between these 85 neural network architectures lies in the output layer. The rest of the layers have the same number of parameters but with different values. In fact the last layer, which is a softmax classifier, depends on the number of classes in the dataset.

Thus, given a source dataset D_s and a target dataset D_t, we first train the network on D_s. We then remove the last layer and replace it with another softmax layer whose

number of neurons is equal to the number of classes in the target dataset D_t. The added softmax layer's parameters are initialized randomly using Glorot's uniform initialization (Glorot and Bengio, 2010). This new network is then re-trained (fine-tuned) on D_t.

We chose to fine-tune the whole network instead of training only the last newly added output layer. We tried to limit back-propagating the gradient to the last layer, but found that the network failed to converge. This is in compliance with the transfer learning literature (Yosinski et al., 2014a), where re-training the whole network almost always leads to better results.

Finally, we should add that one of the advantages of using a global average pooling layer is that we do not need to re-scale the input time series when transferring models between time series of different length.

Inter-datasets similarity

One of the main challenges with transfer learning is choosing the source dataset. In Pan et al., 2011, it was demonstrated that a learning algorithm trained with a certain source domain will not yield an optimal performance if the marginal distributions of the datasets' input are different. In our case, the total number of datasets in the UCR/UEA archive is 85. Therefore for each target dataset in the archive, we have 84 potential source datasets. This makes the trial and error based approach for transfer learning very costly in terms of computational resources. Hence, we propose to use the DTW distance to compute the similarities between the datasets, thus guiding the choice of a source dataset for a given target dataset.

Note that it is practically impossible to directly estimate the performance of a model learned on a source dataset by applying it on a target dataset's train set since the last layer of the network is *specific* (Yosinski et al., 2014a) to the classes of the source dataset.

Before describing in details our method for computing inter-datasets similarity, we start by explaining the DTW distance.

DTW was first proposed for speech recognition when aligning two audio signals (Sakoe and Chiba, 1978). Suppose we want to compute the dissimilarity between two time series, $X_1 = [x_{1,1}, x_{1,2}, \ldots, x_{1,m}]$ and $X_2 = [x_{2,1}, x_{2,2}, \ldots, x_{2,n}]$. The length of X_1 and X_2 are denoted respectively by m and n.

Let $M(X_1, X_2)$ be the $m \times n$ point-wise dissimilarity matrix between X_1 and X_2, where $M_{i,j} = ||x_{1,i} - x_{2,j}||^2$. A warping path $P = ((c_1, d_1), (c_2, d_2), \ldots, (c_s, d_s))$ is a series of points that define a crossing of M. The warping path must satisfy three conditions: (1) $(c_1, d_1) = (1, 1)$; (2) $(c_s, d_s) = (m, n)$; (3) $0 \leq c_{i+1} - c_i \leq 1$ and $0 \leq d_{j+1} - d_j \leq 1$ for all $i < m$ and $j < n$. The DTW measure between two series corresponds to the path through M that minimizes the total distance. In fact, the distance for any path P is equal to $D_P(A, B) = \sum_{i=1}^{s} P_i$. Hence if $\mathbf{P}$ is the space of all possible paths, the optimal one - whose cost is equal to $DTW(A, B)$ - is denoted by P^* and can be computed using: $\min_{P \in \mathbf{P}} D_P(A, B)$. The optimal warping path can be obtained efficiently by applying a dynamic programming technique to fill the cost matrix M.

Now that we have explained in details the DTW algorithm, which is usually used for computing a distance between two time series, we will describe the DBA algorithm which was first proposed by Petitjean, Ketterlin, and Gançarski, 2011 in order to average in the DTW induced space (as opposed to the arithmetic mean in the Euclidean space).

DBA is an averaging method which consists in iteratively refining an initial series in order to minimize its squared DTW distance to the set of series we want to average. Technically for each refinement (or iteration), DBA consists of two main steps:

1. Computing the DTW between each individual series and the current temporary average that we want to refine. This is in order to find optimal associations between the elements of the average series and elements of the set of series to be averaged.

2. Now that we have these associations, we update each element of the average series as the barycenter of the elements associated to it during the previous step.

In order to compute the similarities between the datasets, we first reduce the number of time series for each dataset to one time series (or prototype) per class. The per class prototype is computed by averaging the set of time series in the corresponding class, using the previously described DBA algorithm as a data reduction step. The latter summarizing function was proposed and validated as an averaging method in the DTW induced space. In addition, DBA has been recently used as a data reduction technique where it was evaluated in a nearest centroid classification schema (Petitjean et al., 2014). Therefore, to generate the similarity matrix between the UCR datasets, we computed a distance between each pair of datasets. Finally, for simplicity and since the main goal of this project is not the inter-datasets similarity, we chose the distance between two datasets to be equal to the minimum distance between the prototypes of their corresponding classes. Note that other averaging methods such as soft-DTW (Cutur and Blondel, 2017) and TEKA (Marteau, 2019) could be used instead of DBA in our framework, but we leave such exploration for our future work.

Algorithm 1 shows the different steps followed to compute the distance matrix between the UCR datasets. The first part of the algorithm (lines 1 through 7) presents the data reduction technique similar to Petitjean et al., 2014. For the latter step, we first go through the classes of each dataset (lines 1, 2 and 3) and then average the set of time series for each class. Following the recommendations in Petitjean et al., 2014, the averaging method (DBA) was initialized to be equal to the medoid of the time series selected set (line 4). We fixed the number of iterations for the DBA algorithm to be equal to 10, for which the averaging method has been shown to converge (Petitjean, Ketterlin, and Gançarski, 2011).

After having reduced the different sets for each time series dataset, we proceed to the actual distance computation step (lines 8 through 22). From line 8 to 10, we loop through every possible combination of datasets pairs. Lines 13 and 14 show the loop through each class for each dataset (at this stage each class is represented by one average time series thanks to the data reduction steps). Finally, lines 15 through 19 set the distance between two datasets to be equal to the minimum DTW distance between their corresponding classes.

One final note is that when computing the similarity between the datasets, the only time series data we used came from the training set, thus eliminating any bias due to having seen the test set's distribution.

Algorithm 1 Inter-datasets similarity

Input: N time series datasets in an array D
Output: $N \times N$ datasets similarity matrix

Initialization : matrix M of size $N \times N$
data reduction step
for $i = 1$ to N **do**
    $C = D[i].classes$
    for $c = 1$ to $length(C)$ **do**
        $avg\_init = medoid(C[c])$
        $C[c] = DBA(C[c], avg\_init)$
    end for
end for
distance calculation step
for $i = 1$ to N **do**
    $C_i = D[i].classes$
    for $j = 1$ to N **do**
        $C_j = D[j].classes$
        $dist = \infty$
        for $c_i = 1$ to $length(C_i)$ **do**
            for $c_j = 1$ to $length(C_j)$ **do**
                $cdist = DTW(C_i[c_i], C_j[c_j])$
                $dist = minimum(dist, cdist)$
            end for
        end for
        $M[i, j] = dist$
    end for
end forreturn M

2.2.3 Experimental setup

Datasets

We evaluate our developed framework thoroughly on the largest publicly available benchmark for time series analysis: the UCR/UEA archive (Dau et al., 2019), which consists of 85 datasets selected from various real-world domains. The time series in the archive are already z-normalized to have a mean equal to zero and a standard deviation equal to one. During the experiments, we used the default training and testing set splits provided by UCR. For pre-training a model, we used only the train set of the source dataset. We also fine-tuned the pre-trained model solely on the target dataset's training data. Hence the test sets were only used for evaluation purposes.

2.2.4 Experiments

For each pair of datasets (D_1 and D_2) in the UCR/UEA archive we need to perform two experiments:

- D_1 is the source dataset and D_2 is the target dataset.
- D_1 is the target dataset and D_2 is the source dataset.

Which makes it in total 7140 experiments for the 85 dataset in the archive. Hence, given the huge number of models that need to be trained, we ran our experiments on a cluster of 60 GPUs. These GPUs were a mix of three types of Nvidia graphic cards: GTX 1080 Ti, Tesla K20, K40 and K80. The total sequential running time was approximately 168 days, that is if the computation has been done on a single GPU. But by leveraging the cluster of 60 GPUs, we managed to obtain the results in less than one week. We implemented our framework using the open source deep learning library Keras (Chollet, 2015) with the Tensorflow (Abadi et al., 2016) back-end. For reproducibility purposes, we provide the 7140 trained Keras models (in a HDF5 format) on the companion web page of this project[1]. We have also published the raw results and the full source code of our method to enable the time series community to verify and build upon our findings[2].

2.2.5 Results

The experiments described in the previous section yielded interesting yet hard-to-understand results. In this section, we first present the result of the 85×84 experiments in a form of a matrix (displayed as a heat map in Figure 2.3). We then empirically show how choosing the wrong source dataset for a given target dataset could decrease the network's performance. Therefore, we provide a DTW based solution to choose the best source dataset for a given target dataset. Finally, we detail a few interesting case studies where the behavior of the proposed method has a significant impact on the transfered model's accuracy.

Transfer learning accuracy variation matrix

In order to have a fair comparison across the datasets, we illustrate the variation in the transferred model's accuracy based on the percentage of variation compared to

[1] http://germain-forestier.info/src/bigdata2018/
[2] https://github.com/hfawaz/bigdata18

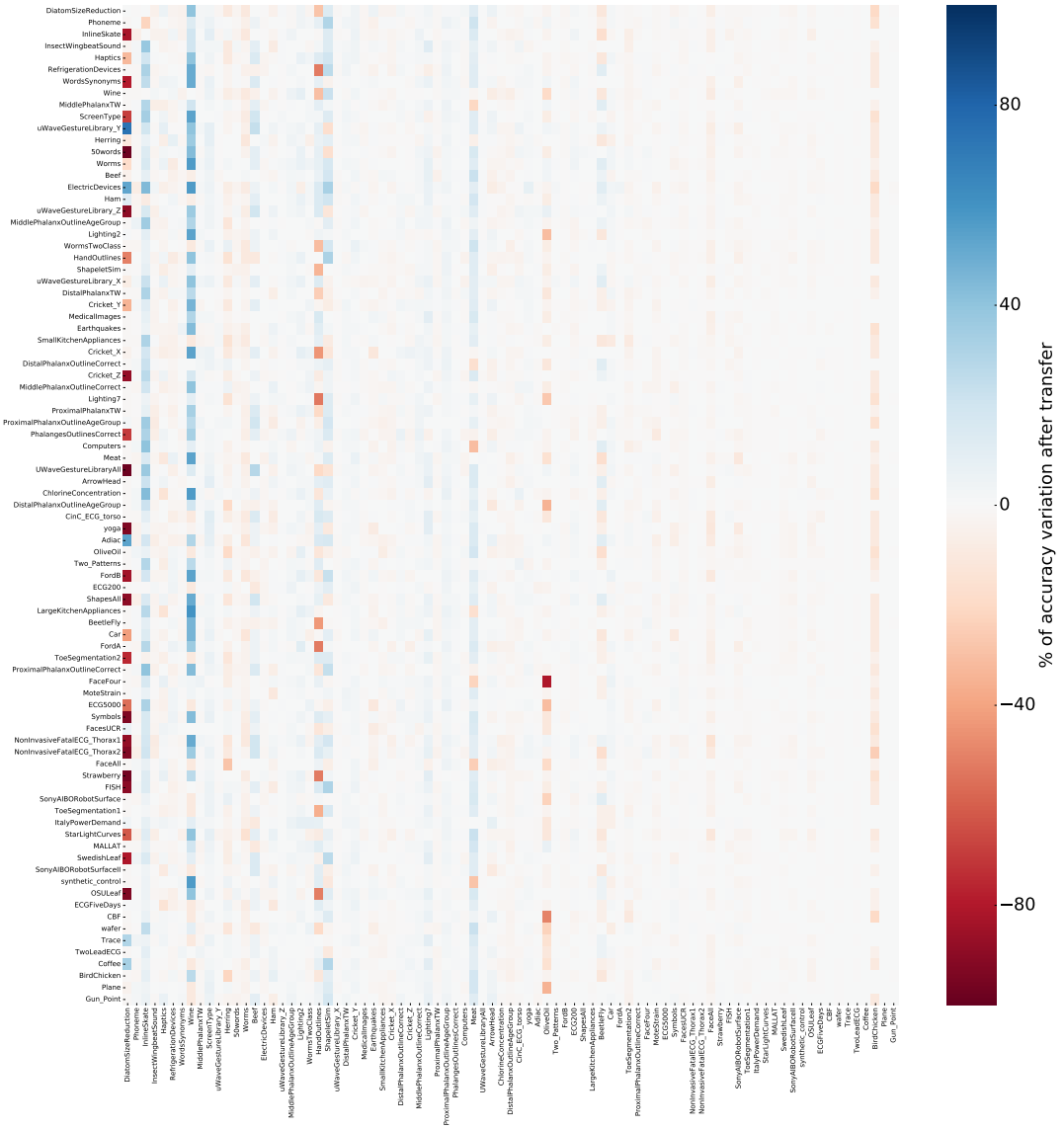


FIGURE 2.3: The variation in percentage over the original accuracy when fine tuning a pre-trained model. The rows' indexes correspond to the source datasets and the columns' indexes correspond to the target datasets. The **red** color shows the extreme case where the chosen pair of datasets (source and target) deteriorates the network's performance. Where on the other hand, the **blue** color identifies the improvement in accuracy when transferring the model from a certain source dataset and fine-tuning on another target dataset. The white color means that no change in accuracy has been identified when using the transfer learning method for two datasets. The matrix actually has a size of 85×85 (instead of 85×84) for visual clarity with its diagonal left out of the analysis. (Best viewed in color).

the original accuracy (without transfer learning). For example, consider the original accuracy (equal to 74.6%) when training the neural network from scratch on the target dataset HandOutlines. Then instead of training the model from scratch (with random initializations) we obtain a 86.5% accuracy when initializing the network's weights to be equal to the weights of a pre-trained network on the source dataset MedicalImages. Hence, the percentage of accuracy variation with respect to the original value is equal to $100 \times (86.5 - 74.6)/74.6 \approx +16\%$. Thus negative values (red in Figure 2.3) indicate a decrease in performance when using the transfer learning approach. Whereas, a positive percentage (blue in Figure 2.3) indicates an increase in performance when fine-tuning a pre-trained model.

When observing the heat map in Figure 2.3, one can easily see that fine-tuning a pre-trained model almost never hurts the performance of the CNN. This can be seen by the dominance of the white color in the heat map, which corresponds to almost no variation in accuracy.

On the other hand, the results which we found interesting are the two extreme cases (red and blue) where the use of transfer learning led to high variations in accuracy. Interestingly for a given target dataset, the choice of source dataset could deteriorate or improve the CNN's performance as we will see in the following subsection.

Naive transfer learning

While observing the heat map in Figure 2.3, we can easily see that certain target datasets (columns) exhibit a high variance of accuracy improvements when varying the source datasets. Therefore, to visualize the worst and best case scenarios when fine-tuning a model against training from scratch, we plotted in Figure 2.4 a pairwise comparison of three aggregated accuracies $\{minimum, median, maximum\}$.

For each target dataset D_t, we took its minimum accuracy among the source datasets and plot it against the model's accuracy when trained from scratch. This corresponds to the red dots in Figure 2.4. By taking the minimum, we illustrate how one can *always* find a bad source dataset for a given target dataset and decrease the model's original accuracy when fine-tuning a pre-trained network.

On the other hand, the maximum accuracy (blue dots in Figure 2.4) shows that there is also *always* a case where a source dataset increases the accuracy when using the transfer learning approach.

As for the median (yellow dots in Figure 2.4), it shows that on average, pre-training and then fine-tuning a model on a target dataset improves without significantly hurting the model's performance.

One extreme case, where the choice of the source dataset had a huge impact on the model's accuracy, is the OliveOil dataset. Precisely the accuracy decreased from 93.3% to 16.7% when choosing respectively MALLAT and FaceFour as source datasets.

This analysis showed us that blindly and naively using the transfer learning approach could drastically decrease the model's performance. Actually, this is largely due to the fact that the initial weights of the network have a significant impact on the training (Glorot and Bengio, 2010). This problem has been identified as *negative transfer learning* in the literature, where there still exists a need to quantify the amount of relatedness between the source and target datasets and whether an attempt to transfer knowledge from the source to the target domain should be made (Weiss, Khoshgoftaar, and Wang, 2016). Therefore in the following paragraph, we show how our similarity based solution can quantify this relatedness between

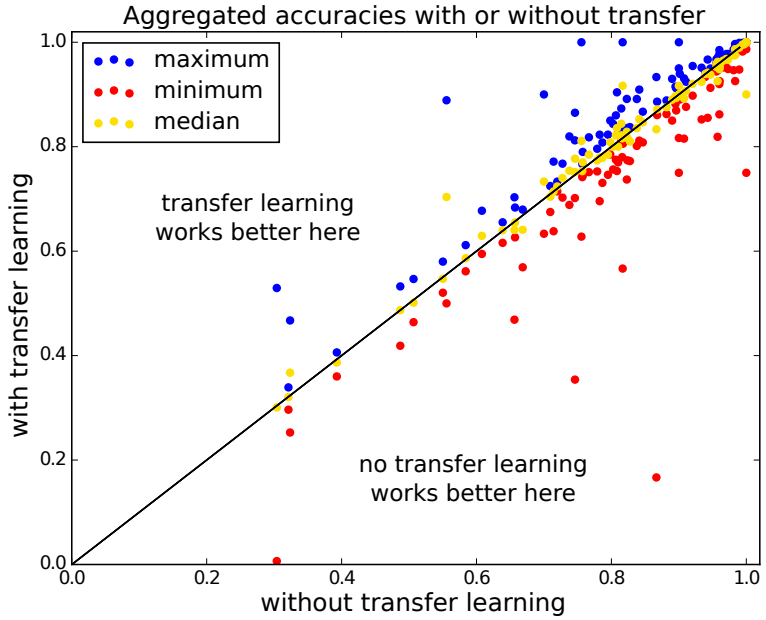


FIGURE 2.4: The three aggregated accuracies (minimum, median and maximum) of the Convolutional Neural Networks with the transfer learning approach against no transfer learning.

the source and the target, thus enabling us to predict the best source dataset for a given target dataset.

Smart transfer learning

In order to know in advance which source dataset is suited for which target dataset, we propose to leverage the similarity between two datasets. Our method is designed specifically for time series data without any previous domain knowledge about the datasets. Using the method we described in subsection 2.2.2, we managed to compute a nearest neighbor for a target dataset and set this nearest neighbor to be the chosen source dataset for the current target dataset in question.

The results showed that this proposed DTW based method will help in achieving what is called *positive transfer* (Weiss, Khoshgoftaar, and Wang, 2016). As opposed to *negative transfer*, positive transfer learning means that the learning algorithm's accuracy increases when fine-tuning a pre-trained model compared to a training from scratch approach (Weiss, Khoshgoftaar, and Wang, 2016).

Figure 2.5 shows a pairwise accuracy plot for two approaches: a random selection process of the source dataset against a "smart" selection of the source dataset using a nearest neighbor algorithm with the distance calculated in algorithm 1. In order to reduce the bias due to the random seed, the accuracy for the random selection approach was averaged over 1000 iterations. This plot shows that on average, choosing the most similar dataset using our method is significantly better than a random selection approach (with $p < 10^{-7}$ for the Wilcoxon signed-rank test). Respectively our method wins, ties and loses on 71, 0 and 14 datasets against randomly choosing the source dataset. We should also note that for the two datasets DiatomSizeReduction and Wine, the nearest neighbor is not always the best choice. Actually, we found that the second nearest neighbor increases drastically the accuracy from 3.3% to 46.7% for DiatomSizeReduction and from 51.9% to 77.8% for Wine (see the 2^{nd} NN dots in Figure 2.5). This means that certain improvements could be incorporated to

our inter-datasets similarity calculation such as adding a warping window (Dau et al., 2017) or changing the number of prototypes for each class which we aim to study in our future work.

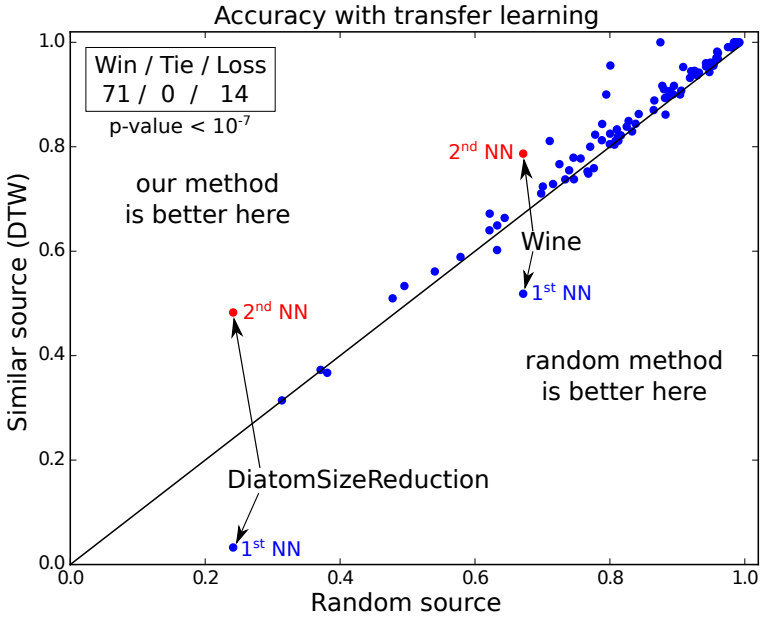


FIGURE 2.5: The accuracy of a fine-tuned model for two cases: (x axis) when the source dataset is selected randomly; (y axis) when the source dataset is selected using our Dynamic Time Warping based solution.

Therefore, since in Figure 2.5 the most similar dataset is the only one that is considered as a potential source for a given target, another interesting study would be to analyze the accuracy on a given target dataset as a function of how dissimilar the source dataset is. However due to the huge number of datasets in the UCR/UEA archive compared to the space limitation, we chose to only study the most interesting cases where the results can be visually interpreted.

Interesting case studies

In this final analysis we chose to work with three interesting target datasets: *ShapeletSim*, *HandOutlines* and *Meat*. These datasets were chosen for different reasons such as the small size of the training set, the relatedness to shapelets and the transfer learning's accuracy variation.

ShapeletSim contains one of the smallest training sets in the UCR/UEA archive (with 20 training instances). Additionally, this dataset is a simulated dataset designed specifically for shapelets which makes it interesting to see how well CNNs can fine-tune (pre-learned) shapelets (Cui, Chen, and Chen, 2016) when varying the source dataset. Figure 2.6 shows how the model's accuracy decreases as we go further from the target dataset. Precisely the average accuracy for the top 3 neighbors reaches 93% compared to the original accuracy of 76%. Actually, we found that the closest dataset to *ShapeletSim* is the RefrigerationDevices dataset which contains readings from 251 households with the task to identify three classes: Fridge, Refrigerator and Upright Freezer. This is very interesting since using other background knowledge one cannot easily predict that using RefrigerationDevices as a source for ShapeletSim will lead to better accuracy improvement. To understand better

this source/target association, we investigated the shapes of the time series of each dataset and found that both datasets exhibit very similar spiky subsequences which is likely the cause for the transfer learning to work between these two datasets.

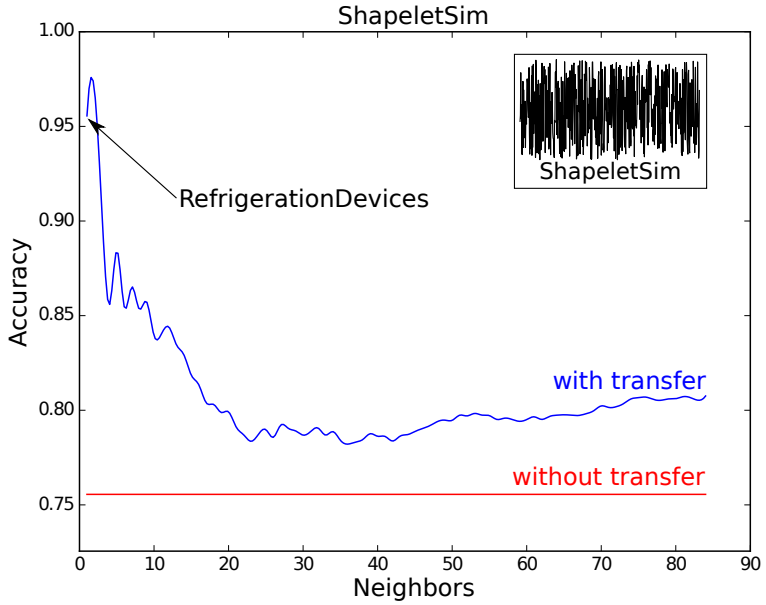


FIGURE 2.6: The fine-tuned model's accuracy variation on the target dataset ShapeletSim with respect to the chosen source dataset neighbor (smoothed for visual clarity - best viewed in color).

HandOutlines is one of the datasets where fine-tuning a pre-trained model almost never improves the accuracy. Unlike *ShapeletSim*, this dataset contains enough labeled data for the learning algorithm to learn from (with 1000 time series in the training set). Surprisingly, we found that one could drastically increase the model's performance when choosing the best source dataset. Figure 2.7 shows a huge difference (10%) between the model's accuracy when fine-tuned using the most *similar* source dataset and the accuracy when choosing the most *dissimilar* source dataset. *HandOutlines* is a classification problem that uses the outlines extracted from hand images. We found that the two most similar datasets (50words and WordsSynonyms) that yielded high accuracy improvements, are also words' outlines extracted from images of George Washington's manuscripts.

Meat is one of the smallest datasets (with 20 training instances) where the transfer learning approach was almost always beneficial. However, we would like to examine the possibility of improving the accuracy even for the case where the transfer learning seems to be positive (Weiss, Khoshgoftaar, and Wang, 2016) for any choice of source dataset. Figure 2.8 shows that the accuracy reaches almost 95% for the top 3 closest datasets and then decrease the less similar the source and target datasets are. While investigating these similarities, we found the top 1 and 3 datasets to be respectively Strawberry and Beef which are all considered spectrograph datasets (Bagnall et al., 2017). As for the second most similar dataset, our method determined it was 50words. Given the huge number of classes (fifty) in 50words our method managed to find some latent similarity between the two datasets which helped in improving the accuracy of the transfer learning process.

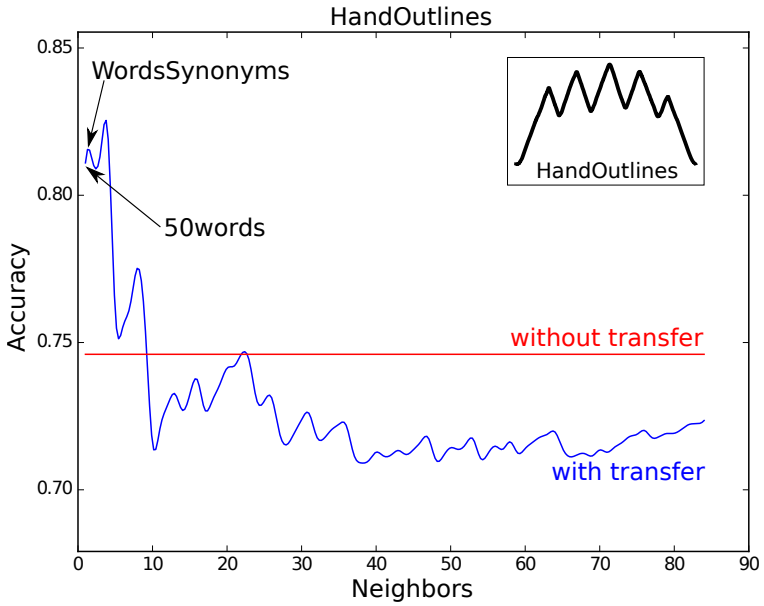


FIGURE 2.7: The fine-tuned model's accuracy variation on the target dataset HandOutlines with respect to the chosen source dataset neighbor (smoothed for visual clarity - best viewed in color).

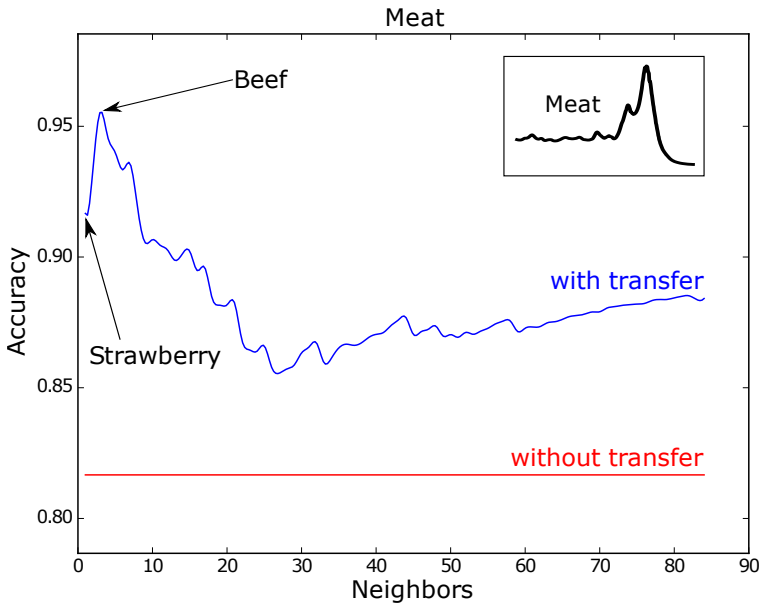


FIGURE 2.8: The fine-tuned model's accuracy variation on the target dataset Meat with respect to the chosen source dataset neighbor (smoothed for visual clarity - best viewed in color).

2.2.6 Conclusion

In this section, we investigated the transfer learning approach on a state of the art deep learning model for TSC problems. Our extensive experiments with every possible combination of source and target datasets in the UCR/UEA archive, were evidence that the choice of the source dataset could have a significant impact on the model's generalization capabilities. Precisely when choosing a bad source dataset for a given target dataset, the optimization algorithm can be stuck in a local optimum. This phenomena has been identified in the transfer learning literature by *negative transfer learning* which is still an active area of research (Weiss, Khoshgoftaar, and Wang, 2016). Thus, when deploying a transfer learning approach, the big data practitioner should give attention to the relationship between the target and the chosen source domains.

These observations motivated us to examine the use of the well known time series similarity measure DTW, to predict the choice of the source dataset when fine-tuning a model on a time series target dataset. After applying this transfer learning guidance, we concluded that transferring deep CNNs on a target dataset works best when fine-tuning a network that was pre-trained on a similar source dataset. These findings are very interesting since no previous observation made the link between the space induced by the classic DTW and the features learned by the Convolutional Neural Networks.

Our results should motivate the big data practitioners to no longer train models from scratch when classifying time series, but instead to fine-tune pre-trained models. Especially because CNNs, if designed properly, can be adapted across different time series datasets with varying length.

In our future work, we aim again to reduce the deep neural network's overfitting phenomena by generating synthetic data using a Weighted DTW Barycenter Averaging method (Forestier et al., 2017b), since the latter distance gave encouraging results in guiding a complex deep learning tool such as transfer learning.

Finally, with big data time series repositories becoming more frequent (Zoumpatianos, Idreos, and Palpanas, 2014), leveraging existing source datasets that are similar to, but not exactly the same as a target dataset of interest, makes a transfer learning method an enticing approach.

2.3 Ensembling

TSC tasks differ from traditional classification tasks by the natural temporal ordering of their attributes (Bagnall et al., 2017). To tackle this problem, a huge amount of research was dedicated into coupling and enhancing time series similarity measures with an NN classifier (Dau et al., 2017; Gharghabi et al., 2018). In Lines and Bagnall, 2015, ten elastic distances were compared to the traditional DTW algorithm to find out that no single measure could outperform the classic NN-DTW for TSC. These findings motivated the authors to construct a single EE classifier that includes all eleven different similarity measures, and achieve a significant improvement compared to the individual classifiers (Lines and Bagnall, 2015). Hence, recent contributions were focused on ensembling different discriminant classifiers such as decision trees (Baydogan, Runger, and Tuv, 2013) and SVMs (Bostrom and Bagnall, 2015) on different data representation techniques such shapelet transform (Bostrom and Bagnall, 2015) or DTW features (Kate, 2016). These ideas gave rise to COTE (Bagnall et al., 2016) and its extended version HIVE-COTE (Lines, Taylor, and Bagnall, 2018) where 37 different classifiers were ensembled over multiple time series data

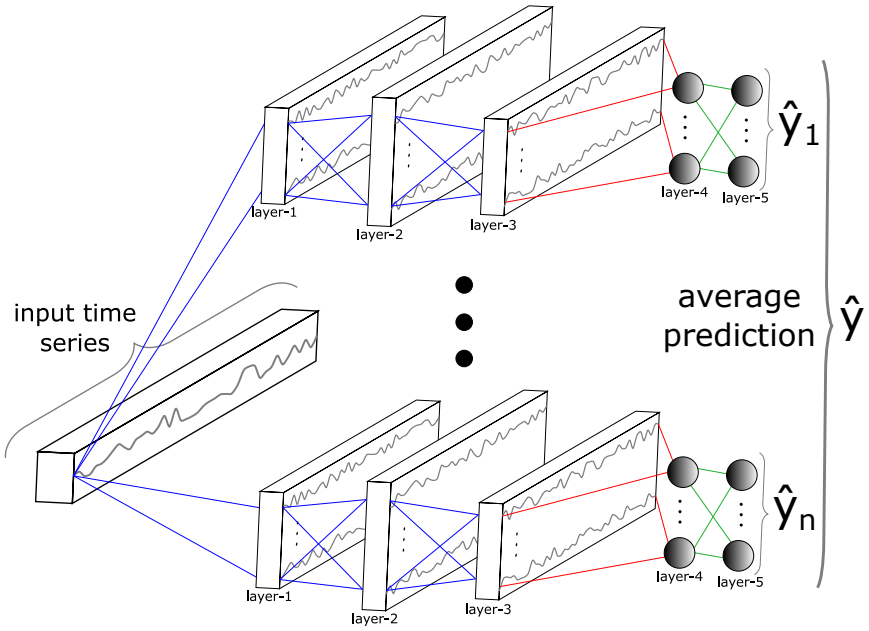


FIGURE 2.9: Ensemble of deep convolutional neural networks for time series classification.

transformation techniques in order to reach current state-of-the-art performance for TSC (Bagnall et al., 2017).

With the advent of deep neural networks into industrial and commercial applications such as self-driving cars (Qiu and Gu, 2018) and speech recognition systems (Liul et al., 2018), time series data mining practitioners started investigating the application of deep learning to TSC problems (Wang, Yan, and Oates, 2017). In the previous chapter, we showed how deep CNNs are able to achieve results that are not significantly different than current state-of-the-art algorithms for TSC problems when evaluated over the 85 time series datasets from the UCR/UEA archive (Dau et al., 2019). These results suggest that building upon deep learning based solutions for TSC could further improve the current state-of-the-art performance of deep neural networks.

One way of improving neural network based classifiers is to build an ensemble of deep learning models. This idea seems very interesting for TSC tasks since the state-of-the-art is moving towards ensembled solutions (Lines, Taylor, and Bagnall, 2018; Lines and Bagnall, 2015; Bagnall et al., 2017; Baydogan, Runger, and Tuv, 2013). In addition, deep neural network ensembles seem to achieve very promising results in many supervised machine learning domains such as skin lesions detection (Goyal and Rajapakse, 2018), facial expression recognition (Wen et al., 2017) and automatic bucket filling (Dadhich, Sandin, and Bodin, 2018).

Therefore, we propose to ensemble the current state-of-the-art deep learning models for TSC developed in the previous chapter, by constructing one model composed of 60 different deep neural networks: 6 different architectures (Wang, Yan, and Oates, 2017; Zheng et al., 2014; Zhao et al., 2017; Serrà, Pascual, and Karatzoglou, 2018) each one with 10 different initial weight values. By evaluating on the 85 datasets from the UCR/UEA archive, we demonstrate a significant improvement over the individual classifiers while also reaching very similar performance to HIVE-COTE: the current state-of-the-art ensemble of 37 non deep learning based time series classifiers. Finally, inspired by the our finding on transfer learning in the previous section, we replace ensembling randomly initialized networks with an

ensemble constructed out of fine-tuned models from 84 different source datasets, which showed a significant improvement for TSC problems.

2.3.1 Background

In this subsection we present some work related to ensembling neural network classifiers, with a focus on time series data.

Constructing an ensemble of many deep learning classifiers has been shown to achieve high performance in many different fields. In Goyal and Rajapakse, 2018, an ensemble of two neural networks was adopted: (1) Inception-v4 and (2) Inception-ResNet-v2. Both of these classifiers are learned with a joint meta-learning approach in an end-to-end manner. A forest CNN was proposed by Lee, Kang, and Kang, 2017 for image classification, where similarly to random forest, the ensemble is constructed by replacing the individual nodes with a CNN and finally the classifier's decision is taken by performing a majority voting scheme over the different decisions of the individual trees in the forest. Another ensemble of CNNs for facial expression recognition was proposed in Wen et al., 2017 where each individual classifier was trained independently to output a probability for each class and then the network's final decision was taken using a probability-based fusion method. In Dadhich, Sandin, and Bodin, 2018, an ensemble of neural networks was found to outperform other hybrid machine learning ensembles when solving an automatic bucket filling problem. Finally in Ienco and Pensa, 2018, deep auto-encoders were ensembled in order to learn an unsupervised latent representation of the input data over multiple resolutions, thus improving the quality of the produced clusters.

Although in almost all use cases ensembling deep neural networks almost always yields to better decisions, we did not find any approach using a neural network ensemble for domain agnostic TSC. Perhaps the work in Jin and Dong, 2016 is the closest to ours where a neural network based ensemble was used to perform biomedical TSC, where individual architectures were constructed with some domain knowledge specific to the classification problem at hand such as choosing the filter length with local and distorted views. Therefore, we decided to further explore ensembling deep neural networks for TSC, by combining multiple deep learning models in different settings.

2.3.2 Methods

In this subsection, we start by presenting the six different architectures composing our ensembles of neural networks. For completeness, we describe the random initialization technique adopted for all models. Finally, we present a transfer learning based alternative to randomly initializing the weights of the networks.

Architectures

The average rank of the six chosen deep learning classifiers, over the 85 datasets from the UCR/UEA archive (Dau et al., 2019; Bagnall et al., 2017) is listed in Table 2.1. All of these architectures were implemented in a common framework during our empirical study presented in the previous chapter, containing originally 9 different deep learning approaches for TSC. However only 6 out of these 9 approaches were probabilistic classifiers whereas the three other classifiers performed a hard prediction: meaning an input time series is assigned a specific class rather than a probability distribution over all the classes in a dataset. Therefore, we chose to only ensemble

Approach	Rank	Wins
ResNet (Wang, Yan, and Oates, 2017)	1.88	41
FCN (Wang, Yan, and Oates, 2017)	2.49	18
Encoder (Serrà, Pascual, and Karatzoglou, 2018)	3.34	10
MLP (Wang, Yan, and Oates, 2017)	4.08	4
Time-CNN (Zhao et al., 2017)	4.38	4
MCDCNN (Zheng et al., 2014)	4.83	3

TABLE 2.1: Average rank of the six classifiers constituting the Neural Network Ensemble for time series classification over the 85 datasets from the UCR/UEA archive.

the 6 probabilistic models, thus allowing us to combine the networks by averaging the a posteriori probability for each class over the individual classifiers' output. Finally, we present a brief description of these 6 different architectures and refer the interested reader to a more thorough explanation in the corresponding papers. All hyperparameters can be found in Ismail Fawaz et al., 2019d.

Multi-Layer Perceptron (MLP) is the simplest form of deep neural networks and was proposed in Wang, Yan, and Oates, 2017 as a baseline architecture for TSC. The architecture contains three hidden layers, with each one fully connected to the output of its previous layer. The main characteristic of this architecture is the use of a Dropout layer (Srivastava et al., 2014) to reduce overfitting. One disadvantage is that since the input time series is fully connected to the first hidden layer, the temporal information in a time series is lost (Ismail Fawaz et al., 2019d).

Fully Convolutional Neural Network (FCN), originally proposed in Wang, Yan, and Oates, 2017, is considered a competitive architecture yielding the second best results when evaluated on the UCR/UEA archive (see Table 2.1). This network is comprised of three convolutional layers, each one performing a non-linear transformation of the input time series. A global average pooling operation is used before the final softmax classifier, thus reducing drastically the number of parameters in a network and allowing an architecture that is invariant to the length of the input time series. The latter characteristic motivated us to perform a transfer learning technique in Ismail Fawaz et al., 2018d, and ensembling the resulting neural networks which is later discussed in this subsection.

Residual Network (ResNet) was originally proposed in Wang, Yan, and Oates, 2017 and showed similar performance to FCN when evaluated on 44 datasets from the archive. However, when evaluated over the 85 datasets, ResNet significantly outperformed FCN (see Table 2.1). The main characteristic of ResNet is the addition of residual connections which enables a direct flow of the gradient (Wang, Yan, and Oates, 2017).

Encoder was originally proposed in Serrà, Pascual, and Karatzoglou, 2018 as a hybrid CNN that modifies the FCN architecture (Wang, Yan, and Oates, 2017) by mainly adding a Dropout layer (Srivastava et al., 2014) and an attention mechanism. The latter operation enables Encoder to learn to localize which regions of the input time series are useful for a certain class identification.

Multi-Channels Deep Convolutional Neural Networks (MCDCNN) was originally proposed in Zheng et al., 2014 for multivariate TSC and adapted to univariate data by Ismail Fawaz et al., 2019d. It consists of a traditional CNN, where each convolutional layer is followed by a max pooling operation, then a traditional fully connected layer is used before the final softmax classifier.

Time Convolutional Neural Network (Time-CNN) was originally proposed for univariate as well as multivariate TSC (Zhao et al., 2017). Similarly to MCDCNN, this network is a traditional CNN with one major exception: the use of the mean squared error instead of the traditional categorical cross-entropy loss function, which has been used by all the deep learning approaches we have mentioned so far. Therefore for Time-CNN, the sum of the output class probabilities is not guaranteed to be equal to one.

Ensembling models with random initial weights

We have described in the previous subsection, the architecture of six different classifiers. The weights for each network are initialized randomly using Glorot's uniform initialization method (Glorot and Bengio, 2010). This technique ensures a uniform distribution of the initial weight values. However due to non-convexity, networks with the same architecture but different initial weights could yield different validation accuracy. In Choromanska et al., 2015, the authors showed that deeper networks are much more stable with respect to the randomness. This would suggest that ensembling relatively non deep architectures would yield to a much better improvement in accuracy than ensembling deeper architectures. Fortunately, for low dimensional time series data, current state-of-the-art architectures are much less deeper than their counterpart networks for high dimensional images. Therefore, we believe that we can leverage this instability of neural networks for time series data by ensembling the decision taken by the same network but with different random initializations, using the following equation:

$$\hat{y}_{i,c} = \frac{1}{n} \sum_{j=1}^{n} \sigma_c(x_i, \theta_j) \;\; | \;\; \forall c \in [1, C] \tag{2.1}$$

with $\hat{y}_{i,c}$ denoting the ensemble's output probability of having the input time series x_i belonging to class c, which is equal to the logistic output σ_c averaged over the n randomly initialized models. We should note that training an ensemble of the same architecture with different initial weight values has been shown to improve neural network's performance on many computer vision problems (Wen et al., 2017), however, we did not encounter any previous work that combines such classifiers for TSC.

Transfer learning

An alternative to training a deep classifier from scratch is to fine-tune a model that has been already pre-trained on a un/related task (Ismail Fawaz et al., 2018d), which was described in the previous section of this current chapter. This process is called transfer learning, where the network is first trained on a source dataset, then the final layer is removed and replaced with a new randomly initialized softmax layer whose number of neurons is equal to the number of classes in the target dataset. The pre-trained model is then fine-tuned or re-trained on the target dataset's training set. With 85 datasets in the archive, each target dataset will have 84 potential source datasets, which motivated us to ensemble the decision of these 84 FCN models.

2.3.3 Results

In this section we present the results of different ensembling schemes when evaluated on the 85 datasets from the UCR/UEA archive (Dau et al., 2019), which is

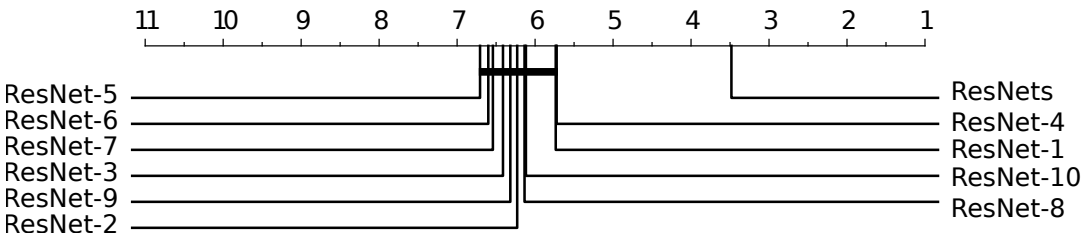


FIGURE 2.10: Critical difference diagram showing the pairwise statistical comparison of ten ResNets with random initializations as well as one ResNet ensemble composed of these ten individual neural networks.

currently the largest publicly available benchmark for time series analysis. In order to compare multiple classifiers over several datasets, we followed the same procedure described in the previous chapter. All experiments were conducted on a hybrid cluster of more than 60 NVIDIA GPUs comprised of GTX 1080 Ti, Tesla K20, K40 and K80. Note that the code and the raw results are publicly available on the project's companion repository[3].

Ensembling randomly initialized models

By ensembling randomly initialized networks, we are able to achieve a significant improvement in accuracy. Figure 2.10 shows a critical difference diagram where ten different random initializations of ResNet did not yield to significantly different results. However, by ensembling these different networks, we were able to demonstrate a significant improvement in the average rank over the 85 datasets. We should note that the latter phenomenon was also observed for the five other neural networks described in the previous subsection. Finally, we should emphasize that an ensembling technique will improve the stability of ResNet in terms of accuracy, in other words reducing the bias due to the initial weight values as well as the randomness induced by gradient descent based optimization.

Ensembling all neural networks

After demonstrating that using an ensemble of neural networks is always better than a single classifier, we sought to answer the following question: *Could an ensemble of hybrid randomly initialized networks achieve even better performance?* Figure 2.11 shows a critical difference diagram containing six ensembles of homogenized networks as well as the hybrid ensemble of *all* available networks. The latter classifier contains sixty different networks: each architecture (six in total) is initialized with ten different random weight values. The results show that ensembling all networks was able to outperform all classifiers. However the statistical test failed to find any significant difference between the full ensemble and individual ResNet/FCN ensembles. This would suggest that the ensemble is highly affected by the poor performance of Time-CNN, MLP and MCDCNN. The latter classifiers showed the worst average rank without any significant difference, thus suggesting that removing them would yield even better performance.

[3] https://github.com/hfawaz/ijcnn19ensemble

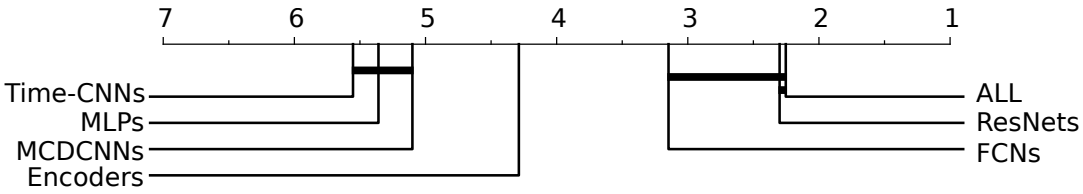


FIGURE 2.11: Critical difference diagram showing the pairwise statistical comparison of six architectures ensembled with ten different random initializations each, as well as one ensemble containing the six models.

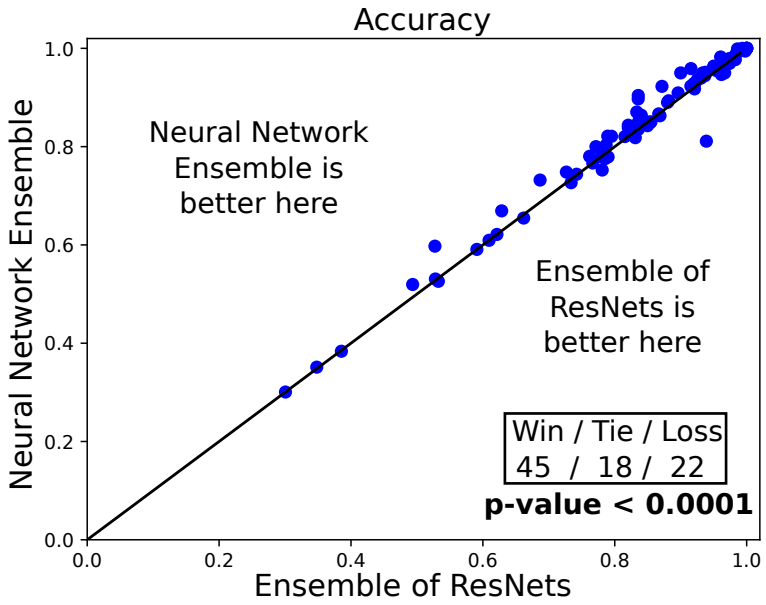


FIGURE 2.12: The Neural Network Ensemble (NNE) composed of ResNet, FCN and Encoder is significantly better than an ensemble of pure ResNets.

Neural Network Ensemble

The results in the previous section, suggest that choosing carefully the classifiers in the pool would yield to a better ensemble. Therefore, we construct an NNE comprised solely of ResNet, FCN and Encoder. These three architectures were the only ones to yield significantly different results when a homogenized ensemble was adopted (Figure 2.11). Further investigations suggested that FCN performs better than ResNet on electrocardiography datasets (Ismail Fawaz et al., 2019d), which would motivate researchers to combine these two classifiers in order to have a robust algorithm that improves the accuracy over the whole datasets. However, for small datasets such as CinCECGTorso, both FCN and ResNet overfitted the dataset very easily with almost 82% test accuracy (Ismail Fawaz et al., 2018b), whereas Encoder managed to achieve very good performance with a 91% accuracy, therefore implying a combination of ResNet, FCN and Encoder would yield to better accuracy on a various range of TSC datasets. Figure 2.12 shows how NNE is able to outperform an ensemble of pure ResNets with 45 wins and 18 ties on 85 datasets from the archive. We believe that the combination of an FCN with ResNet and Encoder, enables the classifier to benefit respectively from the residual linear connections and the attention mechanism.

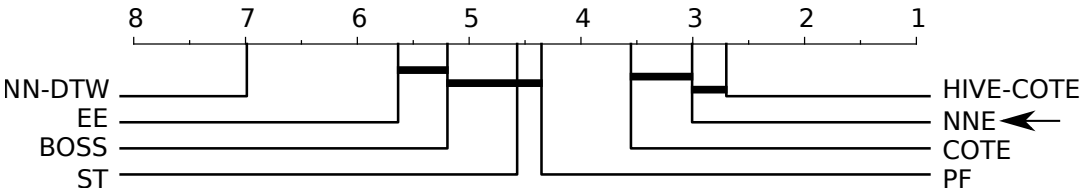


FIGURE 2.13: Critical difference diagram showing the pairwise statistical comparison of current state-of-the-art algorithms with the Neural Network Ensemble (NNE) added to the pool.

To further understand how NNE is performing with respect to current state-of-the-art TSC algorithms, we illustrate in Figure 2.13 a critical difference diagram containing NNE and seven other non deep learning based classiifers: (1) NN-DTW corresponds to the nearest neighbor coupled with the Dynamic Time Warping distance; (2) EE is an ensemble of nearest neighbor classifiers with eleven elastic distances; (3) BOSS corresponds to the ensemble Bag-of-SFA-Symbols; (4) ST is another ensemble of off-the-shelf classifiers computed over the Shapelet Transform data domain; (5) PF or Proximity Forest is an ensemble of decision trees coupled with eleven elastic distances; finally (6) COTE and (7) HIVE-COTE are two ensembles of respectively 35 and 37 classifiers using multiple data transformation techniques. The results for these classifiers were taken from Bagnall et al., 2017 except for PF whose results were taken from the original paper (Lucas et al., 2018). Figure 2.13 clearly shows how our NNE is able to reach state-of-the-art performance for TSC, suggesting that CNNs are able to extract one dimensional discriminant features useful for classification in an end-to-end manner, as opposed to other hand-engineered features used by HIVE-COTE such as the Discrete Fourier Transform, DTW features and the Shapelet Transform.

Ensembling fine-tuned models

Figure 2.14 shows that ensembling fine-tuned FCNs is significantly better than ensembling randomly initialized FCN models that are trained from scratch. However, this transfer learning based ensemble did not manage to outperform ResNets' ensemble nor NNE. These results show that the choice of architecture is very crucial and suggest that an ensemble of transferred ResNets would demonstrate even better performance than an ensemble of pure ResNets or NNE.

2.3.4 Conclusion

In this section, we showed how ensembling deep neural networks can achieve state-of-the-art performance for time series classification. We showed that it would be almost always beneficial to ensemble randomly initialized models rather than choosing one trained neural network out of the ensemble. Finally, we investigated an ensemble of transferred deep CNNs to demonstrate even better performance than ensembling randomly initialized networks. In the future, we would like to consider a meta-ensembling approach where the output logistics of individual deep learning models are fed to a meta-network that learns to map these inputs to the correct prediction.

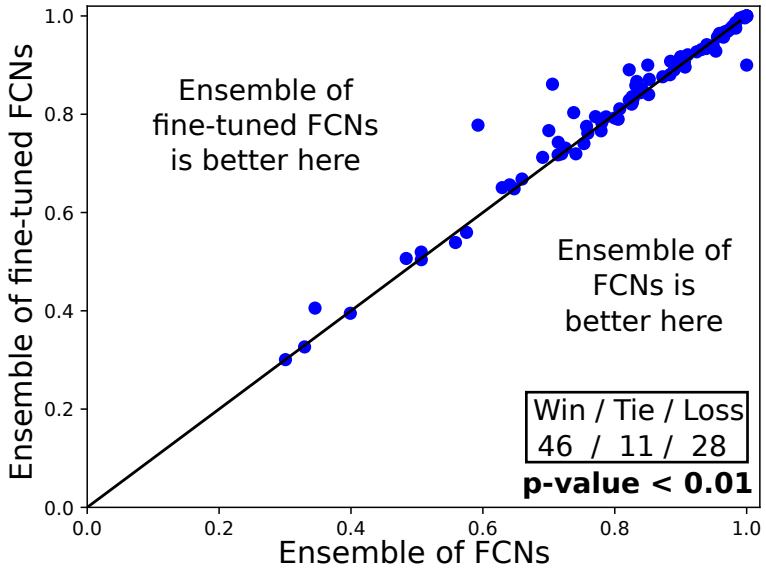


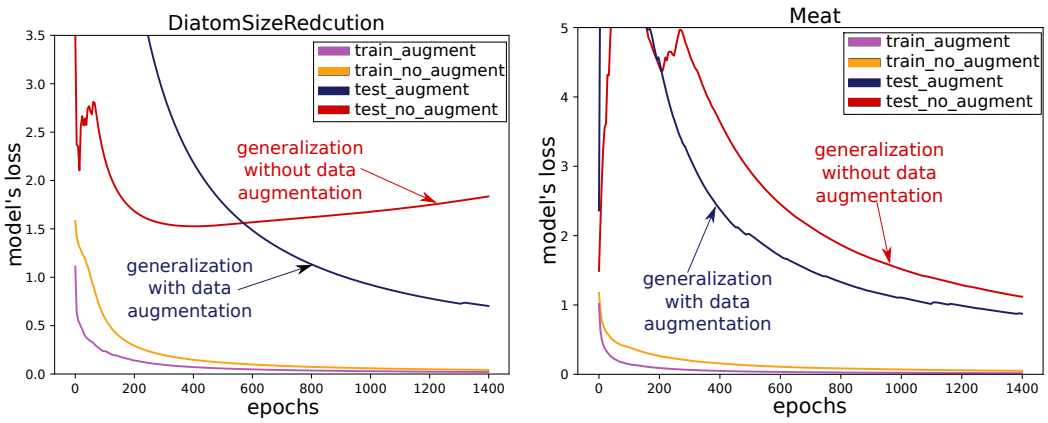
FIGURE 2.14: Ensembling fine-tuned models is significantly better than ensembling randomly initialized FCN models that are trained from scratch.

2.4 Data augmentation

Deep learning usually benefits from large training sets (Zhang et al., 2017). However, for many applications only relatively small training data exist. In TSC, this phenomenon can be observed by analyzing the UCR/UEA archive's datasets (Dau et al., 2019), where 20 datasets have 50 or fewer training instances. These numbers are relatively small compared to the billions of labeled images in computer vision, where deep learning has seen its most successful applications (LeCun, Bengio, and Hinton, 2015).

Although the recently proposed deep CNNs reached state of the art performance in TSC on the UCR/UEA archive (Wang, Yan, and Oates, 2017), they still show low generalization capabilities on some small datasets such as the CinCECGTorso dataset with 40 training instances. This is surprising since the NN-DTW performs exceptionally well on this dataset which shows the relative easiness of this classification task. Thus, inter-time series similarities in such small datasets cannot be captured by the CNNs due to the lack of labeled instances, which pushes the network's optimization algorithm to be stuck in local minimums (Zhang et al., 2017). Figure 2.15 illustrates on an example that the lack of labeled data can sometimes be compensated by the addition of synthetic data.

This phenomenon, also known as *overfitting* in the machine learning community, can be solved using different techniques such as regularization or simply collecting more *labeled* data (Zhang et al., 2017) (which in some domains are hard to obtain). Another well-known technique is data augmentation, where synthetic data are generated using a specific method. For example, images containing street numbers on houses can be slightly rotated without changing what number they actually are (Krizhevsky, Sutskever, and Hinton, 2012). For deep learning models, these methods are usually proposed for image data and do not generalize well to time series (Um et al., 2017). This is probably due to the fact that for images, a visual comparison can confirm if the transformation (such as rotation) did not alter the image's class, while for time series data, one cannot easily confirm the effect of such ad-hoc transformations on the nature of a time series. This is the main reason why data



(A) DiatomSizeReduction

(B) Meat

FIGURE 2.15: The model's loss with/without data augmentation on the DiatomSizeReduction and Meat datasets (smoothed and clipped for visual clarity).

augmentation for TSC have been limited to mainly two relatively simple techniques: slicing and manual warping, which are further discussed in the next subsection.

In this section, we propose to leverage from a DTW based data augmentation technique specifically developed for time series, in order to boost the performance of a deep ResNet for TSC. Our preliminary experiments reveal that data augmentation can drastically increase the accuracy for CNNs on some datasets while having a small negative impact on other datasets. We finally propose to combine the decision of the two trained models and show how it can reduce significantly the rare negative effect of data augmentation while maintaining its high gain in accuracy on other datasets.

2.4.1 Related work

The most used data augmentation method for TSC is the slicing window technique, originally introduced for deep CNNs in Cui, Chen, and Chen, 2016. The method was originally inspired by the image cropping technique for data augmentation in computer vision tasks (Zhang et al., 2017). This data transformation technique can, to a certain degree, guarantee that the cropped image still holds the same information as the original image. On the other hand, for time series data, one cannot make sure that the discriminative information has not been lost when a certain region of the time series is cropped. Nevertheless, this method was used in several TSC problems, such as in Krell, Seeland, and Kim, 2018 where it improved the SVMs accuracy for classifying electroencephalographic time series. In Kvamme et al., 2018, this slicing window technique was also adopted to improve the CNNs' mortgage delinquency prediction using customers' historical transactional data. In addition to the slicing window technique, jittering, scaling, warping and permutation were proposed in Um et al., 2017 as generic time series data augmentation approaches. The authors in Um et al., 2017 proposed an additional data augmentation method specific to wearable sensor time series data that rotates the trajectory of a person's arm around an axis (e.g. the x axis).

In Le Guennec, Malinowski, and Tavenard, 2016, the authors proposed to extend the slicing window technique with a warping window that generates synthetic time

series by warping the data through time. This method was used to improve the classification of their deep CNN for TSC, which was also shown to significantly decrease the accuracy of a NN-DTW classifier when compared to our adopted data augmentation algorithm (Forestier et al., 2017b). We should note that the use of a window slicing technique means that the model should classify each subsequence alone and then finally classify the whole time series using a majority voting approach. Alternatively, our method does not crop time series into shorter subsequences which enables the network to learn discriminative properties from the whole time series in an end-to-end manner.

2.4.2 Method

Architecture

We have chosen to improve the generalization capability of the deep ResNet proposed in Wang, Yan, and Oates, 2017 for two main reasons, whose corresponding architecture is illustrated in Figure 1.7. First, by adopting an already validated architecture, we can attribute any improvement in the network's performance solely to the data augmentation technique. The second reason is that ResNet (Wang, Yan, and Oates, 2017), to the best of our knowledge, is the deepest neural network validated on large number of TSC tasks (such as the UCR/UEA archive (Dau et al., 2019)), which according to the deep learning literature will benefit the most from the data augmentation techniques as opposed to shallow architectures (Bengio et al., 2011). Deep ResNets were first proposed by He et al., 2016 for computer vision tasks. They are mainly composed of convolutions, with one important characteristic: the residual connections which acts like shortcuts that enable the flow of the gradient directly through these connections.

The input of this network is a univariate time series with a varying length l. The output consists of a probability distribution over the C classes in the dataset. The network's core contains three residual blocks followed by a Global Average Pooling layer and a final softmax classifier with C neurons. Each residual block contains three 1-D convolutions of respectively 8, 5 and 3 filter lengths. Each convolution is followed by a batch normalization (Ioffe and Szegedy, 2015) and a ReLU as the activation function. The residual connection consists in linking the input of a residual block to the input of its consecutive layer with the simple addition operation. The number of filters in the first residual blocks is set to 64 filters, while the second and third blocks contain 128 filters.

All network's parameters were initialized using Glorot's Uniform initialization method (Glorot and Bengio, 2010). These parameters were learned using Adam (Kingma and Ba, 2015) as the optimization algorithm. Following Wang, Yan, and Oates, 2017, without any fine-tuning, the learning rate was set to 0.001 and the exponential decay rates of the first and second moment estimates were set to 0.9 and 0.999 respectively. Finally, the categorical cross-entropy was used as the objective cost function during the optimization process.

Data augmentation

The data augmentation method we have chosen to test with this deep architecture, was first proposed in Forestier et al., 2017b to augment the training set for a 1-NN coupled with the DTW distance in a cold start simulation problem. In addition, the 1-NN was shown to sometimes benefit from augmenting the size of the train set even when the whole dataset is available for training. Thus, we hypothesize that

this synthetic time series generation method should improve deep neural network's performance, especially that the generated examples in Forestier et al., 2017b were shown to closely follow the distribution from which the original dataset was sampled. The method is mainly based on a weighted form of DBA technique (Petitjean, Ketterlin, and Gançarski, 2011; Petitjean et al., 2016; Petitjean et al., 2014). The latter algorithm averages a set of time series in a DTW induced space and by leveraging a weighted version of DBA, the method can thus create an infinite number of new time series from a given set of time series by simply varying these weights. Three techniques were proposed to select these weights, from which we chose only one in our approach for the sake of simplicity, although we consider evaluating other techniques in our future work. The weighting method is called Average Selected which consists of selecting a subset of close time series and fill their bounding boxes.

We start by describing in details how the weights are assigned, which constitutes the main difference between an original version of DBA and the weighted version originally proposed in Forestier et al., 2017b. Starting with a random initial time series chosen from the training set, we assign it a weight equal to 0.5. The latter randomly selected time series will act as the initialization of DBA. Then, we search for its 5 nearest neighbors using the DTW distance. We then randomly select 2 out these 5 neighbors and assign them a weight value equal to 0.15 each, making thus the total sum of assigned weights till now equal to $0.5 + 2 \times 0.15 = 0.8$. Therefore, in order to have a normalized sum of weights (equal to 1), the rest of the time series in the subset will share the rest of the weight 0.2. We should note that the process of generating synthetic time series leveraged only the training set thus eliminating any bias due to having seen the test set's distribution.

As for computing the average sequence, we adopted the DBA algorithm in our data augmentation framework. Although other time series averaging methods exist in the literature, we chose the weighted version of DBA since it was already proposed as a data augmentation technique to solve the cold start problem when using a nearest neighbor classifier (Forestier et al., 2017b). Therefore we emphasize that other weighted averaging methods such as soft-DTW (Cutur and Blondel, 2017) and TEKA (Marteau, 2019) could be used instead of DBA in our framework, but we leave such exploration for our future work.

We did not test the effect of imbalanced classes in the training set and how it could affect the model's generalization capabilities. Note that imbalanced time series classification is a recent active area of research that merits an empirical study of its own (Geng and Luo, 2018). At last, we should add that the number of generated time series in our framework was chosen to be equal to double the amount of time series in the most represented class (which is a hyper-parameter of our approach that we aim to further investigate in our future work).

2.4.3 Results

We evaluated the data augmentation method for ResNet on the UCR/UEA archive (Dau et al., 2019), which is the largest publicly available TSC benchmark. The archive is composed of datasets from different real world applications with varying characteristics such the number of classes and the size of the training set. Finally, for training the deep learning models, we leveraged the high computational power of more than 60 GPUs in one huge cluster[4] We should also note that the same parameters' initial values were used for all compared approaches, thus eliminating any bias due to the random initialization of the network's weights.

[4]Our source code is available on https://github.com/hfawaz/aaltd18

Our results show that data augmentation can drastically improve the accuracy of a deep learning model while having a small negative impact on some datasets in the worst case scenario. Figure 2.16a shows the difference in accuracy between ResNet with and without data augmentation, it shows that the data augmentation technique does not lead a significant decrease in accuracy. Additionally, we observe a huge increase of accuracy for the DiatomSizeReduction dataset (the accuracy increases from 30% to 96% when using data augmentation).

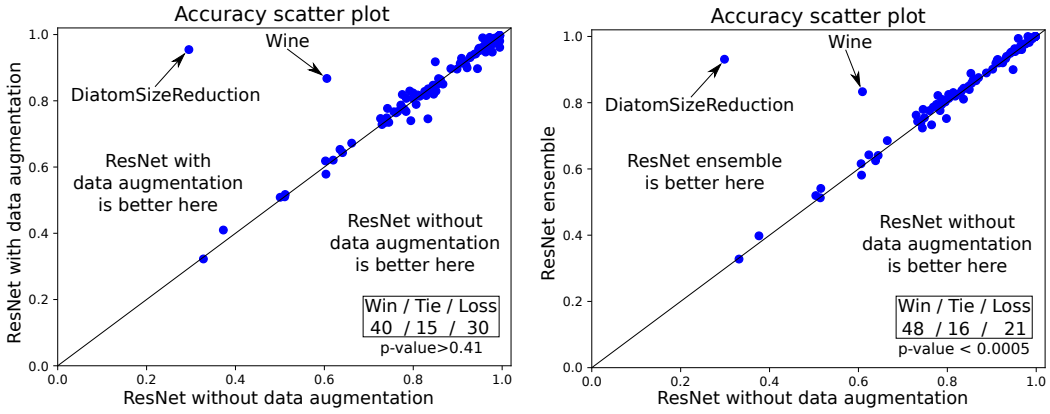
This result is very interesting for two main reasons. First, DiatomSizeReduction has the smallest training set in the UCR/UEA archive (Dau et al., 2019) (with 16 training instances), which shows the benefit of increasing the number of training instances by generating synthetic time series. Secondly, the DiatomSizeReduction dataset is the one where ResNet yield the worst accuracy without augmentation. On the other hand, the NN coupled with DTW (or ED) gives an accuracy of 97% which shows the relative easiness of this dataset where time series exhibit similarities that can be captured by the simple Euclidean distance, but missed by the deep ResNet due to the lack of training data (which is compensated by our data augmentation technique). The results for the Wine dataset (57 training instances) also show an important improvement when using data augmentation.

While we did show that deep ResNet can benefit from synthetic time series on some datasets, we did not manage to show any significant improvement over the whole UCR/UEA archive (p-value > 0.41 for the Wilcoxon signed rank test). Therefore, we decided to leverage an ensemble technique where we take into consideration the decisions of two ResNets (trained with and without data augmentation). In fact, we average the a posteriori probability for each class over both classifier outputs, then assign for each time series the label for which the averaged probability is maximum, thus giving a more robust approach to out-of-sample generated time series. The results in Figure 2.16b show that the datasets which benefited the most from data augmentation exhibit almost no change to their accuracy improvement. While on the other hand the number of datasets where data augmentation harmed the model's accuracy decreased from 30 to 21. The Wilcoxon signed rank test shows a significant difference (p-value < 0.0005). The ensemble's results are in compliance with the recent consensus in the TSC community, where ensembles tend to improve the individual classifiers' accuracy (Bagnall et al., 2017).

2.4.4 Conclusion

In this section, we showed how overfitting small time series datasets can be mitigated using a recent data augmentation technique that is based on DTW and a weighted version of the DBA algorithm. These findings are very interesting since no previous observation made a link between the space induced by the classic DTW and the features learned by the CNNs, whereas our experiments showed that by providing enough time series, CNNs are able to learn time invariant features that are useful for classification.

In our future work, we aim to further test other variant weighting schemes for the DTW-based data augmentation technique, while providing a method that predicts when and for which datasets, data augmentation would be beneficial.



(A) ResNet with/without augmentation

(B) ResNet ensemble vs ResNet

FIGURE 2.16: Accuracy of ResNet with and/or without data augmentation.

2.5 Adversarial examples

As we have previously discussed, TSC problems are encountered in various real world data mining tasks ranging from health care (Abdelfattah, Abdelrahman, and Wang, 2018; Ma, Xiao, and Wang, 2018; Ismail Fawaz et al., 2018c) and security (Tan, Webb, and Petitjean, 2017; Tobiyama et al., 2016) to food safety (Briandet, Kemsley, and Wilson, 1996; Nawrocka and Lamorska, 2013) and power consumption monitoring (Owen and Foreman, 2012; Zheng et al., 2018). With deep learning models revolutionizing many machine learning fields such as computer vision (Krizhevsky, Sutskever, and Hinton, 2012) and natural language processing (Yang et al., 2018; Wang, Li, and Xu, 2018), we have shown in the previous chapter how researchers recently started to adopt these models for TSC tasks (Ismail Fawaz et al., 2019d).

Following the advent of deep learning, researchers started to study the vulnerability of deep networks to adversarial attacks (Yuan et al., 2017). In the context of image recognition, an adversarial attack consists in modifying an original image so that the changes are almost undetectable by a human (Yuan et al., 2017). The modified image is called an adversarial image, which will be misclassified by the neural network, while the original one is correctly classified. One of the most famous real-life attacks consists in altering a traffic sign image so that it is misinterpreted by an autonomous vehicle (Eykholt et al., 2018). Another application is the alteration of illegal content to make it undetectable by automatic moderation algorithms (Yuan et al., 2017). The most common attacks are gradient-based methods, where the attacker modifies the image in the direction of the gradient of the loss function with respect to the input image thus increasing the misclassification rate (Goodfellow, Shlens, and Szegedy, 2015; Kurakin, Goodfellow, and Bengio, 2017; Yuan et al., 2017).

While these approaches have been intensely studied in the context of image recognition (e.g. NeurIPS competition on Adversarial Vision Challenge), adversarial attacks haven not been thoroughly explored for TSC. This is surprising as deep learning models are getting more and more popular to classify time series (Ismail Fawaz et al., 2019e; Wang, Yan, and Oates, 2017; Ma, Xiao, and Wang, 2018; Zheng et al., 2018; Ismail Fawaz et al., 2018b; Ismail Fawaz et al., 2018d). Furthermore, potential adversarial attacks are present in many applications where the use of time series

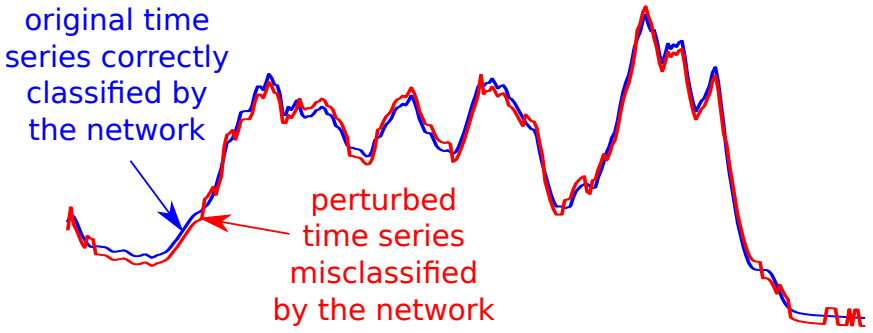


FIGURE 2.17: Example of a perturbed time series that is misclassified by a deep network after applying a small perturbation (time series from the Coffee dataset (Dau et al., 2019) containing spectrographs of coffee beans).

data is crucial. For example, Figure 2.17 shows an original and perturbed time series of coffee beans spectrograph. While a deep neural network correctly classifies the original time series as Robusta beans, adding small perturbations makes it classify it as Arabica. Therefore, since Arabica beans are more valuable than Robusta beans, this attack could be used to deceive food control tests and eventually the consumers.

In this section, we present, transfer and adapt adversarial attacks that have been shown to work well on images to time series data. We also present an experimental study using the 85 datasets of the UCR/UEA archive (Dau et al., 2019) which reveals that neural networks are prone to adversarial attacks. We highlight specific real-life use cases to stress the importance of such attacks in real-life settings, namely food quality and safety, vehicle sensors and electricity consumption. Our findings show that deep networks for time series data are vulnerable to adversarial attacks like their computer vision counterparts. Therefore, this work sheds the light on the need to protect against such attacks, especially when deep learning is used for sensitive TSC applications. We also show that adversarial time series learned using one network architecture can be transferred to different architectures. We then discuss some mechanisms to prevent these attacks while making the models more robust to adversarial examples. Finally, in the spirit of regularizing DNNs, we show how these perturbed time series can be leveraged in order to improve the generalization capability of a deep learning model: a technique called Adversarial Training (Xie et al., 2020).

2.5.1 Background

In this subsection, we start with the necessary definitions for ease of understanding. We then follow by an overview of critical applications based on deep learning approaches for TSC where adversarial attacks could have serious and dangerous consequences. Finally, we present a brief survey of the current state-of-the-art methods for adversarial attacks which have been mainly proposed and validated on image datasets (Yuan et al., 2017).

Definition 1. $f(\cdot) \in F : \mathbb{R}^T \rightarrow \hat{Y}$ represents a deep learning model for TSC.

Definition 2. $J_f(\cdot, \cdot)$ denotes the loss function (e.g. cross-entropy) of the model f.

Definition 3. $X^{'}$ denotes the adversarial example, a perturbed version of X (the original instance) such that $\hat{Y} \neq \hat{Y}^{'}$ and $\|X - X^{'}\|_p \leq \epsilon$.

In this section, we focus on the application of DNNs in crucial and sensitive decision making systems, thus motivating the investigation of neural network's vulnerabilities to adversarial examples. In Ma, Xiao, and Wang, 2018, CNNs were used to mine temporal electronic health data for risk prediction and disease sub-typing. In situations where algorithms are taking the decision for reimbursement of medical treatment, tampering medical records in an imperceptible manner could eventually lead to fraud. Apart from the health care industry, deep CNNs are also being used when monitoring power consumption from houses or factories. For example in Zheng et al., 2018, time series data from smart grids were analyzed for electricity theft detection, where in such use cases perturbed data can help thieves to avoid being detected. Other crucial decision making systems such as malware detection in smart-phones, leverage temporal data in order to classify if an Android application is malicious or not (Tobiyama et al., 2016). Using adversarial attacks, a hacker might generate synthetic data from his/her application allowing it to bypass the security systems and get it installed on the end user's smart-phone. Finally, when deep neural networks are deployed for road anomaly detection (Cabral et al., 2018), perturbing the data recorded by sensors placed on the road could help the entities responsible for such life threatening anomalies, to avoid being captured. We should note that this list of potential attacks is not exhaustive.

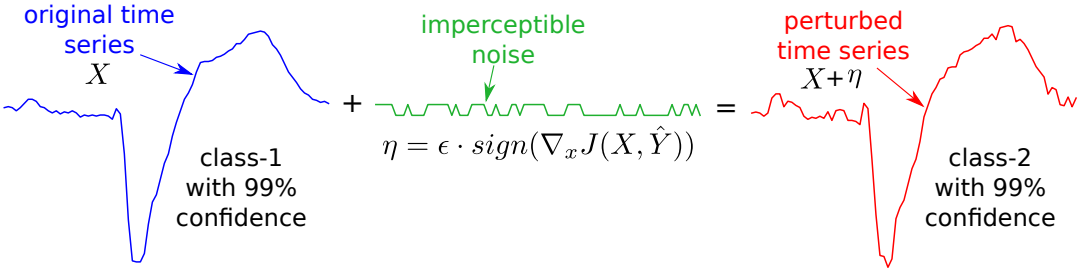


FIGURE 2.18: Example of perturbing the classification of an input time series from the TwoLeadECG (Dau et al., 2019) dataset by adding an imperceptible noise computed using the Fast Gradient Sign Method.

Adversarial attacks

Szegedy et al., 2014 were the first to introduce adversarial examples against deep neural networks for image recognition tasks in 2014. Following these intriguing findings, a huge amount of research has been dedicated to generating, understanding and preventing adversarial attacks on deep neural networks (Goodfellow, Shlens, and Szegedy, 2015; Kurakin, Goodfellow, and Bengio, 2017; Eykholt et al., 2018).

Most of these methods have been proposed for image recognition tasks (Yuan et al., 2017). For example, in Goodfellow, Shlens, and Szegedy, 2015, a fast gradient-based attack was developed as an alternative to expensive optimization techniques (Szegedy et al., 2014), where the authors explained the presence of such adversarial examples with the hypothesis of linearity for deep learning models. This kind of attack was also extended by a more costly iterative procedure (Kurakin, Goodfellow, and Bengio, 2017), where the authors showed for the first time that even printed adversarial images (*i.e.* perceived by a camera) are able to fool a pre-trained network. More recently, it has been shown that perturbing stop signs can

trick autonomous vehicles into misclassifying it as a speed limit sign (Eykholt et al., 2018).

Other fields such as Natural Language Processing have also been investigated to create adversarial attacks such as adding distracting phrases at the end of a paragraph in order to show that deep learning-based reading comprehension systems were not able to distinguish subtle differences in text (Jia and Liang, 2017). For a review on the different adversarial attacks for deep learning systems, we refer the interested readers to a recent survey in Yuan et al., 2017.

For general TSC tasks, it is surprising how adversarial attack approaches have been ignored by the community. The only previous work mentioning attacks for TSC is Oregi et al., 2018. By adapting a soft KNN coupled with DTW, the authors showed that adversarial examples could fool the proposed nearest neighbors classifier on a single simulated dataset (synthetic_control from the UCR/UEA archive Dau et al., 2019). However, the fact that the KNN classifier is no longer considered as the state-of-the-art classifier for time series data (Bagnall et al., 2017), we believe that it is important to investigate the generation of adversarial time series examples that deteriorate the accuracy of state-of-the-art classifiers such as ResNet (Ismail Fawaz et al., 2019d; Wang, Yan, and Oates, 2017) and to validate it on the whole 85 datasets in the UCR/UEA archive. Finally, we formally define an adversarial attack on deep neural networks for TSC.

Definition 4. Given a trained deep learning model f and an original input time series X, generating an adversarial instance $X^{'}$ can be described as a box-constrained optimization problem.

$$\min_{X^{\theta}} \|X^{'} - X\| \; s.t. \\ f(X^{'}) = \hat{Y}^{'}, \; f(X) = \hat{Y} \; and \; \hat{Y} \neq \hat{Y}^{'} \quad (2.2)$$

Let $\eta = X - X^{'}$ be the perturbation added to X, which corresponds to a very low amplitude signal. Figure 2.18 illustrates this process where the green time series corresponds to the added perturbation η. The optimization problem in (2.2) minimizes the perturbation while misclassifying the input time series.

2.5.2 Adversarial attacks for time series

In this subsection, we present two attack methods that we then use to generate adversarial time series examples for the ResNet model described in Chapter 1. For testing adversarial examples, we used ResNet, which was originally proposed for TSC in Wang, Yan, and Oates, 2017, where it was validated on 44 datasets from the UCR/UEA archive (Dau et al., 2019). In Chapter 1, we identified that ResNet achieved state-of-the-art performance for TSC, with results that are not significantly different than the HIVE-COTE, the current state-of-the-art classifier, which is an ensemble of 37 classifiers (Lines, Taylor, and Bagnall, 2018). Note that our adversarial attack methods are independent of the chosen network architecture, and that we chose ResNet for its robustness (Ismail Fawaz et al., 2019d) as well as its use in many critical domains such as malware detection (Cabral et al., 2018). In addition, adversarial examples are known to be transferable across different neural network architectures which enables the synthetic time series to fool other deep learning models: a technique known as black-box attack (Yuan et al., 2017). Finally, we describe a very recent method called AdvProp (Xie et al., 2020) that leverages adversarial training

in order to improve image recognition systems, which we implement and apply to our TSC problem.

Fast Gradient Sign Method

FGSM was first proposed in Goodfellow, Shlens, and Szegedy, 2015 to generate adversarial images that fooled the famous GoogLeNet model. FGSM is considered "fast" and replaces the expensive linear search method previously proposed in Szegedy et al., 2014. The attack is based on a one step gradient update along the direction of the gradient's sign at each time stamp. The perturbation process (illustrated in Figure 2.18) can be expressed as:

$$\eta = \epsilon \cdot sign(\nabla_x J(X, \hat{Y})) \tag{2.3}$$

where ϵ denotes the magnitude of the perturbation (a hyperparameter). The adversarial time series $X^{'}$ can be easily generated with $X^{'} = X + \eta$. The gradient can be efficiently computed using back-propagation.

Basic Iterative Method

BIM extends FGSM by applying it multiple times with a small step size and clip the obtained time series elements after each step to ensure that they are in an ϵ-neighborhood of the original time series (Kurakin, Goodfellow, and Bengio, 2017). In fact, by adding smaller changes or perturbations in an iterative manner, the method is able to generate adversarial examples that are closer to the original samples and have a better chance of fooling the network. Algorithm 2 shows the different steps of this iterative attack which requires setting three hyperparameters: (1) the number of iterations I; (2) the amount of maximum perturbation ϵ and (3) the per step small perturbation α. In our experiments we have set $\epsilon = 0.1$ heuristically similarly to Goodfellow, Shlens, and Szegedy, 2015; Kurakin, Goodfellow, and Bengio, 2017 and the rest of BIM hyperparameters were left at their default value in the Cleverhans API (Papernot et al., 2018).

Algorithm 2 Iterative Adversarial Attack

Parameter: I, ϵ, α
Input: original time series X & its label $\hat{Y}$ **Output:** perturbed time series $X^{'}$

$X^{'} \leftarrow X$
for $i = 1$ to I **do**
  $\eta = \alpha \cdot sign(\nabla_x J(X^{'}, \hat{Y}))$
  $X^{'} = X^{'} + \eta$
  $X^{'} = min\{X + \epsilon, max\{X - \epsilon, X^{'}\}\}$
end for

Adversarial Training

Adversarial training consists of generating adversarial examples during the training process of a neural network classifier. These adversarial instances are used as training examples in order to defend against adversarial attacks (Goodfellow, Shlens, and Szegedy, 2015). This technique currently constitutes the foundation of state-of-the-arts for defending against these attacks (Xie et al., 2019). Most previous approaches have witnessed a decrease in accuracy on clean original instances when the model

is trained in adversarial setting (Tsipras et al., 2018), showing an inevitable trade-off between a model's accuracy and its robustness to adversarial attacks. However, Xie et al., 2020 were the first to show that one can leverage adversarial training to improve the test time accuracy of a DNN. They managed to achieve this by adding a using two different batch normalization layers (Ioffe and Szegedy, 2015): one for the clean original intact training instances and a second one for the perturbed examples. Xie et al., 2020 argued that the main reason behind the deterioration in accuracy when using adversarial training was that the perturbed instances come from a distribution that is significantly different than the clean original ones. Thus having different normalization layers should allow the network to adapt its weights for each distribution. In our case, we have implemented AdvProp and tested its effect when training DNNs with adversarial training for TSC.

2.5.3 Results

Experimental setup

To train the deep neural network, we leveraged the parallel computation of a cluster of more than 60 GPUs (a mix of GTX 1080 Ti, Tesla K20, K40 and K80). All of our experiments were evaluated on the 85 datasets from the publicly available UCR/UEA archive (Dau et al., 2019). The model was trained/tested using the original training/testing splits provided in the archive. To perform the attacks, we have adapted the Cleverhans API (Papernot et al., 2018) by extending the well known attacks for time series data and perturbed only the test instances without using the test labels, similarly to the computer vision literature (Yuan et al., 2017).

For reproducibility and to allow the time series community to verify and build on our findings, the source code for generating adversarial time series is publicly available on our GitHub repository[5]. In addition, we provide on our companion web page[6] the raw results, our pre-trained models as well as a set of perturbed time series for each dataset in the UCR/UEA archive. This would allow time series data mining practitioners to test the robustness of their machine learning models against adversarial attacks.

Adversarial attacks on the whole UCR/UEA archive

For all datasets, both attacks managed to reduce ResNet's accuracy. One exception is the DiatomSizeReduction dataset which is the smallest one in the archive with an already low original accuracy equal to 30% due to overfitting (Ismail Fawaz et al., 2019d). Figure 2.19 shows the accuracy variation for both attacks with respect to the network's original accuracy on the UCR/UEA archive. On average, over the 85 datasets, FGSM and BIM managed to reduce the model's accuracy respectively by 43.2% and 56.89%. The Wilcoxon signed-rank test indicates that BIM is *significantly* better than FGSM in decreasing the model's accuracy, with a p-value $\leq 10^{-15}$. However, we should note that FGSM is a fast approach allowing real-time generation of adversarial time series whereas BIM is time-consuming and requires a certain number of iterations I.

By analyzing the use-cases where both attacks failed to fool the classifier, we found out that the corresponding datasets have two interesting characteristics that could explain the classifier's robustness to adversarial examples. The first one is

[5] https://github.com/hfawaz/ijcnn19attacks

[6] https://germain-forestier.info/src/ijcnn2019/

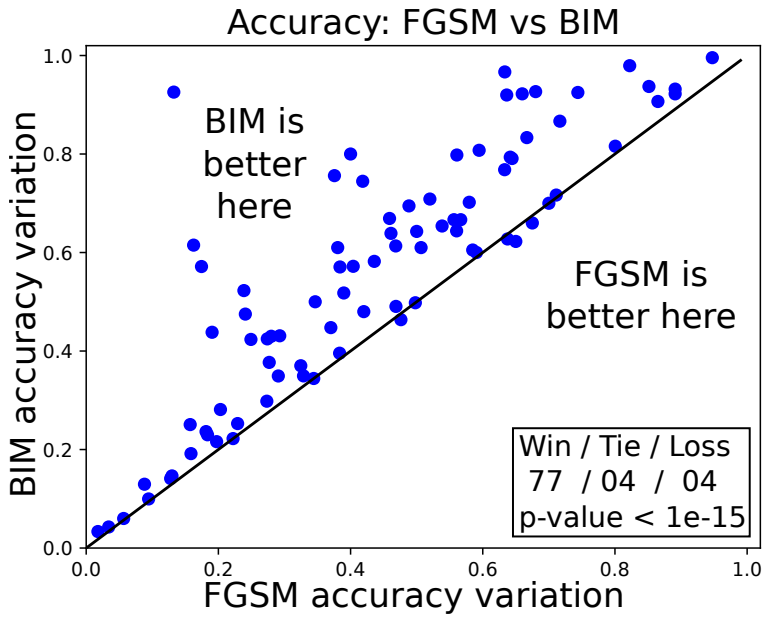


FIGURE 2.19: Accuracy variation for two attacks (FGSM and BIM) with respect to ResNet's original accuracy.

that 50% of the simulated datasets (CBF, Two_patterns and synthetic_control) in the archive are in the top six hardest datasets to attack. Perhaps since these are synthetic datasets generated by humans to serve some human intuition for TSC, small perturbations imperceptible by humans, are not enough to alter the classifier's decision. The second observation is that a network trained on datasets with time series of short length is harder to fool. This is rather expected since the less data points we have, the less amount of perturbation the attacker is allowed to add. For example ItalyPowerDemand contains the shortest sequences ($T = 24$) and is the second most hardest use-case for both attacks.

Multi-Dimensional Scaling

We used Multi-Dimensional Scaling (Kruskal and Wish, 1978; Forestier et al., 2017b) (explained in Chapter 1) with the objective to gain some insights on the spatial distribution of the perturbed time series compared to the original ones. Using the ED on a set of time series (original and perturbed), it is then possible to create a similarity matrix and apply MDS to display the set into a two dimensional space. The latter straightforward approach supposes that the ED is able to strongly separate the raw time series, which is usually not the case evident by the low accuracy of the nearest neighbor when coupled with the ED (Bagnall et al., 2017). Therefore, we decided to use the *linearly* separable representation of time series from the output of the GAP layer, which is used as input to the softmax *linear* classifier (multinomial logistic regression). More precisely, for each input time series, the last convolution outputs a multivariate time series whose dimensions are equal to the number of filters (128) in the last convolution, then the GAP layer averages the latter 128-dimensional multivariate time series over the time dimension resulting in a vector of 128 real values over which the ED is computed. This enables the MDS projection to be as close as possible to ResNet's latent representation of the time series. Obviously, one has to be careful about the interpretation of MDS output, as the data space is highly simplified (each time series X_i is represented as a single data point in 2D space).

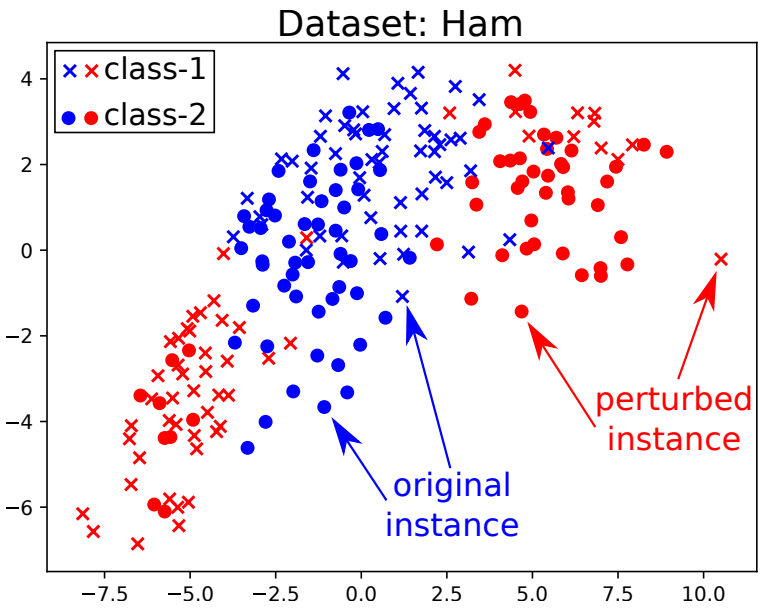


FIGURE 2.20: Multi-Dimensional Scaling showing the distribution of perturbed time series on the whole test set of the Ham dataset where the accuracy decreased from 80% to 21% after performing the BIM attack.

Attacks on food quality and safety

The determination of food quality, type and authenticity along with the detection of adulteration are major issues in the food industry (Nawrocka and Lamorska, 2013). With meat related product, authenticity checking concerns for example the identification of substitution of high value raw materials with cheaper materials like less costly cuts, offal, blood, water, eggs or other types of proteins. These substitutions not only decease the consumers but can also cause severe allergic responses as the substitute materials are hidden. Discriminating meat that has been frozen-and-thawed from fresh meat is also an important issue, as refreezing food can result in an increased amount of bacteria. Spectroscopic methods have been historically very successful at evaluating the quality of agricultural products, especially food (Nawrocka and Lamorska, 2013). This technique is routinely used as a quality assurance tool to determine the composition of food ingredients.

In this context, an adversarial attack could be used to modify recorded spectrographs (seen as time series) in order to hide the low qualify of the food. The Beef dataset (Al-Jowder, Kemsley, and Wilson, 2002) (from the UCR/UEA archive) contains four classes of beef spectrograms, from pure and adulterated beef with varying degrees of potential adulterants (heart, tripe, kidney, and liver). An adversarial attack could thus consist in modifying an adulterated beef to make a network classify it as pure beef. For this dataset, FGSM and BIM reduced the model's accuracy respectively by 56.7% and 66.7%.

The Ham dataset (Olias et al., 2006) contains measurements from 19 Spanish and 18 French dry-cured hams, with the goal to distinguish the provenance of the food. An adversarial attack could consist in perturbing the spectrograms to hide the real provenance of the food. Figure 2.20 shows the MDS projection of the original and perturbed instances for Ham's test set, where one can see that the adversarial examples are *pushed* toward the other class.

The Coffee dataset (Briandet, Kemsley, and Wilson, 1996) is a two class problem

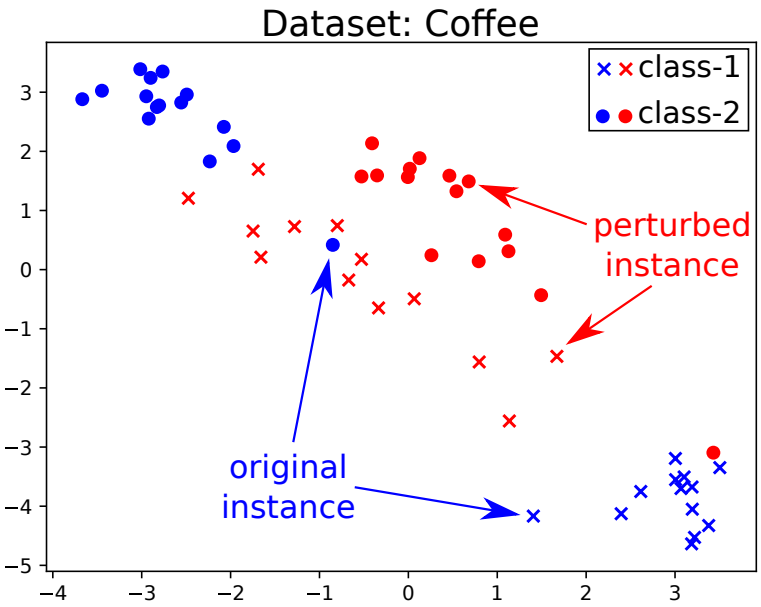


FIGURE 2.21: Multi-Dimensional Scaling showing the distribution of perturbed time series on the whole test set of the Coffee dataset where the accuracy decreased from 100% to 50% after performing the FGSM attack.

to distinguish between Robusta and Arabica coffee beans. Arabica beans are valued most highly by the trade, as they are considered to have a finer flavor than Robusta. An adversarial attack could consist in altering the spectrograms to make Robusta beans look like Arabica beans. Figure 2.21 shows the MDS representation of the original and perturbed time series from the test set. We can clearly see how the instances are *pushed* toward the class frontiers after performing the FGSM attack.

Attacks on vehicle sensors

The increase in the number of sensors and other electrical devices has drastically augmented the amount of data produced in the industry. These data are now routinely used to monitor systems or to perform predictive maintenance and prevent failures (Susto et al., 2015). The car industry is not an exception with the increasing number of sensors present in modern vehicles, especially for advanced driver assistance systems and autonomous driving.

Data are also used to perform diagnostic on vehicles in order to detect engine problems or compliance with environmental regulations. In this context, an adversarial attack could consist in altering sensor readings in order to hide a specific problem or to pass a CO_2 emission test. The famous "dieselgate" (or "emissionsgate") (Brand, 2016) made this kind of attack a reality as multiple automakers have been suspected of using emission control systems during laboratory emissions testing.

To illustrate this use case, we used the FordA datasets (from the UCR/UEA archive) that was was originally used in a competition in the IEEE World Congress on Computational Intelligence, 2008. The classification problem is to diagnose whether a certain symptom exists or not in an automotive subsystem. Each case consists of 500 measurements of engine noise and a class label. In this context, an attack could consist in hiding an engine problem. In practice for the FordA dataset,

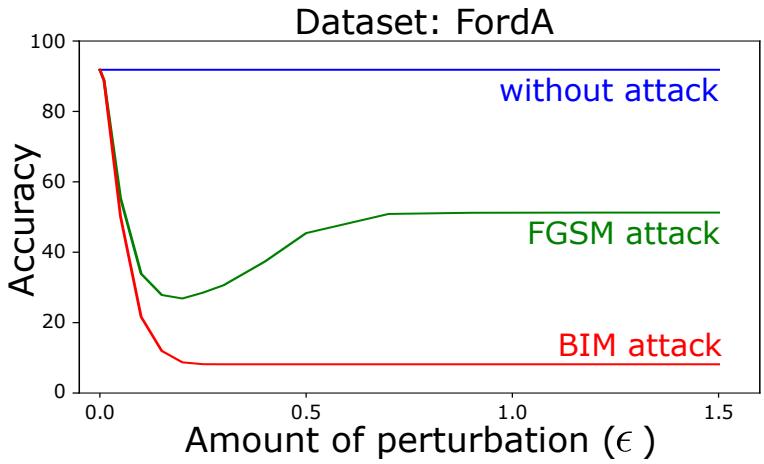


FIGURE 2.22: Accuracy variation with respect to the amount of perturbation for FGSM and BIM attacks on FordA.

the model's accuracy decreased by 57.9% and 70.2% when applying respectively the FGSM and BIM attacks.

Figure 2.22 depicts the variations of ResNet's accuracy on FordA with respect to the amount of perturbation ϵ allowed for the FGSM and BIM attacks. As expected (Kurakin, Goodfellow, and Bengio, 2017), we found that FGSM fails to generate adversarial examples that can fool the network for larger values of ϵ, whereas the BIM produces perturbed time series that can reduce a model's accuracy to almost 0.0%. This can be explained by the fact that BIM adds a small amount of perturbation α on each iteration whereas FGSM adds ϵ amount of noise for each data point in the series that may not be useful for misclassifying the test sample.

Attacks on electricity consumption

Smart meters are electronic devices that record electric power consumption while sending information to the electricity supplier for monitoring, billing and data analysis. These meters typically register energy hourly and report back at least once a day to the supplier by leveraging a two-way communication channel between the device and the supplier's central system. These smart meters have raised a set of concerns in public opinion especially because they send detailed information about how much electricity is being used for each time stamp. Precisely, it has been shown that it is possible to know exactly which type of electric device is or has been used from simply analyzing the electricity consumption data Owen and Foreman, 2012. In this context, an attack could consist in modifying the electricity consumption time series of one device to make it recognized as another in order to hide which devices are actually used by a specific user.

To illustrate this use case, we used the SmallKitchenAppliances dataset from the UCR/UEA archive that was recorded as part of government sponsored study called Powering the Nation (Owen and Foreman, 2012). By collecting and analyzing behavioral data about consumers' daily use of electricity within their homes, the goal is to reduce the UK's carbon footprint. The dataset contains readings from 251 households recorded over a month. Each univariate time series has a length equal to 720 corresponding to 24 hours of readings taken every two minutes. The three classes are: Kettle, Microwave and Toaster. For this dataset, FGSM and BIM managed to reduce the classifier's accuracy respectively by 38.4% and 57%.

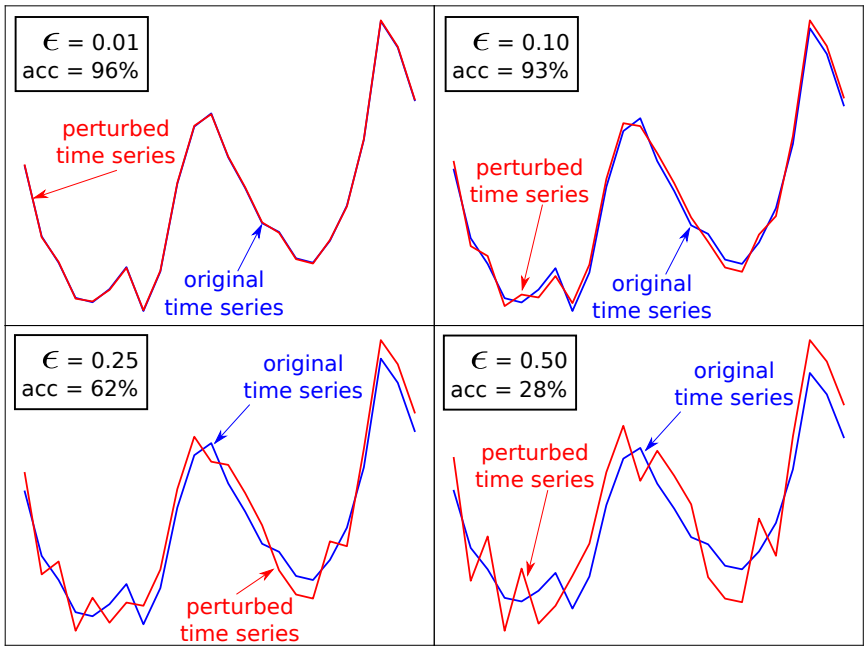


FIGURE 2.23: Accuracy variation for ItalyPowerDemand with respect to the perturbation ϵ where FGSM managed to fool the network with this example for $\epsilon \geq 0.3$.

The ItalyPowerDemand dataset (Keogh et al., 2006), another dataset from the UCR/UEA archive, contains twelve monthly electrical power demand time series from Italy. The task is to differentiate between instances that correspond to winter months (October to March) and summer months (April to September). This dataset contains the shortest time series in the archive ($T = 24$), thus requiring a higher amount of perturbation ϵ in order to be misclassified. Figure 2.23 shows the variation of the model's accuracy as well as the shape of a time series from the ItalyPowerDemand dataset with respect to the amount of noise that is added. For this example, both attacks needed higher values of perturbation ($\epsilon \geq 0.3$) rather than the default setting ($\epsilon = 0.1$).

Are adversarial examples transferable?

To evaluate the transferability of perturbed time series, we used the FCN which was originally proposed in Wang, Yan, and Oates, 2017 and was shown in Chapter 1 to be the second most accurate deep time series classifier when evaluated on the UCR/UEA archive (Dau et al., 2019). We used the test sets altered with FGSM and BIM using ResNet and try to classify it with FCN (both were originally trained on the same train set). For both FGSM and BIM attacks, FCN's accuracy decreases respectively by 38.2% and 42.8% which shows that adversarial examples are capable of generalizing to a different network architecture. The Wilcoxon signed-rank test also shows that BIM is *significantly* better than FGSM in reducing FCN's accuracy, with a p-value $\leq 10^{-7}$. This type of attacks is known as "black box" where the attackers do not have access to the target model's internal parameters (FCN) yet they are able to generate perturbed time series that fool the classifier.

How can we prevent such attacks?

Countermeasures for adversarial attacks (Yuan et al., 2017) follow two defense strategies: (1) *reactive*: identify the perturbed instance; (2) *proactive*: improve the network's robustness without generating adversarial examples. One of the most straightforward proactive methods is *adversarial training*, which consists of (re)training the classifier with adversarial examples. Other reactive techniques consist of detecting the adversarial examples during testing. However, most of these detectors are still prone to attacks that are designed specifically to fool the detectors (Yuan et al., 2017). Therefore, we think that the time series community would have much to offer in this area by leveraging the decades of research into non-probabilistic classifiers such as the nearest neighbor coupled with DTW (Tan et al., 2018). Running classifiers against the adversarial examples that we provide here, is a first step toward identifying vulnerable models and making them more robust to such type of attacks. Finally, a recent approach proposed by Abdu-Aguye et al., 2020 defended against adversarial attacks by framing the problem as an outlier detection task. By constructing a normalcy model based on information and chaos-theoretic measures, Abdu-Aguye et al., 2020 were able to determine whether unseen time series instances are normal or adversarial.

Can we leverage adversarial examples to improve generalization?

As we have previously mentioned, Xie et al., 2020 were among the first to show that adversarial training can be leveraged to improve the generalization capabilities of a neural network, by adopting two batch normalization layers: one for the original images and another for the perturbed ones. When adopting their approach for TSC, we found similar results: using AdvProp was necessary in order to avoid deteriorating the accuracy of our model. Figure 2.24 illustrates the pairwise comparison between applying adversarial training with or without the AdvProp technique. We can clearly see how vanilla adversarial training will deteriorate significantly the accuracy compared to using the AdvProp method. These results are in line with the original AdvProp paper (Xie et al., 2020). However, Figure 2.25 shows that using adversarial training does not improve significantly the accuracy for TSC, unlike the observations in Xie et al., 2020 on the ImageNet dataset. We hope that these preliminary results will give some insights to time series data mining researchers looking to leverage adversarial training as a regularization technique of DNNs for TSC.

2.5.4 Conclusion

In this section, we introduced the concept of adversarial attacks on deep learning models for time series classification. We defined and adapted two attacks, originally proposed for image recognition, for the TSC task. We showed how adversarial perturbations are able to reduce the accuracy for the state-of-the-art deep learning classifier (ResNet) when evaluated on the UCR/UEA archive benchmark. With deep neural networks becoming frequently adopted by time series data mining practitioners in real-life critical decision making systems, we shed the light on some crucial use cases where adversarial attacks could have serious and dangerous consequences. Finally we presented our initial preliminary results and showed that vanilla adversarial training will deteriorate the accuracy, which is why the AdvProp method was necessary to maintain a high accurate classifier.

In the future, we would like to investigate countermeasures techniques to defend machine learning models against such attacks while exploring the transferability of

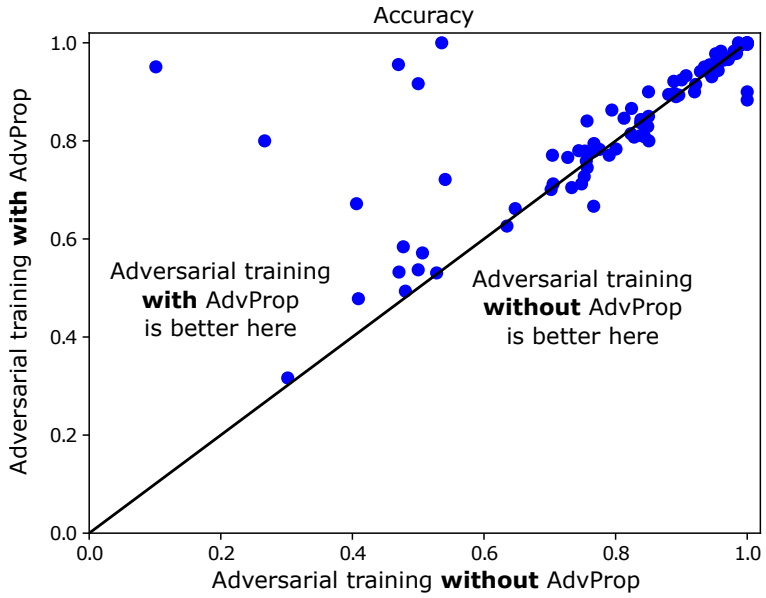


FIGURE 2.24: Accuracy of adversarial training with or without AdvProp.

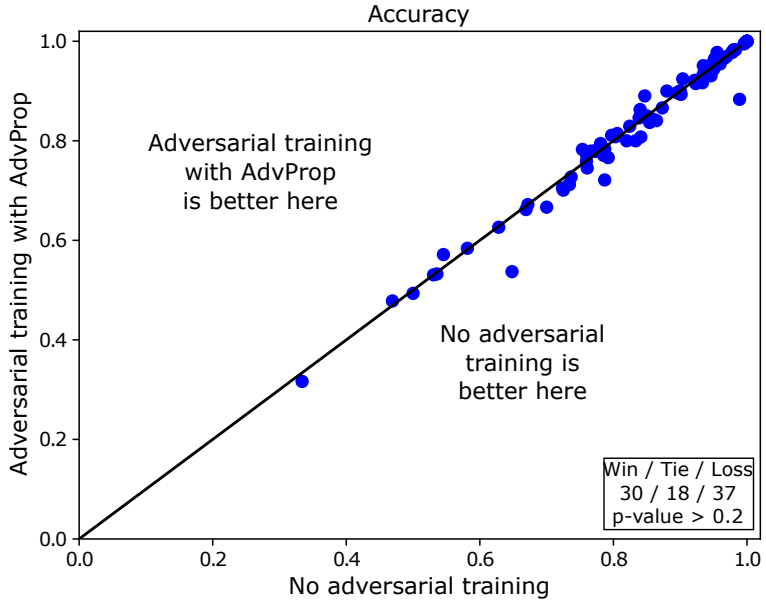


FIGURE 2.25: The effect of adversarial training on the model's accuracy.

adversarial examples to other non deep learning state-of-the-art classifiers. Finally, we would like to further explore the dozens of adversarial attacks that are published each year in order to identify and protect vulnerable deep learning models for TSC.

2.6 Conclusion

In this chapter, we have presented four main regularization techniques of deep learning models for TSC. First by implementing a transfer learning approach, coupled with an inter-dataset similarity selection algorithm, we were able to significantly improve the accuracy of FCN. We then proposed to leverage the high variance due to the stochastic nature of the optimization process of neural networks, by ensembling the decision of more than one network. Unlike transfer learning, we observed that ensembling will almost always improve the classifier's accuracy. We then presented a data augmentation technique based on the famous DTW algorithm, allowing us to increase the number of training samples, thus ameliorating the model's accuracy. Finally, we focused on the vulnerabilities of neural networks to adversarial attacks and gave many use case examples where such attacks might be catastrophic. Then in the spirit of regularizing DNNs, we showed how we can leverage these adversarial attacks to improve a network's generalization capabilities by leveraging a recent approach called AdvProp.

We believe that this chapter should motivate researchers into further looking into many types of regularization methods that were mostly applied to images. We should note that this list of techniques is not exhaustive, many other types of regularization still exist and constitute currently a very hot topic in machine learning such as self-supervised pre-training (Newell and Deng, 2020). Nevertheless, we believe that the current state-of-the-art neural networks architectures for TSC are suboptimal and still lack behind the current advances of deep learning in computer vision. Therefore, we present in the following chapter a novel architecture for TSC based on the famous Inception module proposed by Google for the ImageNet competition.

Chapter 3

InceptionTime: Finding AlexNet for Time Series Classification

3.1 Introduction

Recent times have seen an explosion in the magnitude and prevalence of time series data. Industries varying from health care (Forestier et al., 2018; Lee et al., 2018; Ismail Fawaz et al., 2019c) and social security (Yi et al., 2018) to human activity recognition (Yuan et al., 2018) and remote sensing (Pelletier, Webb, and Petitjean, 2019), all now produce time series datasets of previously unseen scale — both in terms of time series length and quantity. This growth also means an increased dependence on automatic classification of time series data, and ideally, algorithms with the ability to do this at scale.

In the previous chapters, we have shown how these TSC problems, differ significantly to traditional supervised learning for structured data, in that the algorithms should be able to handle and harness the temporal information present in the signal. It is easy to draw parallels from this scenario to computer vision problems such as image classification and object localization, where successful algorithms learn from the spatial information contained in an image. Put simply, the time series problem is essentially the same class of problem, just with one less dimension. Yet despite this similarity, the current state-of-the-art algorithms from the two fields share little resemblance (Ismail Fawaz et al., 2019d).

Deep learning has a long history (in machine learning terms) in computer vision (LeCun et al., 1998) but its popularity exploded with AlexNet (Krizhevsky, Sutskever, and Hinton, 2012), after which it has been unquestionably the most successful class of algorithms (LeCun, Bengio, and Hinton, 2015). Conversely, deep learning has only recently started to gain popularity amongst time series data mining researchers (see Chapter 1). This is emphasized by the fact that ResNet, which is currently considered the state-of-the-art neural network architecture for TSC when evaluated on the UCR/UEA archive (Dau et al., 2019), was originally proposed merely as a baseline model for the underlying task (Wang, Yan, and Oates, 2017). Given the similarities in the data, it is easy to suggest that there is much potential improvement for deep learning in TSC. In Chapter 2, we have shown how it is possible to improve the accuracy of a given deep learning architecture using various regularization techniques such as ensembling, transfer learning, data augmentation and adversarial training. However, we believe that there is still room for improvement in terms of network architecture, which can be considered an orthogonal task to the various DNNs regularization methods.

In this chapter, we take an important step towards finding the equivalent of 'AlexNet' for TSC by presenting InceptionTime — a novel deep learning ensemble for TSC. InceptionTime achieves state-of-the-art accuracy when evaluated on the

UCR/UEA archive (currently the largest publicly available repository for TSC (Dau et al., 2019)) while also possessing ability to scale to a magnitude far beyond that of its strongest competitor.

InceptionTime is an ensemble of five deep learning models for TSC, each one created by cascading multiple Inception modules (Szegedy et al., 2015). Each individual classifier (model) will have exactly the same architecture but with different randomly initialized weight values. The core idea of an Inception module is to apply multiple filters simultaneously to an input time series. The module includes filters of varying lengths, which as we will show, allows the network to automatically extract relevant features from both long and short time series. In fact, the ensemble here follows the same ensembling idea presented in Chapter 2.

After presenting InceptionTime and its results, we perform an analysis of the architectural hyperparameters of deep neural networks — depth, filter length, number of filters — and the characteristics of the Inception module — the bottleneck and residual connection, in order to provide insight into why this model is so successful. In fact, we construct networks with filters larger than have ever been explored for computer vision tasks, taking direct advantage of the fact that time series exhibit one less dimension than images.

3.2 Related work

In Chapter 1, we have shown that deeper CNN models coupled with residual connections such as ResNet can further improve the classification performance. In essence, since time series data exhibit only one structuring dimension (i.e. time, as opposed to two spatial dimensions for images), it is possible to explore more complex models that are usually computationally infeasible for image recognition problems: for example removing the pooling layers that throw away valuable information in favour of reducing the model's complexity. We therefore propose an Inception based network that applies several convolutions with various filters lengths. In contrast to networks designed for images, we are able to explore filters 10 times longer than recent Inception variants for image recognition tasks (Szegedy et al., 2017).

Inception was first proposed by Szegedy et al., 2015 for end-to-end image classification. Now the network has evolved to become Inceptionv4, where Inception was coupled with residual connections to further improve the performance (Szegedy et al., 2017). As for TSC a relatively competitive Inception-based approach was proposed in Karimi-Bidhendi, Munshi, and Munshi, 2018, where time series where transformed to images using Gramian Angular Difference Field, and finally fed to an Inception model that had been pre-trained for (standard) image recognition. Unlike this feature engineering approach, by adopting an end-to-end learning from raw time series data, a one-dimensional Inception model was used for Supernovae classification using the light flux of a region in space as an input MTS for the network (Brunel et al., 2019). However, the authors limited the conception of their Inception architecture to the one proposed by Google for ImageNet (Szegedy et al., 2017). In our work, we explore much larger filters than any previously proposed network for TSC in order to reach state-of-the-art performance on the UCR benchmark.

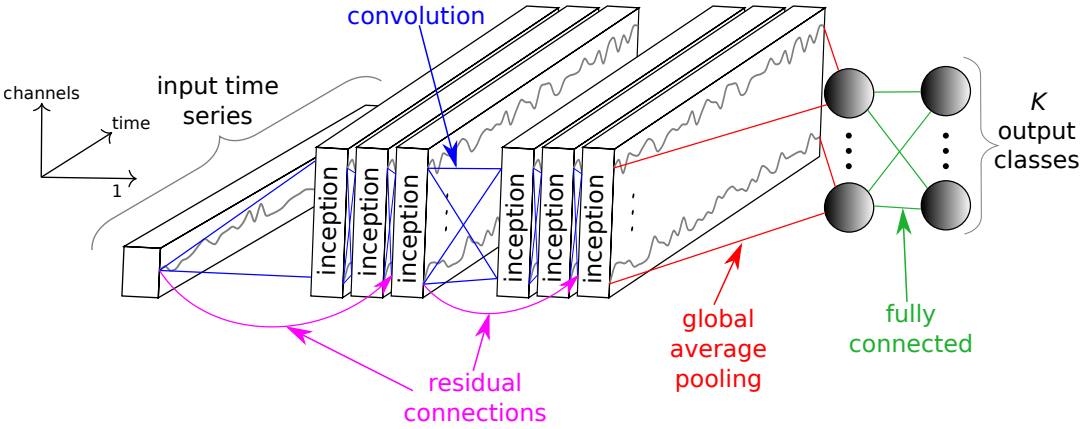


FIGURE 3.1: Our Inception network for time series classification.

3.3 InceptionTime: an accurate and scalable time series classifier

In this section, we start by describing the proposed architecture we call InceptionTime for classifying time series data. Specifically, we detail the main component of our network: the Inception module. We then present our proposed model InceptionTime which consists of an ensemble of 5 different Inception networks initialized randomly. Finally, we adapt the concept of Receptive Field for time series data.

3.3.1 Inception Network: a novel architecture for TSC

The composition of an Inception network classifier contains *two* different residual blocks, as opposed to ResNet, which is comprised of *three*. For the Inception network, each block is comprised of three Inception modules rather than traditional fully convolutional layers. Each residual block's input is transferred via a shortcut linear connection to be added to the next block's input, thus mitigating the vanishing gradient problem by allowing a direct flow of the gradient (He et al., 2016). Following these residual blocks, we employed a GAP layer that averages the output multivariate time series over the whole time dimension. At last, we used a final traditional fully-connected softmax layer with a number of neurons equal to the number of classes in the dataset. Figure 3.1 depicts an Inception network's architecture showing 6 different Inception modules stacked one after the other.

As for the Inception module, Figure 3.2 illustrates the inside details of this operation. Let us consider the input to be an MTS with M dimensions. The first major component of the Inception module is called the "bottleneck" layer. This layer performs an operation of sliding m filters of length 1 with a stride equal to 1. This will transform the time series from an MTS with M dimensions to an MTS with $m \ll M$ dimensions, thus reducing significantly the dimensionality of the time series as well as the model's complexity and mitigating overfitting problems for small datasets. Note that for visualization purposes, Figure 3.2 illustrates a bottleneck layer with $m = 1$. Finally, we should mention that this bottleneck technique allows the Inception network to have much longer filters than ResNet (almost ten times) with roughly the same number of parameters to be learned, since without the bottleneck layer, the filters will have M dimensions compared to $m \ll M$ when using the bottleneck layer. The second major component of the Inception module is sliding multiple filters of different lengths simultaneously on the same input time series. For example

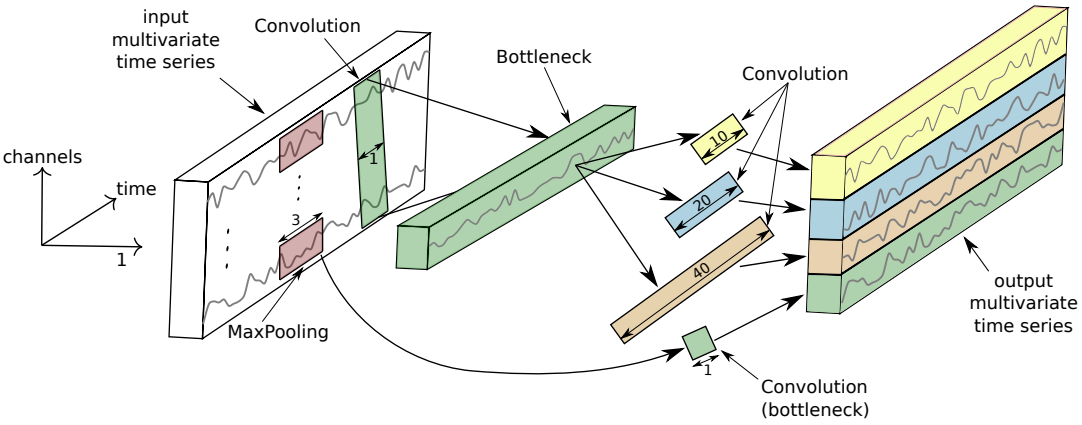


FIGURE 3.2: Inside our Inception module for time series classification. For simplicity we illustrate a bottleneck layer of size $m = 1$.

in Figure 3.2, three different convolutions with length $l \in \{10, 20, 40\}$ are applied to the input MTS, which is technically the output of the bottleneck layer. Additionally, in order to make our model invariant to small perturbations, we introduce another parallel MaxPooling operation, followed by a bottleneck layer to reduce the dimensionality. The output of sliding a MaxPooling window is computed by taking the maximum value in this given window of time series. Finally, the output of each independent parallel convolution/MaxPooling is concatenated to form the output MTS. The latter operations are repeated for each individual Inception module of the proposed network.

By stacking multiple Inception modules and training the weights (filters' values) via backpropagation, the network is able to extract latent hierarchical features of multiple resolutions thanks to the use of filters with various lengths. For completeness, we specify the exact number of filters for our proposed Inception module: 3 sets of filters each with 32 filters of length $l \in \{10, 20, 40\}$ with MaxPooling added to the mix, thus making the total number of filters per layer equal to $32 \times 4 = 128 = M$ - the dimensionality of the output MTS. The default bottleneck size value was set to $m = 32$.

3.3.2 InceptionTime: a neural network ensemble for TSC

Our proposed state-of-the-art InceptionTime model is an ensemble of 5 Inception networks, with each prediction given an even weight. In fact, during our experimentation, we have noticed that a single Inception network exhibits high standard deviation in accuracy, which is very similar to ResNet's behavior (Ismail Fawaz et al., 2019e). We believe that this variability comes from both the randomly initialized weights and the stochastic optimization process itself. This was an important finding for us, previously observed in Scardapane and Wang, 2017, as rather than training only one, potentially very good or very poor, instance of the Inception network, we decided to leverage this instability through ensembling, creating InceptionTime. The following equation explains the ensembling of predictions made by a network with different initializations:

$$\hat{y}_{i,c} = \frac{1}{n} \sum_{j=1}^{n} \sigma_c(x_i, \theta_j) \quad | \quad \forall c \in [1, C] \tag{3.1}$$

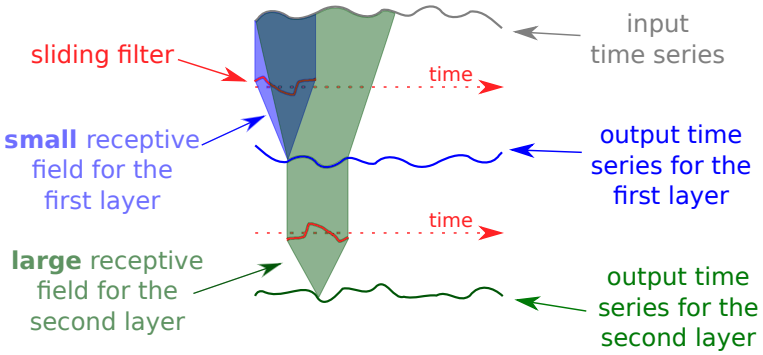


FIGURE 3.3: Receptive field illustration for a two layers CNN.

with $\hat{y}_{i,c}$ denoting the ensemble's output probability of having the input time series x_i belonging to class c, which is equal to the logistic output σ_c averaged over the n randomly initialized models. More details on ensembling neural networks for TSC can be found in Chapter 2. As for our proposed model, we chose the number of individual classifiers to be equal to 5, which is justified in the next section. We should note that we have opted to a neural network ensemble given the small training size of the UCR/UEA archive datasets which are not well suited to deep learning approaches, thus allowing us to control and leverage the variance of the error, which is likely to reduce when increasing the training set's size.

3.3.3 Receptive field

The concept of Receptive Field is an essential tool to the understanding of deep CNNs (Luo et al., 2016). Unlike FC networks or MLPs, a neuron in a CNN depends only on a region of the input signal. This region in the input space is called the receptive field of that particular neuron. For computer vision problems this concept was extensively studied, such as in Liu, Yu, and Han, 2018 where the authors compared the effective and theoretical receptive fields of a CNN for image segmentation.

For temporal data, the receptive field can be considered as a theoretical value that measures the maximum field of view of a neural network in a one-dimensional space: the larger it is, the better the network becomes (in theory) in detecting longer patterns. We now provide the definition of the RF for time series data, which is later used in our experiments. Suppose that we are sliding convolutions with a stride equal to 1. The formula to compute the RF for a network of depth d with each layer having a filter length equal to k_i with $i \in [1, d]$ is:

$$1 + \sum_{i=1}^{d} (k_i - 1) \tag{3.2}$$

By analyzing equation 3.2 we can clearly see that adding two layers to the initial set of d layers, will increase only slightly the value of RF. In fact in this case, if the old RF value is equal to $RF^{'}$, the new value RF will be equal to $RF^{'} + 2 \times (k - 1)$. Conversely, by increasing the filter length k_i, $\forall i \in [1, d]$ by 2, the new value RF will be equal to $RF^{'} + 2 \times d$. This is rather expected since by increasing the filter length for all layers, we are actually increasing the RF for each layer in the network. Figure 3.3 illustrates the RF for a two layers CNN.

In this chapter, we chose to focus on the RF concept since it has been known for computer vision problems, that larger RFs are required to capture more context for

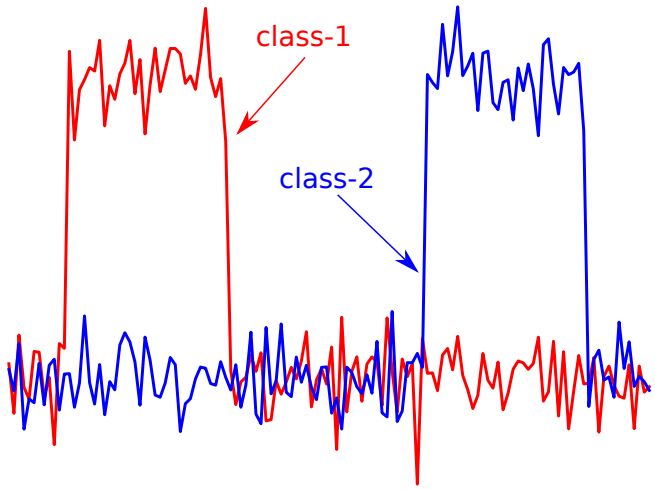


FIGURE 3.4: Example of a synthetic binary time series classification problem.

object recognition (Luo et al., 2016). Following the same line of thinking, we hypothesize that detecting larger patterns from very long one-dimensional time series data, requires larger receptive fields.

3.4 Experimental setup

First, we detail the method to generate our synthetic dataset, which is later used in our architecture and hyperparameter study. For testing our different deep learning methods, we created our own synthetic TSC dataset. The goal was to be able to control the length of the time series data as well as the number of classes and their distribution in time. To this end, we start by generating a univariate time series using uniformly distributed noise sampled between 0.0 and 0.1. Then in order to assign this synthetic random time series to a certain class, we inject a pattern with an amplitude equal to 1.0 in a pre-defined region of the time series. This region will be specific to a certain class, therefore by changing the placement of this pattern we can generate an unlimited amount of classes, whereas the random noise will allow us to generate an unlimited amount of time series instances per class. One final note is that we have fixed the length of the pattern to be equal to 10% the length of the synthetic time series. An example of a synthetic binary TSC problem is depicted in Figure 3.4.

All DNNs were trained by leveraging the parallel computation of a remote cluster of more than 60 GPUs comprised of GTX 1080 Ti, Tesla K20, K40 and K80. Local testing and development was performed on an NVIDIA Quadro P6000. The latter graphics card was also used for computing the training time of a model. When evaluating global accuracy and computational complexity, we have used the UCR/UEA archive (Dau et al., 2019), which is the largest publicly available archive for TSC. The models were trained/tested using the original training/testing splits provided in the archive. To study the effect of different hyperparameters and architectural designs, we used in addition to the traditional UCR benchmark for TSC, the synthetic dataset whose generation is described in details in the previous paragraph. All time series data were z-normalized (including the synthetic series) to have a mean equal to zero and a standard deviation equal to one. This is considered a common best-practice before classifying time series data (Bagnall et al., 2017). Finally, we should

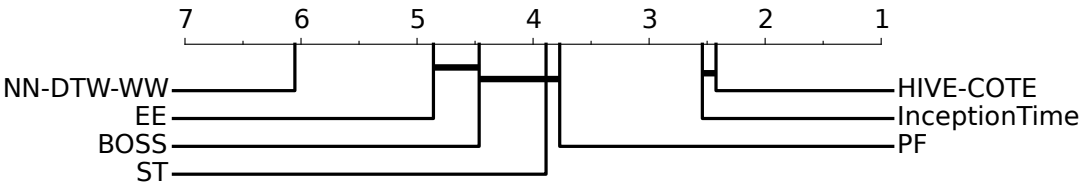


FIGURE 3.5: Critical difference diagram showing the performance of InceptionTime compared to the current state-of-the-art classifiers of time series data.

note that all models are trained using the Adam optimization algorithm (Kingma and Ba, 2015) and all weights are initialized randomly using Glorot's uniform technique (Glorot and Bengio, 2010).

When comparing with the state-of-the-art results published in Bagnall et al., 2017 we used the deep learning model's median test accuracy over the different runs, similarly to what we have done in Chapter 1. Following the recommendations in Demšar, 2006 we adopted the Friedman test (Friedman, 1940) in order to reject the null hypothesis. We then performed the pairwise post-hoc analysis recommended by Benavoli, Corani, and Mangili, 2016 where we replaced the average rank comparison by a Wilcoxon signed-rank test with Holm's alpha (5%) correction (Garcia and Herrera, 2008). To visualize this type of comparison we used a critical difference diagram proposed by Demšar, 2006, where a thick horizontal line shows a cluster of classifiers (a clique) that are not-significantly different in terms of accuracy.

In order to allow for the time series community to build upon and verify our findings, the source code for all these experiments was made publicly available on our companion repository[1]. In addition, we are planning on providing the pre-trained deep learning models, thus allowing data mining practitioners to leverage these networks in a transfer learning setting (Ismail Fawaz et al., 2018d).

3.5 Experiments: InceptionTime

In this section, we present the results of our proposed novel classifier called InceptionTime, evaluated on the 85 datasets of the UCR/UEA archive. We note that throughout this chapter (unless specified otherwise) InceptionTime refers to an ensemble of 5 Inception networks, while the "InceptionTime(n)" notation is used to denote an ensemble of n Inception networks.

Figure 3.5 illustrates the critical difference diagram with InceptionTime added to the mix of the current state-of-the-art classifiers for time series data, whose results were taken from Bagnall et al., 2017. We can see here that our InceptionTime ensemble reaches competitive accuracy with the class-leading algorithm HIVE-COTE, an ensemble of 37 TSC algorithms with a hierarchical voting scheme (Lines, Taylor, and Bagnall, 2016). While the two algorithms share the same clique on the critical difference diagram, the trivial GPU parallelization of deep learning models makes learning our InceptionTime model a substantially easier task than training the 37 different classifiers of HIVE-COTE, whose implementation does not trivially leverage the GPUs' computational power.

To further visualize the difference between the InceptionTime and HIVE-COTE, Figure 3.6 depicts the accuracy plot of InceptionTime against HIVE-COTE for each of the 85 UCR datasets. The results show a Win/Tie/Loss of 40/6/39 in favor of

[1] https://github.com/hfawaz/InceptionTime

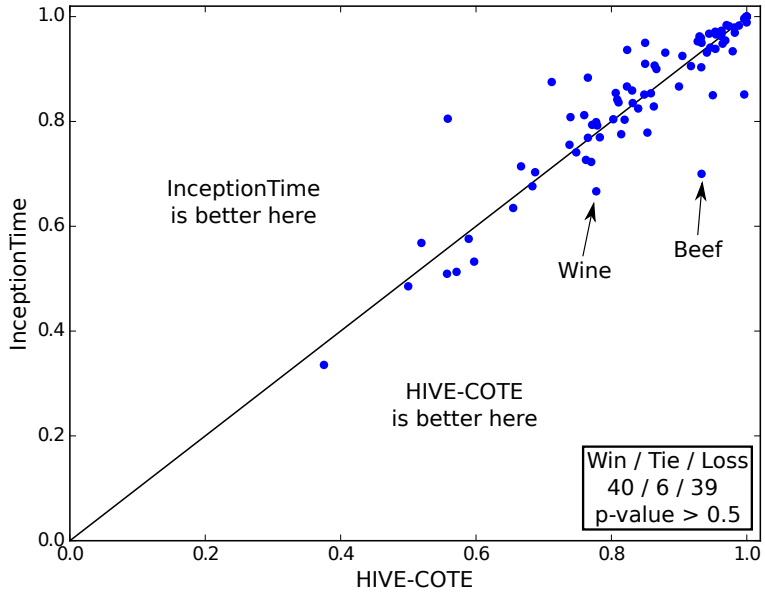


FIGURE 3.6: Accuracy plot showing how our proposed InceptionTime model is not significantly different than HIVE-COTE.

InceptionTime, however the difference is not statistically significant as previously discussed. From Figure 3.6, we can also easily spot the two datasets for which InceptionTime noticeably under-performs (in terms of accuracy) with respect to HIVE-COTE: Wine and Beef. These two datasets contain spectrography data from different types of beef/wine, with the goal being to determine the correct type of meat/wine using the recorded time series data. In Chapter 2, we showed that transfer learning significantly increases the accuracy for these two datasets, especially when fine-tuning a dataset with similar time series data. Our results suggest that further potential improvements may be available for InceptionTime when applying a transfer learning approach, as recent discoveries in Kashiparekh et al., 2019 show that the various filter lengths of the Inception modules have been shown to benefit more from fine-tuning than networks with a static filter length.

Now that we have demonstrated that our proposed technique is able to reach the current state-of-the-art accuracy for TSC problems, we will further investigate the time complexity of our model. Note that during the following experiments, we ran our ensemble on a single Nvidia Quadro P6000 in a sequential manner, meaning that for InceptionTime, 5 different Inception networks were trained one after the other. Therefore we did not make use of our remote cluster of GPUs. First we start by investigating how our algorithm scales with respect to the length of the input time series. Figure 3.7 shows the training time versus the length of the input time series. For this experiment, we used the InlineSkate dataset with an exponential re-sampling. We can clearly see that InceptionTime's complexity increases almost linearly with an increase in the time series' length, unlike HIVE-COTE, whose execution is almost two order of magnitudes slower. Having showed that InceptionTime is significantly faster when dealing with long time series, we now proceed to evaluating the training time with respect to a number of time series in a dataset. To this end, we used a Satellite Image Time Series dataset (Tan, Webb, and Petitjean, 2017). The data contain approximately one million time series, each of length 46 and labeled as one of 24 possible land-use classes (e.g. 'wheat', 'corn', 'plantation', 'urban'). From Figure 3.8 we can easily see how our InceptionTime is an order of magnitude faster than HIVE-COTE, and the trend suggests that this difference will only

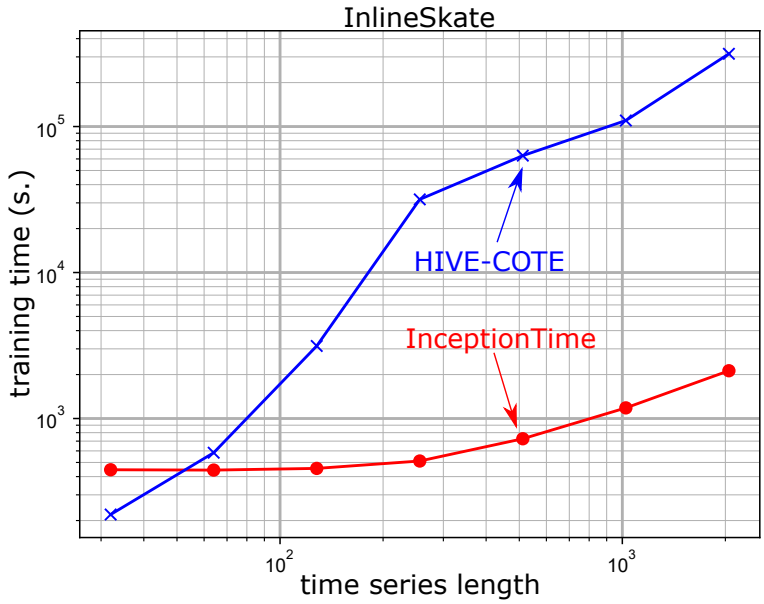


FIGURE 3.7: Training time as a function of the series length for the InlineSkate dataset.

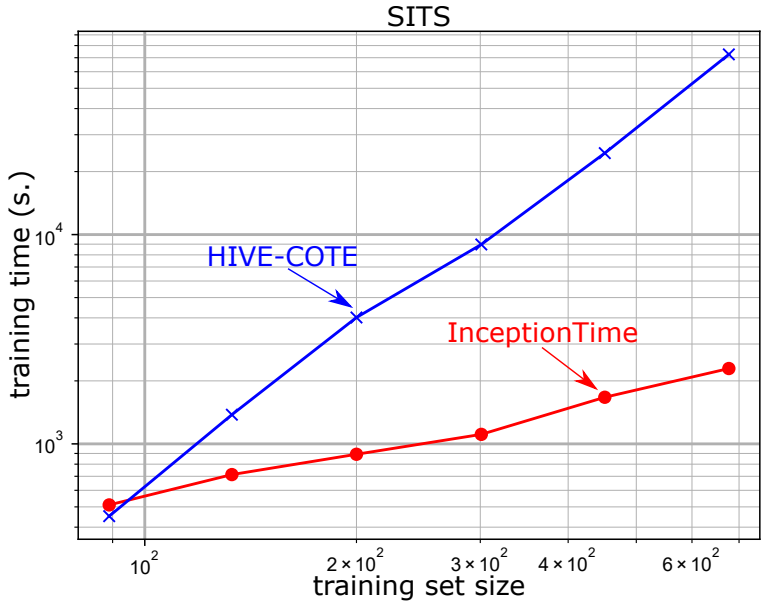


FIGURE 3.8: Training time as a function of the training set size for the SITS dataset.

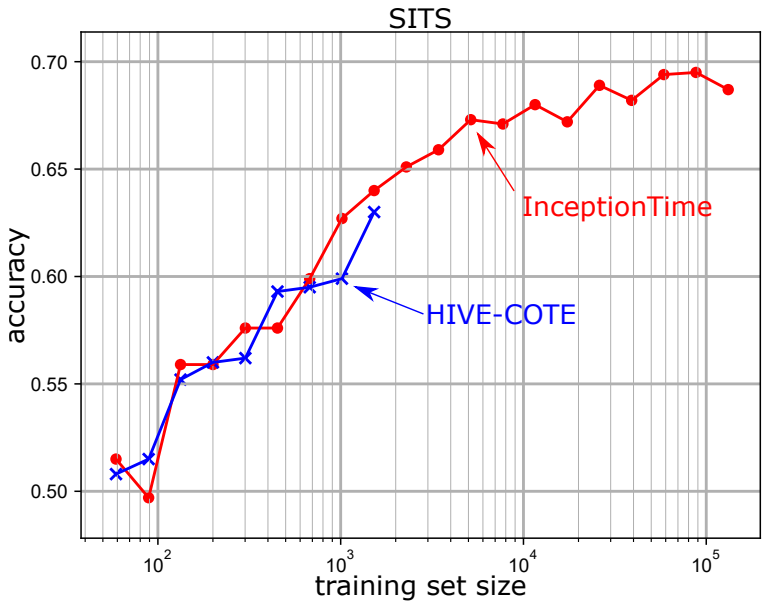


FIGURE 3.9: Accuracy as a function of the training set size for the SITS dataset.

continue to grow, rendering InceptionTime a clear favorite classifier in the Big Data era. Note that HIVE-COTE uses heuristics in its implementation, which explains why the complexity appears lower in the experiments than the expected $O(T^4)$. To summarize, we believe that InceptionTime should be considered as one of the top state-of-the-art methods for TSC, given that it demonstrates equal accuracy to that of HIVE-COTE (see Figure 3.6) while being much faster (see Figure 3.7 and 3.8).

In order to further demonstrate the capability of InceptionTime to handle efficiently a large amount of training samples unlike its counterpart HIVE-COTE, we show in Figure 3.9 how the accuracy continues to increase with InceptionTime for larger training set sizes, where HIVE-COTE would take 100 times longer to run.

The pairwise accuracy plot in Figure 3.10 compares InceptionTime to a model we call ResNet(5), which is an ensemble of 5 different ResNet networks (Ismail Fawaz et al., 2019e). We found that InceptionTime showed a significant improvement over its neural network competitor, the previous best deep learning ensemble for TSC. Specifically, our results show a Win/Tie/Loss of 54/8/23 in favor of InceptionTime against ResNet(5) with a p-value < 0.01, suggesting the significant gain in performance is mainly due to improvements in our proposed Inception network architecture. Additionally, in order to have a fair comparison between ResNet(5) and InceptionTime, we fixed the batch size of ResNet to 64 – equal to the default value used for InceptionTime. This would further highlight that the improvement is mainly due to the architectural design of our proposed network, and not due to some other optimization hyperparameter such as the batch size. Finally, we would like to note that when using the original batch size value proposed by Wang, Yan, and Oates, 2017 for ResNet, we observed similar results: InceptionTime was significantly better than the original ResNet(5) with a Win/Tie/Loss of 53/7/25.

In order to better understand the effect of the randomness on the accuracy of our neural networks, we present in Figure 3.11 the critical difference diagram of different InceptionTime(x) ensembles with $x \in \{1, 2, 5, 10, 20, 30\}$ denoting the number of individual networks in the ensemble. Note that InceptionTime(1) is equivalent to a single Inception network and InceptionTime is equivalent to InceptionTime(5).

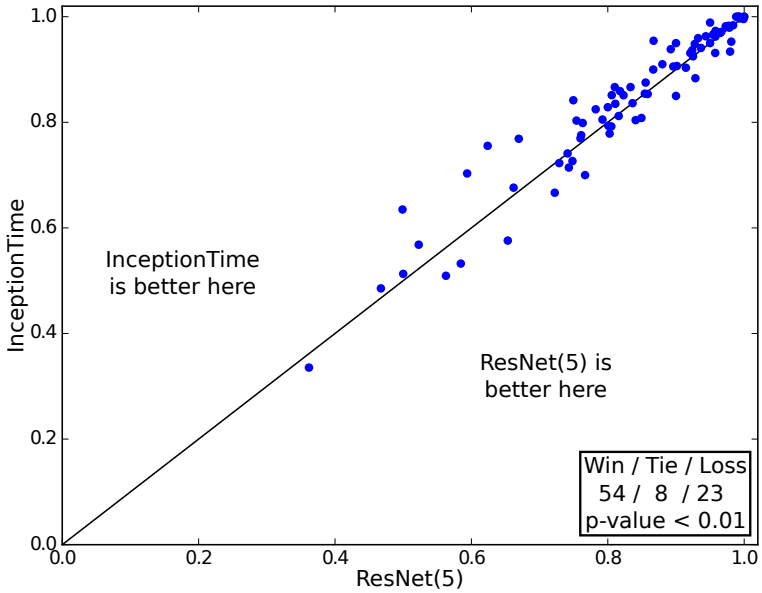


FIGURE 3.10: Plot showing how InceptionTime significantly outperforms ResNet(5).

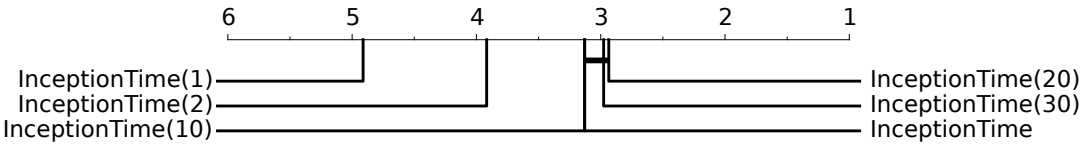


FIGURE 3.11: Critical difference diagram showing the effect of the number of individual classifiers in the InceptionTime ensemble.

By observing Figure 3.11 we notice how there is no significant improvement when $x \geq 5$, which is why we chose to use an ensemble of size 5, to minimize the classifiers' training time.

3.6 Architectural Hyperparameter study

In this section, we will further investigate the hyperparameters of our deep learning architecture and the characteristics of the Inception module in order to provide insight for practitioners looking at optimizing neural networks for TSC. First, we start by investigating the batch size hyperparameter, since this will greatly influence training time of all of our models. Then we investigate the effectiveness of residual and bottleneck connections, both of which are present in InceptionTime. After this we will experiment on model depth, filter length, and number of filters. In all experiments the default values for InceptionTime are: batch size 64; bottleneck size 32; depth 6; filter length {10,20,40}; and, number of filters 32. Finally, since the train/test split (provided in the archive) does not help in estimating the generalization ability of our approach, we have conducted a sensitivity analysis that evaluates the second best value for each of the network's hyperparameters.

3.6.1 Batch size

We started by investigating the batch size hyperparameter on the UCR/UEA archive, since this will greatly influence training time of our models. The critical

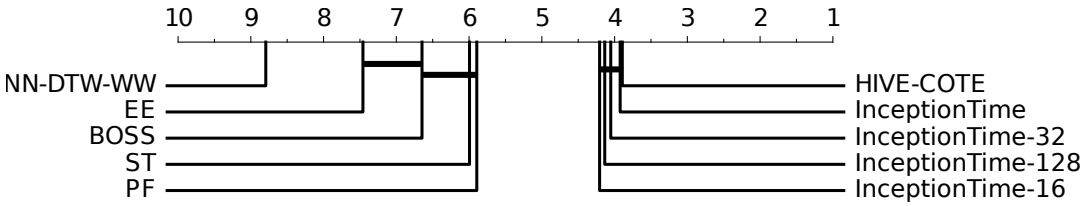


FIGURE 3.12: Critical difference diagram showing the effect of the batch size hyperparameter value over InceptionTime's average rank.

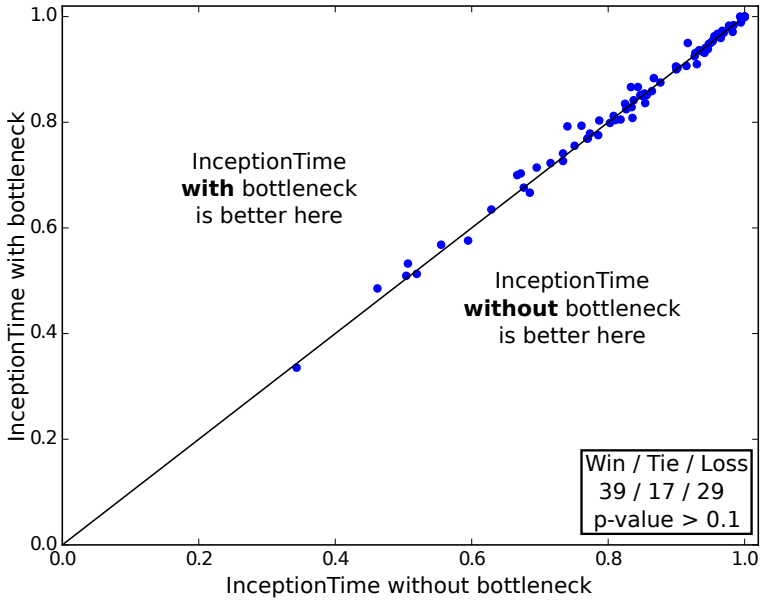


FIGURE 3.13: Accuracy plot for InceptionTime with/without the bottleneck layer.

difference diagram in Figure 3.12 shows how the batch size affects the performance of InceptionTime. The horizontal thick line between the different models shows a non significant difference between them when evaluated on the 85 datasets, with a small superiority to InceptionTime (batch size equal to 64). Finally, we should note that as we did not observe any significant impact on accuracy we did not study the effect of this hyperparameter on the simulated dataset and we chose to fix the batch size to 64 (similarly to InceptionTime) when experimenting on the simulated dataset below.

3.6.2 Bottleneck and residual connections

In Chapter 1, compared to other deep learning classifiers, ResNet achieved the best classification accuracy when evaluated on the 85 datasets and as a result we chose to look at the specific characteristic of this architecture — its residual connections. Additionally, we tested one of the defining characteristics of Inception — the bottleneck feature. For the simulated dataset, we did not observe any significant impact of these two connections, we therefore proceed with experimenting on the 85 datasets from the UCR/UEA archive.

Figure 3.13 shows the pairwise accuracy plot comparing InceptionTime with/without the bottleneck. Similar to the experiments on the simulated dataset, we did not find any significant variation in accuracy when adding or removing the bottleneck layer.

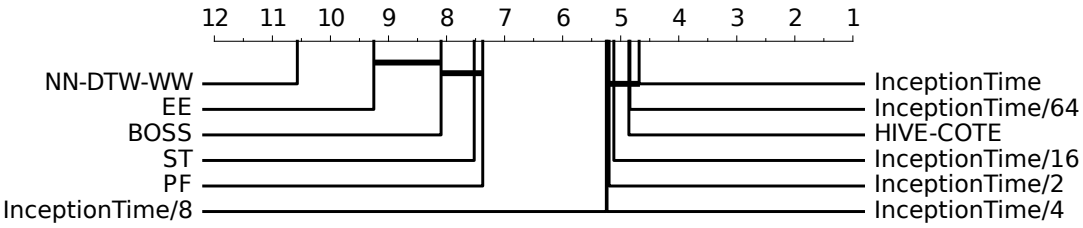


FIGURE 3.14: Critical difference diagram showing how the network's bottleneck size affects InceptionTime' average rank.

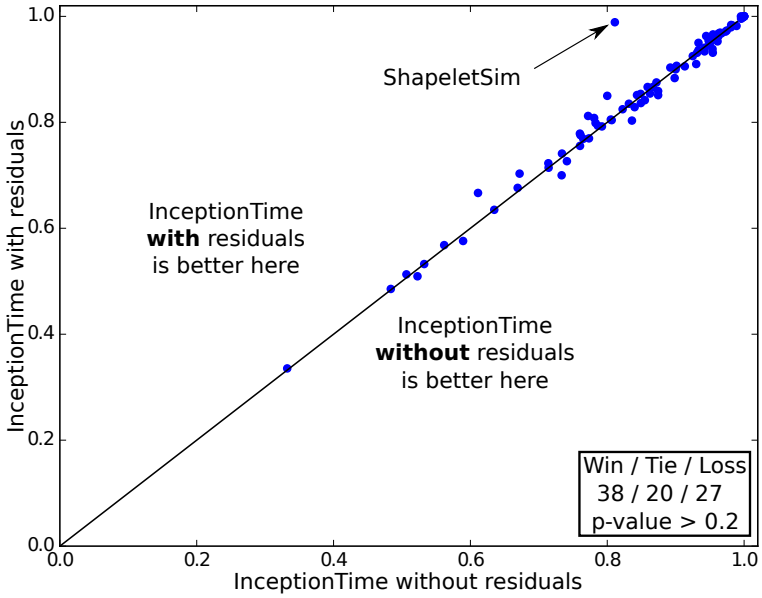


FIGURE 3.15: Accuracy plot for InceptionTime with/without the residual connections.

In fact, using a Wilcoxon Signed-Rank test we found that InceptionTime with the bottleneck layer is only slightly better than removing the bottleneck layer (p-value > 0.1). In terms of accuracy, these results all suggest not to use a bottleneck layer, however we should note that the major benefit of this layer is to significantly decrease the number of parameters in the network. In this case, InceptionTime with the bottleneck contains almost half the number of parameters to be learned, and given that it does not significantly decrease accuracy, we chose to retain its usage. In a more general sense, these experiments suggest that choosing whether or not to use a bottleneck layer is actually a matter of finding a balance between a model's accuracy and its complexity. The latter observation is evident in Figure 3.14 where choosing smaller bottleneck size in order to reduce InceptionTime's runtime will result in small yet insignificant decrease in accuracy.

To test the residual connections, we simply removed the residual connection from InceptionTime. Thus, without any shortcut connection, InceptionTime will simply become a deep convolutional neural network with stacked Inception modules. Figure 3.15 shows how the residual connections have a minimal effect on accuracy when evaluated over the whole 85 datasets in the UCR/UEA archive with a p-value > 0.2.

This result was unsurprising given that for computer vision tasks residual connections are known to improve the convergence rate of the network but not alter its test accuracy (Szegedy et al., 2017). However, for some datasets in the archive, the

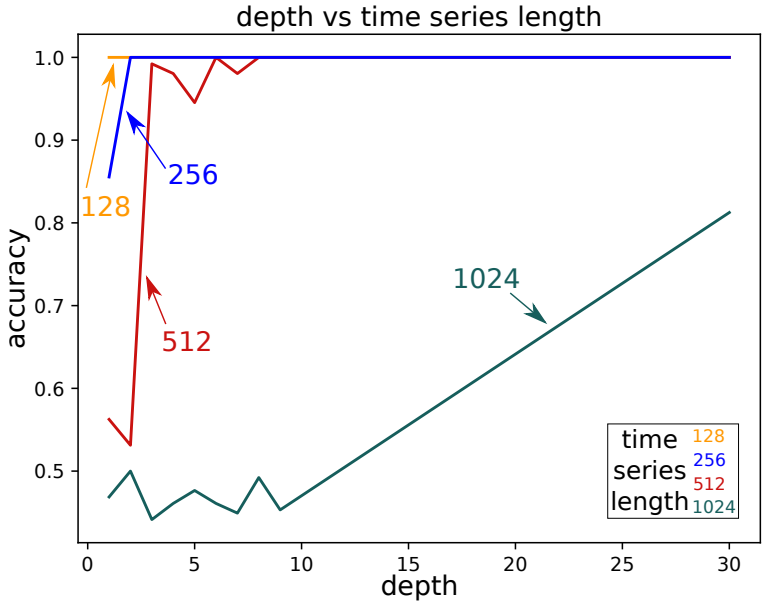


FIGURE 3.16: Inception network's accuracy over the simulated dataset, with respect to the network's depth as well as the length of the input time series.

residual connections did not show any improvement nor deterioration of the network's convergence either. This could be linked to other factors that are specific to these data, such as the complexity of the dataset.

One example of interest that we noticed was a significant decrease in InceptionTime's accuracy when removing the residual component for the ShapeletSim dataset. This is a synthetic dataset, designed specifically for shapelets discovery algorithms, with shapelets (discriminative subsequences) of different lengths (Hills et al., 2014). Further investigations on this dataset indicated that InceptionTime without the residual connections suffered from a severe overfitting.

While not the case here, some research has observed benefits of skip, dense or residual connections (Huang et al., 2017). Given this, and the small amount of labeled data available in TSC compared to computer vision problems, we believe that each case should be independently study whether to include residual connections. The latter observation suggests that a large scale general purpose labeled dataset similar to ImageNet (Russakovsky et al., 2015) is needed for TSC. Finally, we should note that the residual connection has a minimal impact on the network's complexity (Szegedy et al., 2017).

3.6.3 Depth

Most of deep learning's success in image recognition tasks has been attributed to how 'deep' the architectures are (LeCun, Bengio, and Hinton, 2015). Consequently, we decided to further investigate how the number of layers affects a network's accuracy. Unlike the previous hyperparameters, we present here the results on the simulated dataset. Apart from the depth parameter, we used the default values of InceptionTime. For this dataset we fixed the number of training instances to 256 and the number of classes to 2 (see Figure 3.4 for an example). The only dataset parameter we varied was the length of the input time series.

Figure 3.16 illustrates how the model's accuracy varies with respect to the network's depth when classifying datasets of time series with different lengths. Our

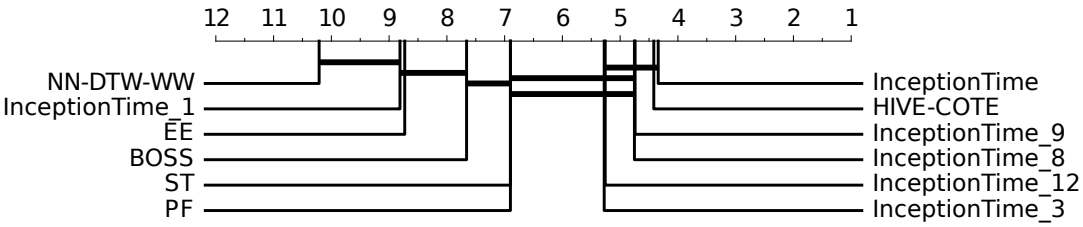


FIGURE 3.17: Critical difference diagram showing how the network's depth affects InceptionTime' average rank.

initial hypothesis was that as longer time series can potentially contain longer patterns and thus should require longer receptive fields in order for the network to separate the classes in the dataset. In terms of depth, this means that longer input time series will garner better results with deeper networks. And indeed, when observing Figure 3.16, one can easily spot this trend: deeper networks deliver better results for longer time series.

In order to further see how much effect the depth of a model has on real TSC datasets, we decided to implement deeper and shallower InceptionTime models, by varying the depth between 1 layer and 12 layers. In fact, compared with the original architecture proposed by Wang, Yan, and Oates, 2017, the deeper (shallower) version of InceptionTime will contain one additional (fewer) residual blocks each one comprised of three inception modules. By adding these layers, the deeper (shallower) InceptionTime model will contain roughly double (half) the number of parameters to be learned. Figure 3.17 depicts the critical difference diagram comparing the deeper and shallower InceptionTime models to the original InceptionTime.

Unlike the experiments on the simulated dataset, we did not manage to improve the network's performance by simply increasing its depth. This may be due to many reasons, however it is likely due to the fact that deeper networks need more data to achieve high generalization capabilities (LeCun, Bengio, and Hinton, 2015), and since the UCR/UEA archive does not contain datasets with a huge number of training instances, the deeper version of InceptionTime was overfitting the majority of the datasets and exhibited a small insignificant decrease in performance. On the other hand, the shallower version of InceptionTime suffered from a significant decrease in accuracy (see InceptionTime_3 and InceptionTime_1 in Figure 3.17). This suggests that a shallower architecture will contain a significantly smaller RF, thus achieving lower accuracy on the overall UCR/UEA archive.

From these experiments we can conclude that increasing the RF by adding more layers will not necessarily result in an improvement of the network's performance, particularly for datasets with a small training set. However, one benefit that we have observed from increasing the network's depth, is to choose an RF that is long enough to achieve good results without suffering from overfitting.

We therefore proceed by experimenting with varying the RF by changing the filter length.

3.6.4 Filter length

In order to test the effect of the filter length, we start by analyzing how the length of a time series influences the accuracy of the model when tuning this hyperparameter. In these experiments we fixed the number of training time series to 256 and the number of classes to 2. Figure 3.18 illustrates the results of this experiment.

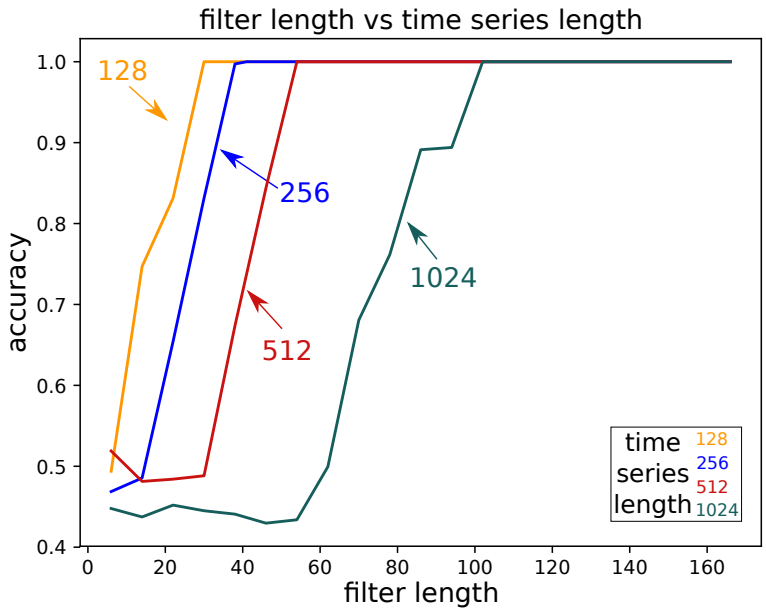


FIGURE 3.18: Inception network's accuracy over the simulated dataset, with respect to the filter length as well as the input time series length.

Length	InceptionTime.8	InceptionTime.64	InceptionTime
<81	**1.71**	2.21	1.79
81-250	1.89	2.11	**1.42**
251-450	2.45	**1.32**	1.86
451-700	2.08	1.85	**1.62**
701-1000	**1.50**	2.60	1.80
>1000	2.14	2.00	**1.71**

TABLE 3.1: Filter length variants of InceptionTime with their corresponding average ranks grouped by the datasets' length. Bold indicates the best model.

We can easily see that as the length of the time series increases, a longer filter is required to produce accurate results. This is explained by the fact that longer kernels are able to capture longer patterns, with higher probability, than shorter ones can. Thus, we can safely say that longer kernels almost always improve accuracy.

In addition to having visualized the accuracy as a function of both depth (Figure 3.16) and filter length (Figure 3.18), we proceed by plotting the accuracy as function of the RF for the simulated time series dataset with various lengths. By observing Figure 3.19 we can confirm the previous observations that longer patterns require longer RFs, with length clearly having a higher impact on accuracy compared to the network's depth. Moreover, by using a large enough RF to cover the whole input time series, the usage of a GAP layer won't affect InceptionTime's ability to discriminate between the two patterns, because performing a GAP does not affect the model's RF.

There is a downside to longer filters however, in the potential for overfitting small datasets, as longer filters significantly increase the number of parameters in the network. To answer this question, we again extend our experiments to the real data from the UCR/UEA archive, allowing us to verify whether long kernels tend

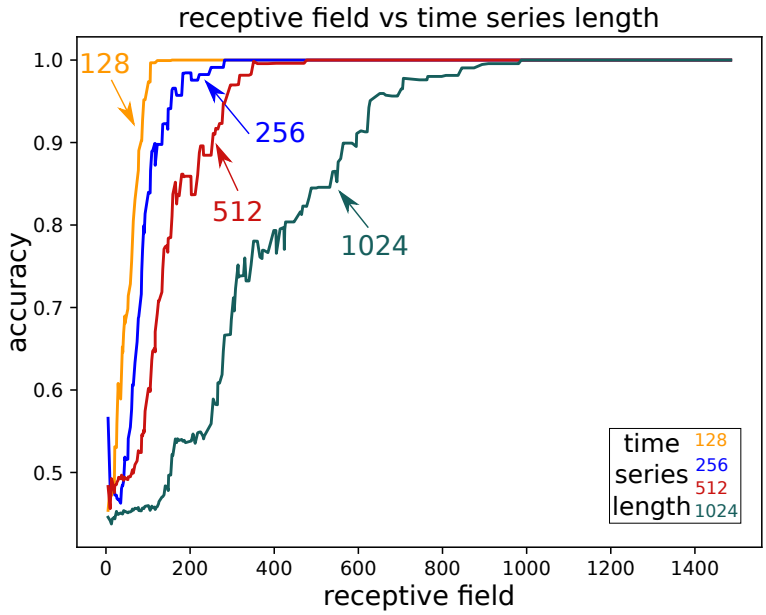


FIGURE 3.19: Inception network's accuracy over the simulated dataset, with respect to the receptive field as well as the input time series length.

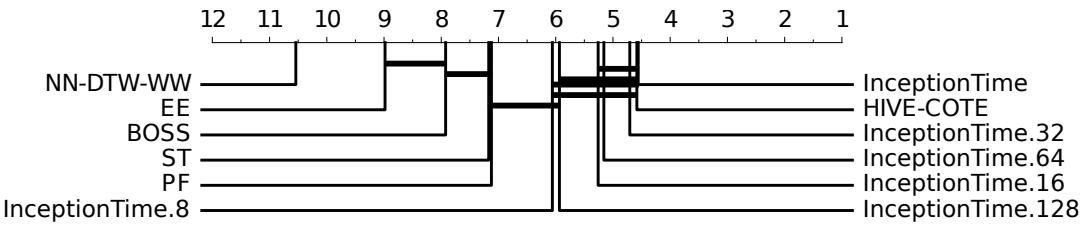


FIGURE 3.20: Critical difference diagram showing the effect of the filter length hyperparameter value over InceptionTime' average rank.

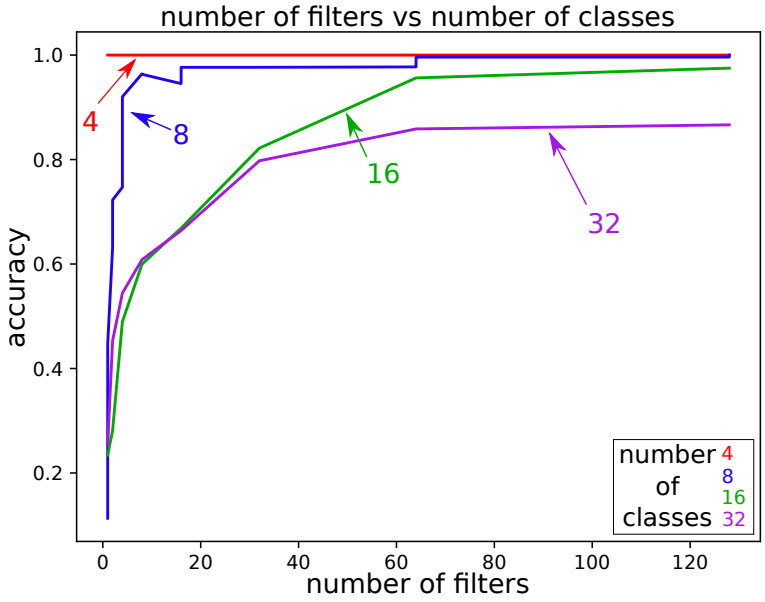


FIGURE 3.21: Inception network's accuracy over the simulated dataset, with respect to the number of filters as well as the number of classes.

to overfit the datasets when a limited amount of training data is available. Therefore, we decided to train and evaluate InceptionTime versions containing both long and short filters on the UCR/UEA archive. Where the original InceptionTime contained filters of length {10,20,**40**}, the five models we are testing here contain filters of length {2,4,**8**}; {4,8,**16**}; {8,16,**32**}; {16,32,**64**}; {32,64,**128**}. Figure 3.20 illustrates a critical difference diagram showing how InceptionTime with longer filters will slightly decrease the network's performance in terms of accurately classifying the time series datasets. We also investigate the relationship between the length of the time series and the length of the network's filter. Table 3.1 depicts the average rank of each variant of InceptionTime over the UCR/UEA archive grouped by the datasets' lengths (with about 15 datasets in each group). Similarly to Figure 3.20, we observe that almost for all time series length, InceptionTime with its default filter length (32) achieves the best or the second best overall accuracy. We can therefore summarize that the results from the simulated dataset do generalize (to some extent) to real datasets: longer filters will improve the model's performance as long as there is enough training data to mitigate the overfitting phenomena.

In summary, we can confidently state that increasing the receptive field of a model by adopting longer filters will help the network in learning longer patterns present in longer time series. However there is an accompanying disclaimer that it may negatively impact the accuracy for some datasets due to overfitting.

3.6.5 Number of filters

To provide some directions on how the number of filters affects the performance of the network, we experimented with varying this hyperparameter with respect to the number of classes in the dataset. To generate new classes in the simulated data, we varied the position and length of the patterns; for example, to create data with three classes, we inject patterns of the same length at three different positions. For this series of experiments, we fixed the length of the time series to 256 and the number of training examples to 256.

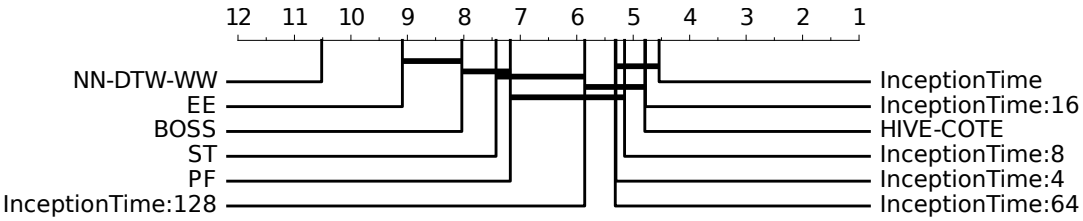


FIGURE 3.22: Critical difference diagram showing how the network's width affects InceptionTime' average rank.

Figure 3.21 depicts the network's accuracy with respect to the number of filters for datasets with a differing number of classes. Our prior intuition was that the more classes, or variability, present in the training set, the more features are required to be extracted in order to discriminate the different classes, and this will necessitate a greater number of filters. This is confirmed by the trend displayed in Figure 3.21, where the datasets with more classes require more filters to be learned in order to be able to accurately classify the input time series.

After observing on the synthetic dataset that the number of filters significantly affects the performance of the network, we asked ourselves if the current implementation of InceptionTime could benefit/lose from a naive increase/decrease in the number of filters per layer. Our proposed InceptionTime model contains 32 filters per Inception module's component, while for these experiments we tested six ensembles, by varying the hyperparameter with a power of two. Figure 3.22 illustrates a critical difference diagram showing how increasing the number of filters per layer significantly deteriorated the accuracy of the network, whereas decreasing the number of filters did not significantly affect the accuracy. It appears that our InceptionTime model contains enough filters to separate the classes of the 85 UCR datasets, of which some have up to 60 classes (ShapesAll dataset).

Increasing the number of filters also has another side effect: it causes an explosion in the number of parameters in the network. The wider InceptionTime contains four times the number of parameters than the original implementation. We therefore conclude that naively increasing the number of filters is actually detrimental, as it will drastically increase the network's complexity and eventually cause overfitting.

3.6.6 Sensitivity analysis

Working with open benchmarks such as the UCR/UEA archive has pushed the community towards publishing high quality TSC algorithms. The UCR/UEA archive provides a train/test split for the data, which has allowed researchers to directly benchmark their works with the ones of others, as well as providing splits that were potentially more challenging and realistic than assuming that both train and test data were sampled from the same population. Having the train/test split available has however also led to the potential issue that the techniques designed on this benchmark archive might overfit it. This is especially true of deep learning classifiers that contain dozens of optimization and architectural hyperparameters.

In an effort to give an idea of the sensitivity of InceptionTime to changes in its parameters, we have evaluated the performance of having chosen the second-best value of each of its parameters, that is the second-best value for the depth of the network (i.e. a value of 9 instead of the best value of 6), for its width (i.e. 16 instead of 32), for the length of the convolutions (final value of 32 instead of 40), for the batch size (i.e. 32 instead of 64), and for the bottleneck size (i.e. 64 instead of the default one

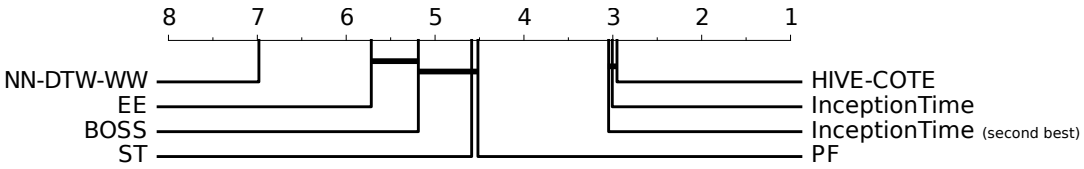


FIGURE 3.23: Critical difference diagram showing how choosing the second best hyperparameters affects InceptionTime's average rank.

32). This gave us a new architecture — InceptionTime$_{\text{(second best)}}$ — which we then compared with InceptionTime and also other algorithms. Figure 3.23 depicts the average rank of current state-of-the-art TSC algorithms with both InceptionTime's default and second best hyperparameters added to the mix. We can clearly see that the effect is minimal: the ranking is a tiny bit lower but they are all well within the critical difference with HIVE-COTE (a post-hoc statistical test fails to reject the null hypothesis (p-value ≈ 0.71) making the difference between the default and second best hyperparameters non significant).

3.7 Conclusion

Deep learning for time series classification still lags behind neural networks for image recognition in terms of experimental studies and architectural designs. In this chapter, we fill this gap by introducing InceptionTime, inspired by the recent success of Inception-based networks for various computer vision tasks. We ensemble these networks to produce new state-of-the-art results for TSC on the 85 datasets of the UCR/UEA archive. Our approach is highly scalable, two orders of magnitude faster than current state-of-the-art models such as HIVE-COTE. The magnitude of this speed up is consistent across both Big Data TSC repositories as well as longer time series with high sampling rate. We further investigate the effects on overall accuracy of various hyperparameters of the CNN architecture. For these, we go far beyond the standard practices for image data, and design networks with long filters. We look at these by using a simulated dataset and frame our investigation in terms of the definition of the receptive field for a CNN for TSC. In the future, we would like to explore how to design DNNs for multivariate TSC while investigating more recent architectural advancements that are being published each year for computer vision tasks.

Chapter 4

Time series analysis for surgical training

4.1 Introduction

Over the past one hundred years, the classic teaching methodology of "see one, do one, teach one" has governed the surgical education systems worldwide. With the advent of Operation Room 2.0, recording video, kinematic and many other types of data during the surgery became an easy task, thus allowing artificial intelligence systems to be deployed and used in surgical and medical practice. Recently, motion sensor data (e.g. kinematics) as well as surgical videos has been shown to provide a structure for peer coaching enabling novice trainees to learn from experienced surgeons by replaying those videos and/or kinematic trajectories. In this chapter, we tackle two problems present in the current surgical training curriculum:

1. Manual feedback from senior surgeons observing less experienced trainees is a laborious task that is very expensive, time-consuming and prone to subjectivity.

2. The high inter-operator variability in surgical gesture duration and execution renders learning from comparing novice to expert surgical videos a very difficult task.

For the first problem, we designed a CNN (inspired by the FCN architecture explained in Chapter 1) to predict surgical skills by extracting latent patterns in the trainees' motions performed during robotic surgery. As for the second problem, we propose to align multiple videos based on the alignment of their corresponding kinematic multivariate time series data, by leveraging the multiple alignment procedure based on DBA (explained in Chapter 2). This chapter is divided into two main sections, each one tackling the aforementioned problem, before concluding with our future work.

4.2 Deep learning for surgical skills evaluation

Over the last century, the standard training exercise of Dr. William Halsted has dominated surgical education in various regions of the world (Polavarapu et al., 2013). His training methodology of "see one, do one, teach one" is still one of the most adopted approaches to date (Ahmidi et al., 2017). The main idea is that the student could become an experienced surgeon by observing and participating in mentored surgeries (Polavarapu et al., 2013). These training techniques, although widely used, lack of an objective surgical skill evaluation method (Kassahun et al., 2016). Standard assessment of surgical skills is presently based on checklists that are filled by

an expert watching the surgical task (Ahmidi et al., 2017). In an attempt to predict a trainee's skill level without using on an expert surgeon's judgement, OSATS was proposed and is currently adopted for clinical practice (Niitsu et al., 2013). Alas, this type of observational rating still suffers from several external and subjective factors such as the inter-rater reliability, the development process and the bias of respectively the checklist and the evaluator (Hatala et al., 2015).

Further studies demonstrated that a vivid relationship occurs between a surgeon's technical skill and the postoperative outcomes (Bridgewater et al., 2003). The latter approach suffers from the fact that the aftermath of a surgery hinges on the physiological attributes of the patient (Kassahun et al., 2016). Furthermore, obtaining this type of data is very strenuous, which renders these skill evaluation techniques difficult to carry out for surgical education. Recent progress in surgical robotics such as the *da Vinci* surgical system (Intuitive Surgical Sunnyvale, 2018) enabled the recording of video and kinematic data from various surgical tasks. Ergo, a substitute for checklists and outcome-based approaches is to generate, from these kinematics, GMFs such as the surgical task's speed, time completion, motion smoothness, curvature and other holistic characteristics (Zia and Essa, 2018; Kassahun et al., 2016). While most of these techniques are efficacious, it is not perspicuous how they could be leveraged to support the trainee with a detailed and constructive feedback, in order to go beyond a naive classification into a skill level (i.e., expert, intermediate, etc.). This is problematic as feedback on medical practice enables surgeons to reach higher skill levels while improving their performance Islam et al., 2016.

Lately, a field entitled *Surgical Data Science* (Maier-Hein et al., 2017) has emerged by dint of the increasing access to a huge amount of complex data which pertain to the staff, the patient and sensors for capturing the procedure and patient related data such as kinematic variables and images (Gao et al., 2014). Instead of extracting GMFs, recent inquiries have a tendency to break down surgical tasks into finer segments called "gestures", manually before training the model, and finally estimate the trainees' performance based on their assessment during these individual gestures (Tao et al., 2012). Even though these methods achieved promising and accurate results in terms of evaluating surgical skills, they necessitate labeling a huge amount of gestures before training the estimator (Tao et al., 2012). We pointed out two major limits in the actual existing techniques that estimate surgeons' skill level from their corresponding kinematic variables: firstly, the absence of an interpretable result of the skill prediction that can be used by the trainees to reach higher surgical skill levels; secondly, the requirement of gesture boundaries that are pre-defined by annotators which is prone to inter-annotator reliability and time-consuming (Vedula et al., 2016).

In this section, we design a novel architecture based on FCNs, dedicated to evaluating surgical skills. By employing one-dimensional kernels over the kinematic time series, we avoid the need to extract unreliable and sensitive gesture boundaries. The original hierarchical structure of our model allows us to capture global information specific to the surgical skill level, as well as to represent the gestures in latent low-level features. Furthermore, to provide an interpretable feedback, instead of using a dense layer like most traditional deep learning architectures (Zhou et al., 2016), we place a GAP layer which allows us to take advantage from the class activation map, proposed originally by Zhou et al., 2016, to localize which fraction of the trial impacted the model's decision when evaluating the skill level of a surgeon. Using a standard experimental setup on the largest public dataset for robotic surgical data analysis: the JHU-ISI Gesture and Skill Assessment Working Set (Gao et al., 2014),

we show the precision of our FCN model. Our main contribution is to demonstrate that deep learning can be leveraged to understand the complex and latent structures when classifying surgical skills and predicting the OSATS score of a surgery, especially since there is still much to be learned on what does exactly constitute a surgical skill (Kassahun et al., 2016).

4.2.1 Background

Here we turn our attention to the recent advances leveraging the kinematic data for surgical skills evaluation. The problem we are interested in requires an input that consists of a set of time series recorded by the da Vinci's motion sensors representing the input surgery and the targeted task is to attribute a skill level to the surgeon performing a trial. One of the earliest work focused on extracting GMFs from kinematic variables and training off-the-shelf classifiers to output the corresponding surgical skill level (Kassahun et al., 2016). Although these methods yielded impressive results, their accuracy depends highly on the quality of the extracted features. As an alternative to GMF-based techniques, recent studies tend to break down surgical tasks into smaller segments called surgical gestures, manually before the training phase, and assess the skill level of the surgeons based on their fine-grained performance during the surgical gestures, for example, using sparse hidden Markov model (Tao et al., 2012). Although the latter technique yields high accuracy, it requires manual segmentation of the surgical trial into fine-grained gestures, which is considered expensive and time-consuming. Hence, recent surgical skills evaluation techniques have focused on algorithms that do not require this type of annotation and are mainly data driven (Ismail Fawaz et al., 2018c; Zia and Essa, 2018; Wang and Majewicz Fey, 2018; Forestier et al., 2017a). For surgical skill evaluation, we distinguish two tasks. The first one is to output the discrete skill level of a surgeon such as novice (N), intermediate (I) or expert (E). For example, Zia and Essa, 2018 adopted the approximate entropy algorithm to extract features from each trial which are later fed to a nearest neighbor classifier. More recently, Wang and Majewicz Fey, 2018 proposed a CNN-based approach to classify sliding windows of time series; therefore, instead of outputting the class for the whole surgery, the network is trained to output the class in an online setting for each window. In Forestier et al., 2018, the authors emphasized the lack of explainability for these latter approaches, by highlighting the fact that interpretable feedback to the trainees is important for a novice to become an expert surgeon (Islam et al., 2016). Therefore, the authors proposed an approach that uses a sliding window technique with a discretization method that transforms the time series into a bag of words and trains a nearest neighbor classifier coupled with the cosine similarity. Then, using the weight of each word, the algorithm is able to provide a degree of contribution for each sliding window and therefore give some sort of useful feedback to the trainees that explains the decision taken by the classifier. Although the latter technique showed interesting results, the authors did sacrifice the accuracy in favor of interpretability. On the other hand, using our fully convolutional neural network, we provide the trainee with an interpretable yet very accurate model by leveraging the class activation map algorithm, originally proposed for computer vision tasks by Zhou et al., 2016. The second type of problem in surgical skill evaluation is to train a model that predicts the OSATS score for a certain surgical trial. For example, Zia and Essa, 2018 extended their ApEn model to predict the OSATS score, also known as global rating score. Interestingly, the latter extension to a regression model instead of a classification one enabled the authors to propose a technique that provides interpretability of the model's decision, whereas

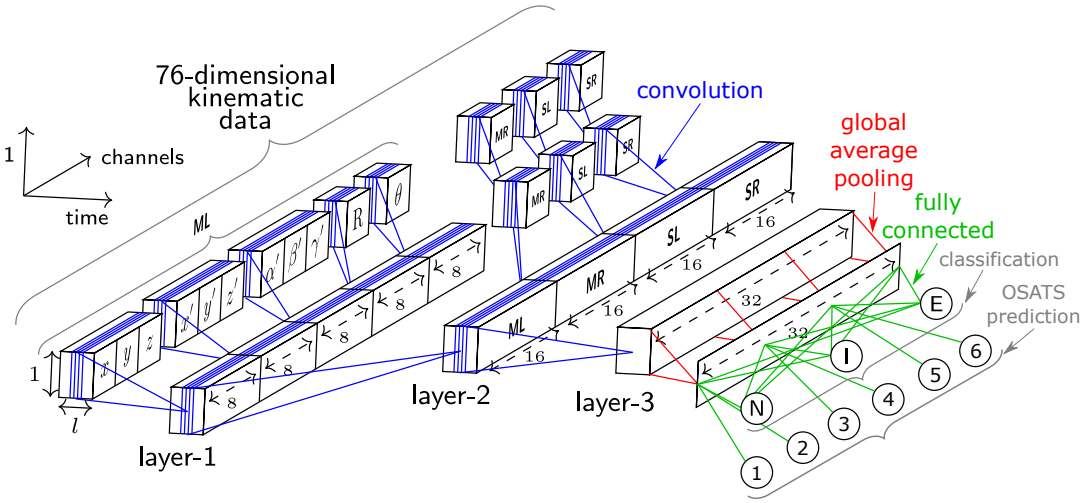


FIGURE 4.1: Fully convolutional network for surgical skill evaluation.

our neural network provides an explanation for both classification and regression tasks.

We present briefly the dataset used in this project as we rely on the features' definitions to describe our method. The JIGSAWS dataset, first published by Gao et al., 2014, has been collected from eight right-handed subjects with three different surgical skill levels: novice (N), intermediate (I) and expert (E), with each group having reported, respectively, less than 10 h, between 10 and 100 h and more than 100 h of training on the Da Vinci. Each subject performed five trials of each one of the three surgical tasks: suturing, needle passing and knot tying. For each trial, the video and kinematic variables were registered. In this project, we focused solely on the kinematics which are numeric variables of four manipulators: right and left masters (controlled by the subject's hands) and right and left slaves (controlled indirectly by the subject via the master manipulators). These 76 kinematic variables are recorded at a frequency of 30 Hz for each surgical trial. Finally, we should mention that in addition to the three self-proclaimed skill levels (N,I,E), JIGSAWS also contains the modified OSATS score (Gao et al., 2014), which corresponds to an expert surgeon observing the surgical trial and annotating the performance of the trainee. The main goal of this work is to evaluate surgical skills by considering either the self-proclaimed discrete skill level (classification) or the OSATS score (regression) as our target variable. We conceive each trial as a multivariate time series and designed a one-dimensional CNN dedicated to learn automatically useful features for surgical skill evaluation in an end-to-end manner (Ismail Fawaz et al., 2019d).

4.2.2 Method

Our approach takes inspiration of the recent success of CNNs for time series classification shown in Chapter 1. Figure 4.1 illustrates the fully convolutional neural network architecture, which we have designed specifically for surgical skill evaluation using temporal kinematic data. The network's input is an MTS with a variable length l and 76 channels. For the classification task, the output layer contains a number of neurons equal to three (N,I,E) with the softmax activation function, whereas for the regression task (predicting the OSATS score), the number of neurons in the last layer is equal to six: (1) "Respect for tissue"; (2) "Suture/needle handling"; (3)

"Time and motion"; (4) "Flow of operation"; (5) "Overall performance"; (6) "Quality of final product" (Gao et al., 2014), with a linear activation function.

Compared with convolutions for image recognition, where usually the model's input exhibits two spatial dimensions (height and width) and three channels (red, green and blue), the input to our network is a time series with one spatial dimension (surgical task's length l) and 76 channels (denoting the 76 kinematics: $x, y, z, \dots$). One of the main challenges we have encountered when designing our architecture was the large number of channels (76) compared to the traditional red, green and blue channels (3) for the image recognition problem. Hence, instead of applying the filters over the whole 76 channels at once, we propose to carry out different convolutions for each group and subgroup of channels. We used domain knowledge when grouping the different channels, in order to decide which channels should be clustered together.

Firstly, we separate the 76 channels into four distinct groups, such as each group should contain the channels from one of the manipulators: the first, second, third and fourth groups correspond to the four manipulators (ML: master left, MR: master right, SL: slave left and SR: slave right) of the *da Vinci* surgical system. Thus, each group assembles 19 of the total kinematic variables. Next, each group of 19 channels is divided into five different subgroups each containing variables that we believe should be semantically clustered together. For each cluster, the variables are grouped into the following five sub-clusters:

- First sub-cluster with three variables for the Cartesian coordinates (x, y, z);
- Second sub-cluster with three variables for the linear velocity (x', y', z');
- Third sub-cluster with three variables for the rotational velocity $(\alpha', \beta', \gamma')$;
- Fourth sub-cluster with nine variables for the rotation matrix R;
- Fifth sub-cluster with one variable for the gripper angular velocity (θ).

Figure 4.1 illustrates how the convolutions in the first layer are different for each subgroup of kinematic variables. Following the same line of thinking, the convolutions in the second layer are different for each group of variables (SL, SR, ML and MR). However, in the third layer, the same filters are applied for all dimensions (or channels), which corresponds to the traditional CNN.

To take advantage from the CAM method while reducing the number of parameters (weights) in our network, we employed a global average pooling operation after the last convolutional layer. In other words, the convolution's output (the MTS) will shrink from a length l to 1, while maintaining the same number of dimensions in the third layer. Without any sort of validation, we choose the following default hyper-parameters. We used 8 kernels for the first convolution, and then we doubled the number of kernels, thus allowing us to balance the number of parameters for each layer as a function of its depth. We used ReLU as the nonlinear hidden activation function for all convolutional layers with a stride of 1 and a kernel length equal to 3.

We fixed our objective loss function to be the categorical cross-entropy to learn the network's parameters in an end-to-end manner for the classification task, and the mean squared error when learning a regressor to predict the OSATS score, which can be written as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2. \tag{4.1}$$

Method	Suturing			Needle passing			Knot tying		
	Micro	Macro	ρ	Micro	Macro	ρ	Micro	Macro	ρ
S-HMM (Tao et al., 2012)	97.4	n/a	n/a	96.2	n/a	n/a	94.4	n/a	n/a
ApEn (Zia and Essa, 2018)	**100**	n/a	0.59	**100**	n/a	0.45	**99.9**	n/a	**0.66**
Sax-Vsm (Forestier et al., 2017a)	89.7	86.7	n/a	96.3	95.8	n/a	61.1	53.3	n/a
CNN (Wang and Majewicz Fey, 2018)	93.4	n/a	n/a	89.9	n/a	n/a	84.9	n/a	n/a
FCN (proposed)	**100**	**100**	**0.60**	**100**	**100**	**0.57**	92.1	**93.2**	0.65

TABLE 4.1: Micro, macro and Spearman's coefficient ρ for surgical skill evaluation.

The network's weights were optimized using the Adam optimization algorithm (Kingma and Ba, 2015). The default value of the learning rate was fixed to 0.001 as well as the first and second moment estimates were set to 0.9 and 0.999 respectively (Chollet, 2015). We initialized the weights using Glorot's uniform initialization (Glorot and Bengio, 2010). We randomly shuffled the training set before each epoch, whose maximum number was set to 1000 epochs. We then saved the model at each training iteration by choosing the network's state that minimizes the loss function on a random (non-seen) split from the training data. This process is also referred to as "model checkpoint" by the deep learning community (Chollet, 2015), allowing us to choose the best number of epochs based on the validation loss. Finally, to avoid overfitting, we added an $l2$ regularization parameter whose default value was fixed to 10^{-5}. For each surgical task, we have trained a different network, resulting in three different models.[1] We adopted for both classification and regression tasks a leave-one-super-trial-out scheme (Ahmidi et al., 2017).

The use of a GAP layer allows us to employ the CAM algorithm, which was originally designed for image classification tasks by Zhou et al., 2016 and later introduced for time series data in Wang, Yan, and Oates, 2017. Using the CAM, we are able to highlight which fractions of the surgical trial contributed highly to the classification. This method was discussed in details in Chapter 1. Finally, for the regression task, the CAM can be extended in a trivial manner: Instead of computing the contribution to a classification, we are computing the contribution to a certain score prediction (1 out of 6 in total).

4.2.3 Results

The first task consists in assigning a skill level for an input surgical trial out of the three possible levels: novice (N), intermediate (I) and expert (E). In order to compare with current state-of-the-art techniques, we adopted the *micro* and *macro* measures defined in Ahmidi et al., 2017. The *micro* measure refers simply to the traditional *accuracy* metric. However, the *macro* takes into consideration the support of each class in the dataset, which boils down to computing the *precision* metric. Table 4.1 reports the *macro* and *micro* metrics of five different models for the surgical skill classification of the three tasks: suturing, knot tying and needle passing. For the proposed FCN model, we average the accuracy over 40 runs to reduce the bias induced by the randomness of the optimization algorithm. From these results, it appears that FCN is much more accurate than the other approaches with 100% accuracy for the needle passing and suturing tasks. As for the knot tying task, we report 92.1% and 93.2%, respectively, for the *micro* and *macro* configurations. When comparing the other four

[1] Our source code is available at https://github.com/hfawaz/ijcars19

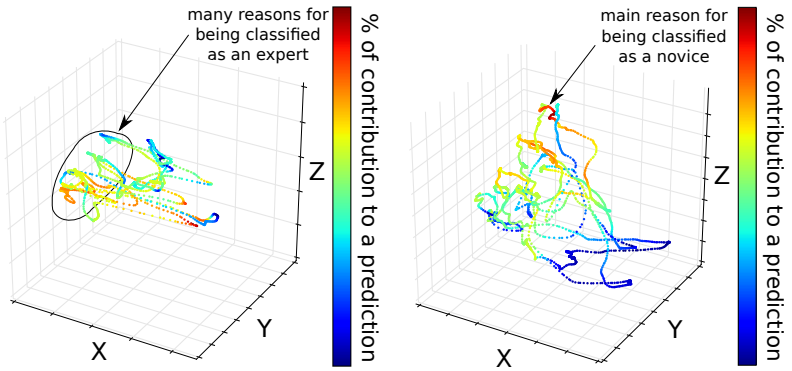
techniques, for the knot tying surgical task, FCN exhibits relatively lower accuracy, which can be explained by the minor difference between the experts and intermediates for this task: Mean OSATS score is 17.7 and 17.1 for expert and intermediate, respectively.

An S-HMM was designed to classify surgical skills (Tao et al., 2012). Although this approach does leverage the gesture boundaries for training purposes, our method is much more accurate without the need to manually segment each surgical trial into finer gestures. Zia and Essa, 2018 introduced ApEn to generate characteristics from each surgical task, which are later given to a classical nearest neighbor classifier with a cosine similarity metric. Although ApEn and FCN achieved state-of-the-art results with 100% accuracy for the first two surgical tasks, it is still not obvious how ApEn could be used to give feedback for the trainee after finishing his/her training session. Forestier et al., 2017a introduced a sliding window technique with a discretization method to transform the MTS into bag of words. To justify their low accuracy, Forestier et al., 2017a insisted on the need to provide *explainable* surgical skill evaluation for the trainees. On the other hand, FCN is equally *interpretable* yet much more accurate; in other words, we do not sacrifice accuracy for interpretability. Finally, Wang and Majewicz Fey, 2018 designed a CNN whose architecture is dependent on the length of the input time series. This technique was clearly outperformed by our model which reached better accuracy by removing the need to pre-process time series into equal length thanks to the use of GAP.

We extended the application of our FCN model (Ismail Fawaz et al., 2018c) to the regression task: predicting the OSATS score for a given input time series. Although the community made a huge effort toward standardizing the comparison between different surgical skills evaluation techniques (Ahmidi et al., 2017), we did not find any consensus over which evaluation metric should be adopted when comparing different regression models. However, Zia and Essa, 2018 proposed the use of Spearman's correlation coefficient (denoted by ρ) to compare their 11 combination of regression models. The latter is a nonparametric measure of rank correlation that evaluates how well the relationship between two distributions can be described by a monotonic function. In fact, the regression task requires predicting six target variables; therefore, we compute ρ for each target and finally report the corresponding mean over the six predictions. By adopting the same validation methodology proposed by Zia and Essa, 2018, we are able to compare our proposed FCN model to their best performing method. Table 4.1 reports also the ρ values for the three tasks, showing how FCN reaches higher ρ values for two out of three tasks. In other words, the prediction and the ground truth OSATS score are more correlated when using FCN than the ApEn-based solution proposed by Zia and Essa, 2018 for the second task and equally correlated for the other two tasks.

The CAM technique allows us to visualize which parts of the trial contributes the most to a skill classification. By localizing, for example, discriminative behaviors specific to a skill level, observers can start to understand motion patterns specific to certain class of surgeons. To further improve themselves (the novice surgeons), the model, using the CAM's result, can pinpoint to the trainees their good/bad motor behaviors. This would potentially enable novices to achieve greater performance and eventually become experts.

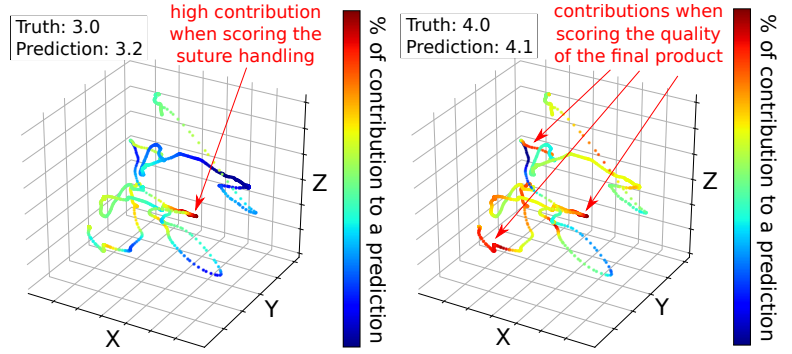
By generating a heatmap from the CAM, we can see in Figure 4.2 how it is indeed possible to visualize the feedback for the trainee. In fact, we examine a trial of an expert and novice surgeon: The expert's trajectory is illustrated in Figure 4.2a while the novice's trajectory is depicted in Figure 4.2b. In this example, we can see how the model was able to identify which motion (red subsequence) is the main reason



(A) Suturing task for an expert

(B) Suturing task for a novice

FIGURE 4.2: Using class activation map to provide explainable classification.



(A) Suture/needle handling

(B) Quality of the final product

FIGURE 4.3: Feedback using the CAM on subject E's second knot-tying trial.

for identifying a subject as a novice. Concretely, we can easily spot a pattern that is being recognized by the model when outputting the classification of subject H's skill level: The orange and red 3D subsequences correspond to same surgical gesture "pulling suture" and are exhibiting a high influence over the model's decision. This feedback could be used to explain to a young surgeon which movements are classifying him/her as a novice and which ones are classifying another subject as an expert. Thus ultimately, this sort of feedback could guide the novices into becoming experts.

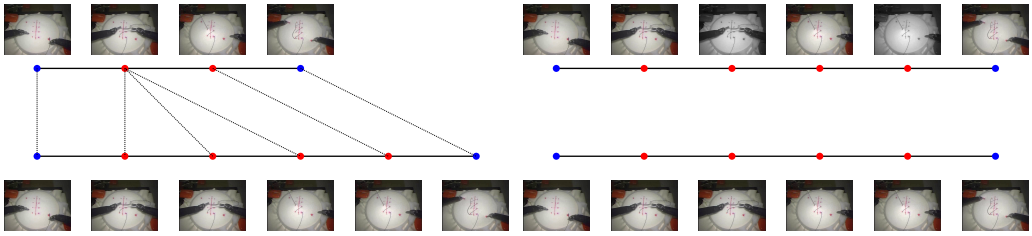
After having shown how our *classifier* can be interpreted to provide feedback to the trainees, we now present the result of applying the same visualization (based on the CAM algorithm) in order to explain the OSATS score prediction. Figure 4.3 depicts the trajectory with its associated heatmaps for subject E performing the second trial of the knot-tying task. Figure 4.3a and 4.3b illustrates the trajectory's heatmap, respectively, for "suture/needle handling" and "quality of the final product" OSATS score predictions. At first glimpse, one can see how a prediction that requires focusing on the whole surgical trial leverages more than one region of the input surgery—this is depicted by the multiple red subsequences in Figure 4.3b. However, when outputting a rating for a specific task such as "suture/needle handling"—the model is focusing on less parts of the input trajectory which is shown in Figure 4.3a.

4.2.4 Conclusion

In this project, we proposed a deep learning-based method for surgical skills evaluation from kinematic data. We achieved state-of-the-art accuracy by designing a specific FCN, while providing explainability that justifies a certain skill evaluation, thus allowing us to mitigate the CNN's black-box effect. Furthermore, by extending our architecture we were able to provide new state-of-the-art performance for predicting the OSATS score from the input kinematic time-series data. In the following section, we present a technique for automatic alignment of surgical videos from kinematic time series (Ismail Fawaz et al., 2019c) for surgical skills evaluation.

4.3 Automatic alignment of surgical videos using kinematic time series data

Educators have always searched for innovative ways of improving apprentices' learning rate. While classical lectures are still most commonly used, multimedia resources are becoming more and more adopted (Smith and Ransbottom, 2000) especially in Massive Open Online Courses (Means et al., 2009). In this context, videos have been considered as especially interesting as they can combine images, text, graphics, audio and animation. The medical field is no exception, and the use of video-based resources is intensively adopted in medical curriculum (Masic, 2008) especially in the context of surgical training (Kneebone et al., 2002). The advent of robotic surgery also simulates this trend as surgical robots, like the Da Vinci (Intuitive Surgical Sunnyvale, 2018), generally record video feeds during the intervention. Consequently, a large amount of video data has been recorded in the last ten years (Rapp et al., 2016). This new source of data represent an unprecedented opportunity for young surgeons to improve their knowledge and skills (Gao et al., 2014). Furthermore, video can also be a tool for senior surgeons during teaching periods to assess the skills of the trainees. In fact, a recent study by Mota et al., 2018 showed that residents spend more time viewing videos than specialists, highlighting the need for



(A) Video without alignment

(B) Video with alignment

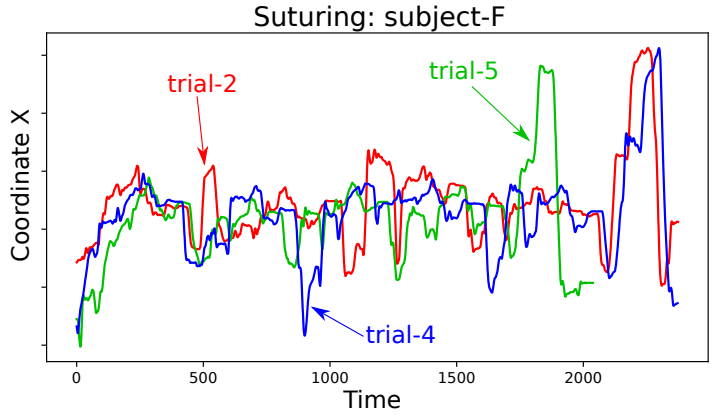
FIGURE 4.4: Example on how a time series alignment is used to synchronize the videos by duplicating the gray-scale frames. Best viewed in color.

young surgeons to fully benefit from the procedure. In Herrera-Almario et al., 2016, the authors showed that knot-tying scores and times for task completion improved significantly for the subjects that watched the videos of their own performance.

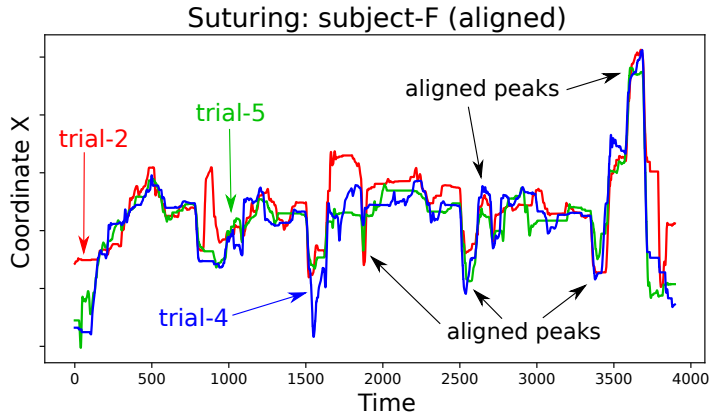
However, when the trainees are willing to asses their progress over several trials of the same surgical task by re-watching their recorded surgical videos simultaneously, the problem of videos being out-of-synch makes the comparison between different trials very difficult if not impossible. This problem is encountered in many real life case studies, since experts on average complete the surgical tasks in less time than novice surgeons (McNatt and Smith, 2001). Thus, when trainees do enhance their skills, providing them with a feedback that pinpoints the reason behind the surgical skill improvement becomes problematic since the recorded videos exhibit different duration and are not perfectly aligned.

Although synchronizing videos has been the center of interest for several computer vision research venues, contributions are generally focused on a special case where multiple simultaneously recorded videos (with different characteristics such as viewing angles and zoom factors) are being processed (Wolf and Zomet, 2002; Wedge, Kovesi, and Huynh, 2005; Padua et al., 2010). Another type of multiple video synchronization uses hand-engineered features (such as points of interest trajectories) from the videos (Wang et al., 2014; Evangelidis and Bauckhage, 2011), making the approach highly sensitive to the quality of the extracted features. This type of techniques was highly effective since the raw videos were the only source of information available, whereas in our case, the use of robotic surgical systems enables capturing an additional type of data: the kinematic variables such as the x, y, z Cartesian coordinates of the Da Vinci's end effectors (Gao et al., 2014).

In this section, we propose to leverage the sequential aspect of the recorded kinematic data from the Da Vinci surgical system, in order to synchronize their corresponding video frames by aligning the time series data (see Figure 4.4 for an example). When aligning two time series, the off-the-shelf algorithm is DTW (Sakoe and Chiba, 1978) which we indeed used to align two videos. However, when aligning multiple sequences, the latter technique does not generalize in a straightforward and computationally feasible manner (Petitjean et al., 2014). Hence, for multiple video synchronization, we propose to align their corresponding time series to the average time series, computed using the DBA algorithm (explained in details in Chapter 2). This process is called Non-Linear Temporal Scaling and has been originally proposed to find the multiple alignment of a set of discretized surgical gestures (Forestier et al., 2014), which we extend in this work to continuous numerical kinematic data. Figure 4.5 depicts an example of stretching three different time series



(A) Original time series without alignment



(B) Warped time series with alignment

FIGURE 4.5: Example of aligning coordinate X's time series for subject F, when performing three trials of the suturing surgical task.

using the NLTS algorithm. Examples of the synchronized videos and the associated code can be found on our GitHub repository[2], where we used the JHU-ISI Gesture and Skill Assessment Working Set (Gao et al., 2014) to validate our work.

4.3.1 Methods

Here, we detail each step of our video synchronization approach. We start by describing the DTW algorithm which allows us to align two videos. Then, we describe how NLTS enables us to perform multiple video synchronization with respect to the reference average time series computed using the DBA algorithm.

Dynamic Time Warping

Dynamic Time Warping was first proposed for speech recognition when aligning two audio signals (Sakoe and Chiba, 1978). Suppose we want to compute the dissimilarity between two time series, for example two different trials of the same surgical task, $A = (a_1, a_2, \ldots, a_m)$ and $B = (b_1, b_2, \ldots, b_n)$. The length of A and B are denoted respectively by m and n, which in our case correspond to the surgical trial's duration. Here, a_i is a vector that contains six real values, therefore A and B can be seen as two distinct MTS.

[2] https://github.com/hfawaz/aime19

To compute the DTW dissimilarity between two MTS, several approaches were proposed by the time series data mining community (Shokoohi-Yekta et al., 2017), however in order to apply the subsequent algorithm NLTS, we adopted the "dependent" variant of DTW where the Euclidean distance is used to compute the difference between two instants i and j. Let $M(A, B)$ be the $m \times n$ point-wise dissimilarity matrix between A and B, where $M_{i,j} = ||a_i - b_j||^2$. A warping path $P = ((c_1, d_1), (c_2, d_2), \ldots, (c_s, d_s))$ is a series of points that define a crossing of M. The warping path must satisfy three conditions: (1) $(c_1, d_1) = (1, 1)$; (2) $(c_s, d_s) = (m, n)$; (3) $0 \leq c_{i+1} - c_i \leq 1$ and $0 \leq d_{j+1} - d_j \leq 1$ for all $i < m$ and $j < n$. The DTW measure between two series corresponds to the path through M that minimizes the total distance. In fact, the distance for any path P is equal to $D_P(A, B) = \sum_{i=1}^{s} P_i$. Hence if $\mathbf{P}$ is the space of all possible paths, the optimal one - whose cost is equal to $DTW(A, B)$ - is denoted by P^* and can be computed using: $\min_{P \in \mathbf{P}} D_P(A, B)$.

The optimal warping path can be obtained efficiently by applying a dynamic programming technique to fill the cost matrix M. Once we find this optimal warping path between A and B, we can deduce how each time series element in A is linked to the elements in B. We propose to exploit this link in order to identify which time stamp should be duplicated in order to align both time series, and by duplicating a time stamp, we are also duplicating its corresponding video frame. Concretely, if elements a_i, a_{i+1} and a_{i+2} are aligned with the element b_j when computing P^*, then by duplicating twice the video frame in B for the time stamp j, we are dilating the video of B to have a length that is equal to A's. Thus, re-aligning the video frames based on the aligned Cartesian coordinates: if subject S_1 completed "*inserting the needle*" gesture in 5 seconds, whereas subject S_2 performed the same gesture within 10 seconds, our algorithm finds the optimal warping path and duplicates the frames for subject S_1 in order to synchronize with subject S_2 the corresponding gesture. Figure 4.4 illustrates how the alignment computed by DTW for two time series can be used in order to duplicate the corresponding frames and eventually synchronize the two videos.

Non-Linear Temporal Scaling

The previous DTW based algorithm works perfectly when synchronizing only two surgical videos. The problem arises when aligning three or more surgical trials simultaneously, which requires a multiple series alignment. The latter problem has been shown to be NP-Complete (Wang and Jiang, 1994) with the exact solution requiring $O(L^N)$ operations for N sequences of length L. This is clearly not feasible in our case where L varies between 10^3 and 10^4 and $N \geq 3$, which is why we ought to leverage an approximation of the multiple sequence alignment solution provided by the DBA algorithm which we detailed in the Chapter 2.

NLTS was originally proposed for aligning discrete sequences of surgical gestures (Forestier et al., 2014). In this project, we extend the technique for numerical continuous sequences (time series). The goal of this final step is to compute the approximated multiple alignment of a set of sequences D which will eventually contain the precise information on how much a certain frame from a certain series should be duplicated. We first start by computing the average sequence T (using DBA) for a set of time series D that we want to align simultaneously. Then, by recomputing the Compact Multiple Alignment between the refined average T and the set of time series D, we can extract an alignment between T and each sequence in D. Thus, for each time series in D we will have the necessary information (extracted from the multiple alignment) in order to dilate the time series appropriately to have a length

FIGURE 4.6: Snapshots of the three surgical tasks in the JIGSAWS dataset (from left to right): suturing, knot-tying, needle-passing (Gao et al., 2014).

that is equal to T's, which also corresponds to the length of the longest time series in D. Figure 4.2, depicts an example of aligning three different time series using the NLTS algorithm.

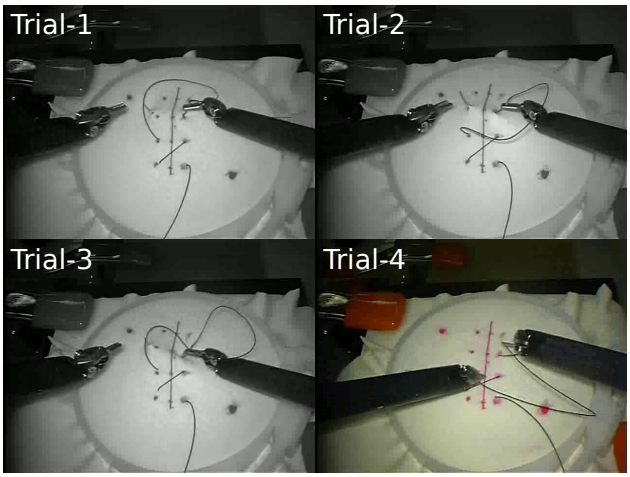
4.3.2 Experiments

We start by describing the JIGSAWS dataset we have used for evaluation, before presenting our experimental study.

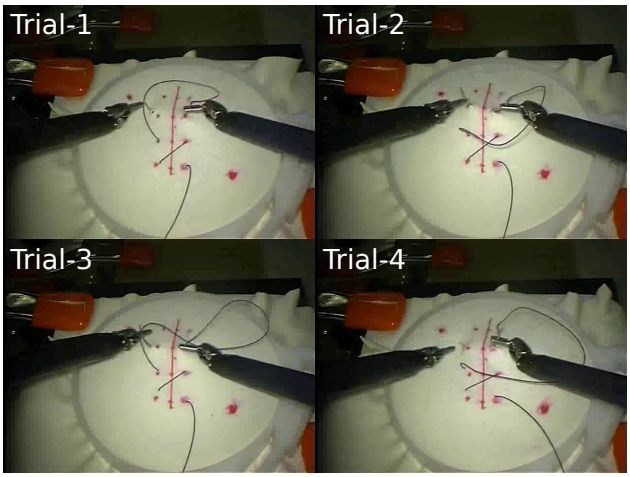
Dataset

The JIGSAWS dataset (Gao et al., 2014) includes data for three basic surgical tasks performed by study subjects (surgeons). The three tasks (or their variants) are usually part of the surgical skills training program. Figure 4.6 shows a snapshot example for each one of the three surgical tasks (Suturing, Knot Tying and Needle Passing). The JIGSAWS dataset contains kinematic and video data from eight different subjects with varying surgical experience: two experts (E), two intermediates (I) and four novices (N) with each group having reported respectively more than 100 hours, between 10 and 100 hours and less than 10 hours of training on the Da Vinci. All subjects were reportedly right-handed.

The subjects repeated each surgical task five times and for each trial the kinematic and video data were recorded. When performing the alignment, we used the kinematic data which are numeric variables of four manipulators: left and right masters (controlled directly by the subject) and left and right slaves (controlled indirectly by the subject via the master manipulators). These kinematic variables (76 in total) are captured at a frequency equal to 30 frames per second for each trial. Out of these 76 variables, we only consider the Cartesian coordinates (x, y, z) of the left and right slave manipulators, thus each trial will consist of an MTS with 6 temporal variables. We chose to work only with this subset of kinematic variables to make the alignment coherent with what is visible in the recorded scene: the robots' end-effectors which can be seen in Figure 4.6. However other choices of kinematic variables are applicable, which we leave the exploration for our future work. Finally we should mention that in addition to the three self-proclaimed skill levels (N,I,E) JIGSAWS contains the modified OSATS score (Gao et al., 2014), which corresponds to an expert surgeon observing the surgical trial and annotating the performance of the trainee.



(A) Videos synchronization process



(B) Perfectly aligned videos

FIGURE 4.7: Video alignment procedure with duplicated (gray-scale) frames.

Results

We have created a companion web page[3] to our project where several examples of synchronized videos can be found. Figure 4.7 illustrates the multiple videos alignment procedure using our NLTS algorithm, where gray-scale images indicate duplicated frames (paused video) and colored images indicate a surgical motion (unpaused video). In Figure 4.7a we can clearly see how the gray-scale surgical trials are perfectly aligned. Indeed, the frozen videos show the surgeon ready to perform "*pulling the needle*" gesture (Gao et al., 2014). On the other hand, the colored trial (bottom right of Figure 4.7a) shows a video that is being played, where the surgeon is performing "*inserting the needle*" gesture in order to catch up with the other paused trials in gray-scale. Finally, the result of aligning simultaneously these four surgical trials is depicted in Figure 4.7b. By observing the four trials, one can clearly see that the surgeon is now performing the same surgical gesture "*pulling the needle*" simultaneously for the four trials. We believe that this type of observation will enable a novice surgeon to locate which surgical gestures still need some improvement in order to eventually become an expert surgeon.

[3] https://germain-forestier.info/src/aime2019/

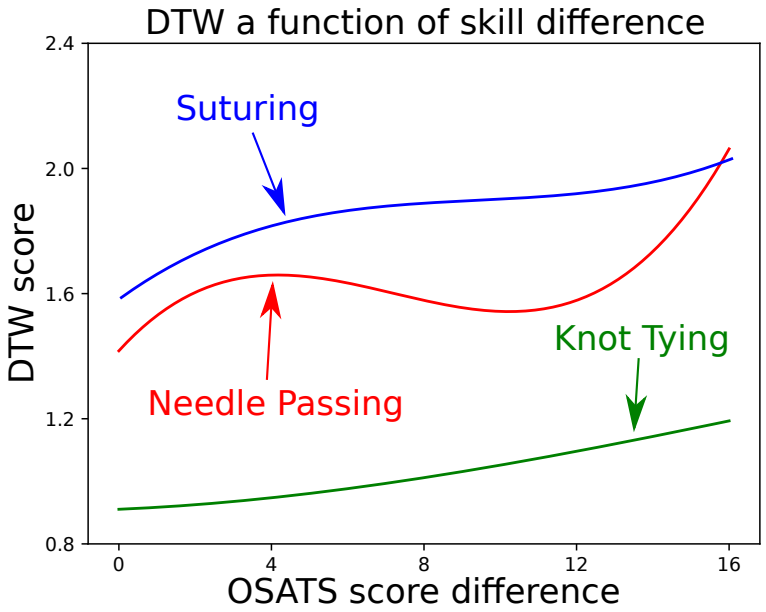


FIGURE 4.8: A polynomial fit (degree 3) of DTW dissimilarity score (y-axis) as a function of the OSATS score difference between two surgeons (x-axis).

Furthermore, in order to validate our intuition that DTW is able to capture characteristics that are in relationship with the motor skill of a surgeon, we plotted the DTW distance as a function of the OSATS (Gao et al., 2014) score difference. For example, if two surgeons have both an OSATS score of 10 and 16 respectively, the corresponding difference is equal to $|10 - 16| = 6$. In Figure 4.8, we can clearly see how the DTW score increases whenever the OSATS score difference increases. This observation suggests that the DTW score is low when both surgeons exhibit similar dexterity, and high whenever the trainees show different skill levels. Therefore, we conclude that the DTW score can serve as a heuristic for estimating the quality of the alignment (whenever annotated skill level is not available) - especially since we observed low quality alignments for surgeons with very distinct surgical skill levels.

Finally, we should note that this work is suitable for many research fields involving motion kinematic data with their corresponding video frames. Examples of such medical applications are assessing mental health from videos (Yamada and Kobayashi, 2017) where wearable sensor data can be seen as time series kinematic variables and leveraged in order to synchronize a patient's videos and compare how well the patient is responding to a certain treatment. Following the same line of thinking, this idea can be further applied to kinematic data from wearable sensors coupled with the corresponding video frames when evaluating the Parkinson's disease evolution (Criss and McNames, 2011) as well as infant grasp skills (Li et al., 2019).

4.3.3 Conclusion

In this section, we showed how kinematic time series data recorded from the Da Vinci's end effectors can be leveraged in order to synchronize the trainee's videos performing a surgical task. With personalized feedback during surgical training becoming a necessity (Ismail Fawaz et al., 2018a; Forestier et al., 2018), we believe that replaying *synchronized* and well *aligned* videos would benefit the trainees in understanding which surgical gestures did or did not improve after hours of training, thus

enabling them to further reach higher skills and eventually become experts. We acknowledge that this work needs an experimental study to quantify how beneficial is replaying synchronized videos for the trainees versus observing non-synchronized trials. Therefore, we leave such exploration and clinical try outs to our future work.

4.4 Conclusion

In this chapter, we tackled two different surgical skills related problems. First by designing a one dimensional FCN, we were able to reach state-of-the-art results for the surgical skills evaluation (classification and regression). Doing so, we were able to mitigate the DNNs' black-box effect, using the CAM technique in order to highlight the reason behind a certain surgical skill identification. The second surgical skill related problem was due to surgical training videos being our-of-synch, thus making it hard for trainees to understand and compare videos between different surgeons with various skill levels. We tackled the latter problem by proposing the use of NLTS in order to align and synchronize multiple videos simultaneously. These two projects were orthogonal in the sense that they could also complement each other: perhaps video and time series synchronization could be a pre-processing step that would enhance the performance surgical skills evaluation models. We leave such exploration to our future work when extending these two projects.

Conclusion and future works

Overview of contributions

The work presented in this manuscript first examined the current benchmark approaches in the area of deep learning for time series classification. Using the knowledge gained from this study, a number of new methods have been proposed to contribute to the deep learning for TSC literature, leading up to the proposal for the final classifier InceptionTime, which to the best of our knowledge constitutes the first time series classifier to achieve similar results to the famous HIVE-COTE ensemble, while outperforming it significantly in terms of running-time. We should emphasize that in order to foster reproducibility, the corresponding code for each paper is published on the following GitHub profile: https://github.com/hfawaz.

The main goal of this research was to answer the question outlined in Chapter 1: Is there a current DNN approach that reaches state-of-the-art performance for TSC and is less complex than HIVE-COTE? Most previous work on TSC haven't considered deep learning models as potential strong classifiers, which led to researchers proposing neural networks as baselines with little to no competitive performance in terms of accuracy when compared to other non deep learning based techniques such as HIVE-COTE. Arriving to our benchmark in 2018, we showed how neural networks can compete with state-of-the-art TSC algorithms, while providing interpretability using the CAM method.

Furthermore, in a series of research projects around regularizing DNNs for TSC, we were able to showcase how much more accuracy we can squeeze from a vanilla neural network architecture designed originally as a baseline. We started by proposing a novel DBA-based method for predicting the best source dataset for a given target dataset when fine-tuning a neural network classifier in a transfer learning setting. Following the ensembles trend of time series classifiers, we showed how we can leverage the variance in a DNN due to the stochastic nature of its optimization process, in order to gain a significant boost in accuracy by ensembling various neural networks with the same or different architectures. We then proposed to further build again upon the DBA algorithm to generate synthetic time series, allowing us to augment the training set and eventually improve the generalization capability of a deep learning classifier. Finally, we turned our attention to a very hot and trending topic in machine learning: adversarial attacks. We showed how vulnerable DNNs are to adversarial examples and highlighted several use case studies where such attacks could be detrimental, while showing how the latter technique can be employed as part of a regularization method called adversarial training.

Building upon this knowledge gained from studying the field of TSC, we identified a main bottleneck that hinders the practical usage of many published ensembles in real life time series data mining problems. This downside stems from focusing on developing very accurate classifiers, while ignoring the running time of the classifier. We were therefore keen on developing the first neural network ensemble (called

InceptionTime) that is able to reach similar results to the current state-of-the-art ensemble HIVE-COTE, while providing scalability in terms of long and large time series dataset. The magnitude of this speed up is consistent across both Big Data TSC repositories as well as longer time series with high sampling rate.

Motivated originally by the problem of evaluating surgical skills from kinematic time series sensor data, we designed a special CNN network by leveraging our understanding of DNNs gained during the initial study conducted on domain agnostic TSC problems. For this specific surgical data science project, we encountered many challenges that were not present in other TSC problems: such as having very high sampling rates and a huge number of variables to work with. For this task, we provided an FCN based classifier while focusing on its interpretability in order to help surgeons using the system to improve their skill set.

Having summarized the aforementioned projects, we will present in the next section the different limitations of our approaches as well as the potential research area that could be of interest to many TSC researchers looking deeper into artificial neural networks.

Discussion of future works

The deep learning benchmark for TSC published during this thesis has taken a big leap towards introducing time series data mining practitioners to the potential of neural networks when classifying sequential temporal data. While still being one of the largest studies of deep learning for TSC to date, our review focused on a small subset of discriminative end-to-end DNNs. The latter means that we have excluded many types of approaches based on self-supervised learning such as training a neural network on a pre-text task with large unlabeled dataset then using the learned latent representation as input features to an off-the-shelf classifier (Franceschi, Dieuleveut, and Jaggi, 2019). Since self-supervised learning has allowed computer vision researchers to leverage efficiently the large amount of unlabeled images and videos on the web (Jing and Tian, 2020), we believe that the TSC community would benefit from benchmarking this type of approaches, allowing the exploitation of a large quantity of unlabeled raw time series data.

Following the review in Chapter 1, we have presented various regularization methods that help in improving the generalization capabilities of a given neural network for TSC. We believe that there exist much more research potential in developing specific time series data augmentation techniques that make use of the temporal aspect of the data. For example we could leverage a recent proposed approach that allows the network to learn the optimal warping by applying continuous piecewise affine transformations (Weber et al., 2019), and thus generating an infinite number of warped time series training examples. Furthermore, restricting transfer learning to a single architecture limits the potential of this famous implicit regularization technique, therefore it would be interesting to study how various architectures could benefit / harm the neural network's accuracy when used in a fine-tuning setting. In addition, having showed that ensembling DNNs by averaging the output class provabilities leads to a certain increase in the network's performance, we believe that there is still room for improvement, since averaging the a posteriori probability for each class is a very basic approach and perhaps a more robust meta-ensembling technique would demonstrate further improvements (Boubrahimi, Ma, and Angryk,

2018). Finally, with adversarial attacks exposing the vulnerabilities of neural networks, we believe that more and more researchers should focus on developing defenses for TSC models. In fact, the work done in this thesis on crafting adversarial examples has already inspired researchers world wide to propose temporal defense mechanisms (Abdu-Aguye et al., 2020).

In Chapter 3 we focused on the scalability of current state-of-the-art methods and proposed an Inception based network for TSC. We were inspired by the recent neural architecture advances in the field of computer vision. Following the same line of thinking, researchers would further adapt the current state-of-the-art machine learning approaches such as designing algorithms that themselves would design a neural network. Rakhshani et al., 2019 spotted the potential of meta-heuristics - specifically differential evolution - when building neural networks that are specific to each TSC problem. We therefore believe that InceptionTime is not the ultimate solution to all TSC datasets, but rather a strong and robust starting point to any data mining practitioner that would like to design a DNN for solving their TSC problem. Furthermore, perhaps DNNs are not the sole answer when choosing the best TSC algorithm. In fact, when referring to the "no-free-lunch theorem" developed by Wolpert, Macready, et al., 1995, and based on our results we believe that perhaps the best approach, is a meta-algorithm that would predict the best classifier based on characteristics extracted from the data.

Going back to our initial motivation for solving TSC problems in Chapter 4, we developed an FCN based classifier for MTS classification, allowing us to leverage the CAM technique to provide a level of model interpretability. We would like to first highlight the fact that our feedback technique would benefit from an extended real use-case validation process, for example verifying with expert surgeons if indeed the model is able to detect the main reason for classifying a surgical skill. However, this is usually expensive and was not possible during this thesis, but perhaps a research project in collaboration with expert and novice surgeons would be extremely interesting to the community. In addition, the fact that we are performing only a leave-one-super-trial-out setup means that a surgeon should be present in the training set in order to make a prediction. However, since only two experts exist in the dataset, this suggests that performing a leave-one-user-out setup would mean having only one expert in the training set. This constitutes a huge problem originating from the limited dataset size. Therefore, we believe that our approach should be validated on a larger dataset. Nevertheless, another potential solution is to perform adversarial learning in order to force the network to detect patterns that are discriminative in terms of surgical skills but not invariant in terms of subject ID. This would be inspired from the recent success of adversarial learning for speaker invariant speech recognition systems (Adi et al., 2019).

Finally, with deep learning showing successful results in various machine learning fields such text mining, image segmentation and speech recognition, we believe that there is still much more to be explored for the TSC research area, especially since the deep learning discipline is moving very fast. We hope that this thesis, with its accompanied code and published models, could be a cornerstone for future deep learning research on time series classification.

Bibliography

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Accessed 28 Feb 2019. URL: https://www.tensorflow.org/.

Abadi, Martín et al. (2016). "TensorFlow: A System for Large-scale Machine Learning". In: *USENIX Conference on Operating Systems Design and Implementation*, pp. 265–283.

Abdelfattah, Sherif M, Ghodai M Abdelrahman, and Min Wang (2018). "Augmenting The Size of EEG datasets Using Generative Adversarial Networks". In: *International Joint Conference on Neural Networks*, pp. 1–6.

Abdu-Aguye, Mubarak G et al. (2020). "Detecting Adversarial Attacks In Time-Series Data". In: *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3092–3096.

Adi, Y. et al. (2019). "To Reverse the Gradient or Not: an Empirical Comparison of Adversarial and Multi-task Learning in Speech Recognition". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3742–3746.

Aggarwal, Charu C (2014). *Data classification: algorithms and applications*. CRC press.

Aghabozorgi, Saeed, Ali Seyed Shirkhorshidi, and Teh Ying Wah (2015). "Time-series clustering–a decade review". In: *Information Systems* 53, pp. 16–38.

Ahmidi, N. et al. (2017). "A Dataset and Benchmarks for Segmentation and Recognition of Gestures in Robotic Surgery". In: *IEEE Transactions on Biomedical Engineering* 64.9, pp. 2025–2041. ISSN: 0018-9294.

Al-Jowder, O., E.K. Kemsley, and R.H. Wilson (1997). "Mid-infrared spectroscopy and authenticity problems in selected meats: a feasibility study". In: *Food Chemistry* 59.2, pp. 195 –201.

Al-Jowder, Osama, EK Kemsley, and Reginald H Wilson (2002). "Detection of adulteration in cooked meat products by mid-infrared spectroscopy". In: *Journal of agricultural and food chemistry* 50.6, pp. 1325–1329.

Anghinoni, L. et al. (2018). "Time Series Trend Detection and Forecasting Using Complex Network Topology Analysis". In: *International Joint Conference on Neural Networks*, pp. 1–7.

Arief-Ang, Irvan B., Flora D. Salim, and Margaret Hamilton (2017). "DA-HOC: Semi-supervised Domain Adaptation for Room Occupancy Prediction Using CO2 Sensor Data". In: *International Conference on Systems for Energy-Efficient Built Environments*, pp. 1–10.

Aswolinskiy, Witali, René Felix Reinhart, and Jochen Steil (2016). "Time Series Classification in Reservoir- and Model-Space: A Comparison". In: *Artificial Neural Networks in Pattern Recognition*, pp. 197–208.

— (2017). "Time Series Classification in Reservoir- and Model-Space". In: *Neural Processing Letters* 48, pp. 789–809.

Bagnall, A. et al. (2015). "Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9, pp. 2522–2535.

Bagnall, A. et al. (2016). "Time-series classification with COTE: The collective of transformation-based ensembles". In: *International Conference on Data Engineering*, pp. 1548–1549.

Bagnall, Anthony and Gareth Janacek (2014). "A Run Length Transformation for Discriminating Between Auto Regressive Time Series". In: *Journal of Classification* 31.2, pp. 154–178.

Bagnall, Anthony et al. (2017). "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery* 31.3, pp. 606–660.

Bagnall, Anthony et al. (2020). "On the Usage and Performance of The Hierarchical Vote Collective of Transformation-based Ensembles version 1.0 (HIVE-COTE 1.0)". In: *ArXiv*.

Bahdanau, D., K. Cho, and Y. Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations*.

Baird, Henry S. (1992). "Document Image Defect Models". In: *Structured Document Image Analysis*. Berlin: Springer, pp. 546–556.

Banerjee, D. et al. (2017). "A Deep Transfer Learning Approach for Improved Post-Traumatic Stress Disorder Diagnosis". In: *IEEE International Conference on Data Mining*, pp. 11–20.

Baydogan, M. G., G. Runger, and E. Tuv (2013). "A Bag-of-Features Framework to Classify Time Series". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11, pp. 2796–2802.

Baydogan, Mustafa Gokce (2015). *Multivariate Time Series Classification Datasets*. http://www.mustafabaydogan.com. Accessed 28 Feb 2019.

Bellman, Richard (2010). *Dynamic Programming*. Princeton University Press.

Benavoli, Alessio, Giorgio Corani, and Francesca Mangili (2016). "Should We Really Use Post-hoc Tests Based on Mean-ranks?" In: *Machine Learning Research* 17.1, pp. 152–161.

Bengio, Yoshua et al. (2011). "Deep Learners Benefit More from Out-of-Distribution Examples". In: *International Conference on Artificial Intelligence and Statistics*. Vol. 15, pp. 164–172.

Bengio, Yoshua et al. (2013). "Generalized Denoising Auto-encoders As Generative Models". In: *International Conference on Neural Information Processing Systems*, pp. 899–907.

Bianchi, F. M. et al. (2018). "Reservoir computing approaches for representation and classification of multivariate time series". In: *ArXiv*. eprint: 1803.07870.

Bishop, Christopher (2006). *Pattern Recognition and Machine Learning*. Springer.

Blázquez-García, Ane et al. (2020). "A review on outlier/anomaly detection in time series data". In: *ArXiv*.

Bostrom, Aaron and Anthony Bagnall (2015). "Binary Shapelet Transform for Multiclass Time Series Classification". In: *Big Data Analytics and Knowledge Discovery*, pp. 257–269.

Boubrahimi, Soukaina Filali, Ruizhe Ma, and Rafal Angryk (2018). "Neuro-ensemble for time series data classification". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 50–59.

Brand, Christian (2016). "Beyond 'Dieselgate': Implications of unaccounted and future air pollutant emissions and energy use for cars in the United Kingdom". In: *Energy Policy* 97, pp. 1–12.

Briandet, Romain, E Katherine Kemsley, and Reginald H Wilson (1996). "Discrimination of Arabica and Robusta in instant coffee by Fourier transform infrared

spectroscopy and chemometrics". In: *Journal of agricultural and food chemistry* 44.1, pp. 170–174.

Bridgewater, Ben et al. (2003). "Surgeon specific mortality in adult cardiac surgery: comparison between crude and risk stratified data". In: *BMJ* 327.7405, pp. 13–17.

Brunel, Anthony et al. (2019). "A CNN adapted to time series for the classification of Supernovae". In: *Electronic Imaging*.

Cabral, F. S. et al. (2018). "An Automatic Survey System for Paved and Unpaved Road Classification and Road Anomaly Detection using Smartphone Sensor". In: *IEEE International Conference on Service Operations and Logistics, and Informatics*, pp. 65–70.

Che, Z. et al. (2017a). "Boosting Deep Learning Risk Prediction with Generative Adversarial Networks for Electronic Health Records". In: *IEEE International Conference on Data Mining*, pp. 787–792.

Che, Zhengping et al. (2017b). "DECADE: A Deep Metric Learning Model for Multivariate Time Series". In: *KDD Workshop on Mining and Learning from Time Series*.

Chen, H et al. (2015). "Model Metric Co-Learning for Time Series Classification." In: *International Joint Conference on Artificial Intelligence*, pp. 3387 –3394.

Chen, Huanhuan et al. (2013). "Model-based Kernel for Efficient Time Series Analysis". In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 392–400.

Chollet, François et al. (2015). *Keras*. https://keras.io.

Choromanska, Anna et al. (2015). "The loss surfaces of multilayer networks". In: *International Conference on Artificial Intelligence and Statistics*, pp. 192–204.

Chouikhi, N., B. Ammar, and A. M. Alimi (2018). "Genesis of Basic and Multi-Layer Echo State Network Recurrent Autoencoders for Efficient Data Representations". In: *ArXiv*. eprint: 1804.08996.

Criss, K. and J. McNames (2011). "Video assessment of finger tapping for Parkinson's disease and other movement disorders". In: *IEEE International Conference on Engineering in Medicine and Biology Society*, pp. 7123–7126.

Cristian Borges Gamboa, J. (2017). "Deep Learning for Time-Series Analysis". In: *ArXiv*.

Csurka, G. (2017). "Domain Adaptation for Visual Applications: A Comprehensive Survey". In: *ArXiv e-prints*.

Cui, Z., W. Chen, and Y. Chen (2016). "Multi-Scale Convolutional Neural Networks for Time Series Classification". In: *ArXiv e-prints*.

Cutur, Marco and Mathieu Blondel (2017). "Soft-DTW: a Differentiable Loss Function for Time-Series". In: *International Conference on Machine Learning*.

Dadhich, S., F. Sandin, and U. Bodin (2018). "Predicting Bucket-Filling Control Actions of a Wheel-Loader Operator Using a Neural Network Ensemble". In: *International Joint Conference on Neural Networks*, pp. 1–6.

Dau, Hoang Anh et al. (2017). "Judicious setting of Dynamic Time Warping's window width allows more accurate classification of time series". In: *IEEE International Conference on Big Data*, pp. 917–922.

Dau, Hoang Anh et al. (2019). "The UCR time series archive". In: *IEEE/CAA Journal of Automatica Sinica* 6.6, 1293–1305. ISSN: 2329-9274. DOI: 10.1109/jas.2019.1911747. URL: http://dx.doi.org/10.1109/JAS.2019.1911747.

Demšar, Janez (2006). "Statistical Comparisons of Classifiers over Multiple Data Sets". In: *Machine Learning Research* 7, pp. 1–30.

Deng, Houtao et al. (2013). "A time series forest for classification and feature extraction". In: *Information Sciences* 239, pp. 142 –153.

Esling, Philippe and Carlos Agon (2012). "Time-series Data Mining". In: *ACM Computing Surveys* 45.1, 12:1–12:34.

Evangelidis, Georgios D and Christian Bauckhage (2011). "Efficient and robust alignment of unsynchronized video sequences". In: *Joint Pattern Recognition Symposium*, pp. 286–295.

Eykholt, Kevin et al. (2018). "Robust Physical-World Attacks on Deep Learning Visual Classification". In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1625–1634.

Faust, Oliver et al. (2018). "Deep learning for healthcare applications based on physiological signals: A review". In: *Computer Methods and Programs in Biomedicine* 161, pp. 1 –13.

Forestier, Germain et al. (2014). "Non-linear temporal scaling of surgical processes". In: *Artificial Intelligence in Medicine* 62.3, pp. 143 –152.

Forestier, Germain et al. (2017a). "Discovering Discriminative and Interpretable Patterns for Surgical Motion Analysis". In: *Artificial Intelligence in Medicine*, pp. 136–145. ISBN: 978-3-319-59758-4.

Forestier, Germain et al. (2017b). "Generating Synthetic Time Series to Augment Sparse Datasets". In: *IEEE International Conference on Data Mining*, pp. 865–870.

Forestier, Germain et al. (2018). "Surgical motion analysis using discriminative interpretable patterns". In: *Artificial Intelligence in Medicine* 91, pp. 3 –11.

Franceschi, Jean-Yves, Aymeric Dieuleveut, and Martin Jaggi (2019). "Unsupervised scalable representation learning for multivariate time series". In: *Advances in Neural Information Processing Systems*, pp. 4650–4661.

Friedman, Milton (1940). "A Comparison of Alternative Tests of Significance for the Problem of *m* Rankings". In: *The Annals of Mathematical Statistics* 11.1, pp. 86–92.

Gallicchio, C. and A. Micheli (2017). "Deep Echo State Network (DeepESN): A Brief Survey". In: *ArXiv*. eprint: 1712.04323.

Gao, Yixin et al. (2014). "The JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS): A Surgical Activity Dataset for Human Motion Modeling". In: *Modeling and Monitoring of Computer Assisted Interventions – MICCAI Workshop*.

Garcia, Salvador and Francisco Herrera (2008). "An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons". In: *Machine learning research* 9, pp. 2677–2694.

Geng, Y. and X. Luo (2018). "Cost-Sensitive Convolution based Neural Networks for Imbalanced Time-Series Classification". In: *ArXiv*. eprint: 1801.04396.

Gharghabi, Shaghayegh et al. (2018). "An ultra-fast time series distance measure to allow data mining in more complex real-world deployments". In: *IEEE International Conference on Data Mining*, pp. 17–20.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *International Conference on Artificial Intelligence and Statistics*. Vol. 9, pp. 249–256.

Gogolou, Anna et al. (2018). "Comparing similarity perception in time series visualizations". In: *IEEE transactions on visualization and computer graphics* 25.1, pp. 523–533.

Goldberg, Yoav (2016). "A Primer on Neural Network Models for Natural Language Processing". In: *Artificial Intelligence Research* 57.1, pp. 345–420.

Gong, Z. et al. (2018). "Multiobjective Learning in the Model Space for Time Series Classification". In: *IEEE Transactions on Cybernetics* 99, pp. 1–15. ISSN: 2168-2267.

Goodfellow, I. J., J. Shlens, and C. Szegedy (2015). "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations*.

Goyal, Manik and Jagath C. Rajapakse (2018). "Deep neural network ensemble by data augmentation and bagging for skin lesion classification". In: *ArXiv*.

Grabocka, Josif et al. (2014). "Learning Time-series Shapelets". In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 392–401.

Hatala, Rose et al. (2015). "Constructing a validity argument for the Objective Structured Assessment of Technical Skills (OSATS): a systematic review of validity evidence". In: *Advances in Health Sciences Education* 20.5, pp. 1149–1175.

Hatami, N., Y. Gavet, and J. Debayle (2017). "Classification of time-series images using deep convolutional neural networks". In: *International Conference on Machine Vision*.

He, K. et al. (2016). "Deep Residual Learning for Image Recognition". In: *IEEE Computer Vision and Pattern Recognition*, pp. 770–778.

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *IEEE International Conference on Computer Vision*, pp. 1026–1034.

Herrera-Almario, Gabriel E. et al. (2016). "The effect of video review of resident laparoscopic surgical skills measured by self- and external assessment". In: *The American Journal of Surgery* 211.2, pp. 315 –320.

Hills, Jon et al. (2014). "Classification of time series by shapelet transformation". In: *Data Mining and Knowledge Discovery* 28.4, pp. 851–881.

Hinton, G. et al. (2012). "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6, pp. 82–97.

Hinton, Geoffrey E and Sam T Roweis (2003). "Stochastic neighbor embedding". In: *Advances in neural information processing systems*, pp. 857–864.

Hoerl, Arthur E. and Robert W. Kennard (1970). "Ridge Regression: Applications to Nonorthogonal Problems". In: *Technometrics* 12.1, pp. 69–82.

Holm, Sture (1979). "A Simple Sequentially Rejective Multiple Test Procedure". In: *Scandinavian Journal of Statistics* 6.2, pp. 65–70.

Höppner, Frank (2016). "Improving time series similarity measures by integrating preprocessing steps". In: *Data Mining and Knowledge Discovery* 31, pp. 851–878.

Hu, Qinghua, Rujia Zhang, and Yucan Zhou (2016). "Transfer learning for short-term wind speed prediction with deep neural networks". In: *Renewable Energy* 85, pp. 83 –95.

Huang, Gao et al. (2017). "Densely connected convolutional networks". In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

Ienco, D. and R. G. Pensa (2018). "Semi-Supervised Clustering With Multiresolution Autoencoders". In: *International Joint Conference on Neural Networks*, pp. 1–8.

Ignatov, Andrey (2018). "Real-time human activity recognition from accelerometer data using Convolutional Neural Networks". In: *Applied Soft Computing* 62, pp. 915 –922.

Intuitive Surgical Sunnyvale, C. A. (2018). *The Da Vinci Surgical System*.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International Conference on Machine Learning*. Vol. 37, pp. 448–456.

Islam, Gazi et al. (2016). "Affordable, web-based surgical skill training and evaluation tool". In: *Journal of Biomedical Informatics* 59, pp. 102 –114. ISSN: 1532-0464.

Ismail Fawaz, H. et al. (2018a). "Evaluating Surgical Skills from Kinematic Data Using Convolutional Neural Networks". In: *Medical Image Computing and Computer Assisted Intervention*. Vol. 11073, pp. 214–221.

Ismail Fawaz, Hassan et al. (2018b). "Data augmentation using synthetic data for time series classification with deep residual networks". In: *International Workshop on Advanced Analytics and Learning on Temporal Data, ECML PKDD*.

— (2018c). "Evaluating surgical skills from kinematic data using convolutional neural networks". In: *International Conference On Medical Image Computing and Computer Assisted Intervention*, pp. 214–221.

— (2018d). "Transfer learning for time series classification". In: *IEEE International Conference on Big Data*, pp. 1367–1376.

— (2019a). "Accurate and interpretable evaluation of surgical skills from kinematic data using fully convolutional neural networks". In: *International Journal of Computer Assisted Radiology and Surgery*.

— (2019b). "Adversarial Attacks on Deep Neural Networks for Time Series Classification". In: *IEEE International Joint Conference on Neural Networks*.

Ismail Fawaz, Hassan et al. (2019c). "Automatic Alignment of Surgical Videos Using Kinematic Data". In: *Artificial Intelligence in Medicine*, pp. 104–113.

Ismail Fawaz, Hassan et al. (2019d). "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery*.

— (2019e). "Deep Neural Network Ensembles for Time Series Classification". In: *IEEE International Joint Conference on Neural Networks*.

Ismail Fawaz, Hassan et al. (2020). "InceptionTime: Finding AlexNet for Time Series Classification". In: *Data Mining and Knowledge Discovery*.

Jaeger, Herbert and Harald Haas (2004). "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication". In: *Science* 304.5667, pp. 78–80.

Jia, Robin and Percy Liang (2017). "Adversarial Examples for Evaluating Reading Comprehension Systems". In: *Empirical Methods in Natural Language Processing*, pp. 2021–2031.

Jin, Lin-peng and Jun Dong (2016). "Ensemble Deep Learning for Biomedical Time Series Classification". In: *Computational Intelligence and Neuroscience* 2016, pp. 13–.

Jing, Longlong and Yingli Tian (2020). "Self-supervised visual feature learning with deep neural networks: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Karimi-Bidhendi, S., F. Munshi, and A. Munshi (2018). "Scalable Classification of Univariate and Multivariate Time Series". In: *IEEE International Conference on Big Data*, pp. 1598–1605.

Kashiparekh, Kathan et al. (2019). "ConvTimeNet: A Pre-trained Deep Convolutional Neural Network for Time Series Classification". In: *IEEE International Joint Conference on Neural Networks*.

Kassahun, Yohannes et al. (2016). "Surgical robotics beyond enhanced dexterity instrumentation: a survey of machine learning techniques and their role in intelligent and autonomous surgical actions". In: *International Journal of Computer Assisted Radiology and Surgery* 11.4, pp. 553–568. ISSN: 1861-6429.

Kasteren, T. L. M. Van, G. Englebienne, and B. J. A. Kröse (2008). "Recognizing activities in multiple contexts using transfer learning". In: *Association for the Advancement of Artificial Intelligence - AI in Elder care*, pp. 142–149.

Kate, Rohit J. (2016). "Using dynamic time warping distances as features for improved time series classification". In: *Data Mining and Knowledge Discovery* 30.2, pp. 283–312.

Keogh, E. et al. (2006). "Intelligent Icons: Integrating Lite-Weight Data Mining and Visualization into GUI Operating Systems". In: *IEEE International Conference on Data Mining*, pp. 912–916.

Keogh, Eamonn and Abdullah Mueen (2017). "Curse of Dimensionality". In: *Encyclopedia of Machine Learning and Data Mining*, pp. 314–315.

Kim, Kyoung-jae (2003). "Financial time series forecasting using support vector machines". In: *Neurocomputing* 55.1-2, pp. 307–319.

Kim, Y. (2014). "Convolutional Neural Networks for Sentence Classification". In: *Empirical Methods in Natural Language Processing*.

Kingma, D. P. and J. Ba (2015). "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations*.

Kneebone, Roger et al. (2002). "An innovative model for teaching and learning clinical procedures". In: *Medical education* 36.7, pp. 628–634.

Kotsifakos, Alexios and Panagiotis Papapetrou (2014). "Model-Based Time Series Classification". In: *Advances in Intelligent Data Analysis*, pp. 179–191.

Krasin, Ivan et al. (2017). "OpenImages: A public dataset for large-scale multi-label and multi-class image classification." In: *Dataset available from https://storage.googleapis.com/openimages/web/index.html*.

Krell, M. M., A. Seeland, and S. K. Kim (2018). "Data Augmentation for Brain-Computer Interfaces: Analysis on Event-Related Potentials Data". In: *ArXiv*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*, pp. 1097–1105.

Kruskal, Joseph B and Myron Wish (1978). *Multidimensional scaling. Number 07–011 in Sage University Paper series on quantitative applications in the social sciences*.

Kurakin, A., I. Goodfellow, and S. Bengio (2017). "Adversarial examples in the physical world". In: *International Conference on Learning Representations - Workshop track*.

Kvamme, Havard et al. (2018). "Predicting mortgage default using convolutional neural networks". In: *Expert Systems with Applications* 102, pp. 207 –217.

Längkvist, Martin, Lars Karlsson, and Amy Loutfi (2014). "A review of unsupervised feature learning and deep learning for time-series modeling". In: *Pattern Recognition Letters* 42, pp. 11 –24.

Large, J., J. Lines, and A. Bagnall (2017). "The Heterogeneous Ensembles of Standard Classification Algorithms (HESCA): the Whole is Greater than the Sum of its Parts". In: *ArXiv*. eprint: 1710.09220.

Le, Quoc and Tomas Mikolov (2014). "Distributed Representations of Sentences and Documents". In: *International Conference on Machine Learning*. Vol. 32, pp. II–1188–II–1196.

Le Guennec, Arthur, Simon Malinowski, and Romain Tavenard (2016). "Data Augmentation for Time Series Classification using Convolutional Neural Networks". In: *ECML/PKDD on Advanced Analytics and Learning on Temporal Data*.

LeCun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 0018-9219.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521, pp. 436–444.

LeCun, Yann et al. (1998). "Efficient BackProp". In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pp. 9–50.

Lee, J., M. Kang, and J. Kang (2017). "Ensemble of binary tree structured deep convolutional network for image classification". In: *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, pp. 1448–1451.

Lee, Wonsung et al. (2018). "Diagnosis Prediction via Medical Context Attention Networks Using Deep Generative Modeling". In: *IEEE International Conference on Data Mining*, pp. 1104–1109.

Li, Zhenqiang et al. (2019). "Manipulation-skill Assessment from Videos with Spatial Attention Network". In: *ArXiv*.

Lin, S. and G. C. Runger (2018). "GCRNN: Group-Constrained Convolutional Recurrent Neural Network". In: *IEEE Transactions on Neural Networks and Learning Systems* 99, pp. 1–10.

Lines, J., S. Taylor, and A. Bagnall (2016). "HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification". In: *IEEE International Conference on Data Mining*, pp. 1041–1046.

Lines, Jason and Anthony Bagnall (2015). "Time series classification with ensembles of elastic distance measures". In: *Data Mining and Knowledge Discovery* 29.3, pp. 565–592.

Lines, Jason, Sarah Taylor, and Anthony Bagnall (2018). "Time Series Classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles". In: *ACM Transactions on Knowledge Discovery from Data* 12.5, 52:1–52:35.

Liu, C., W. Hsaio, and Y. Tu (2018). "Time Series Classification with Multivariate Convolutional Neural Network". In: *IEEE Transactions on Industrial Electronics* 66, pp. 1–1.

Liu, Yongge, Jianzhuang Yu, and Yahong Han (2018). "Understanding the effective receptive field in semantic image segmentation". In: *Multimedia Tools and Applications* 77.17, pp. 22159–22171.

Liul, B. et al. (2018). "Stochastic Multiple Choice Learning for Acoustic Modeling". In: *International Joint Conference on Neural Networks*, pp. 1–6.

Long, Mingsheng et al. (2015). "Learning Transferable Features with Deep Adaptation Networks". In: *International Conference on Machine Learning*. Vol. 37, pp. 97–105.

Lu, J. et al. (2015). "A 1 TOPS/W Analog Deep Machine-Learning Engine With Floating-Gate Storage in 0.13 μm CMOS". In: *IEEE Journal of Solid-State Circuits* 50.1, pp. 270–281.

Lucas, B. et al. (2018). "Proximity Forest: An effective and scalable distance-based classifier for time series". In: *Data Mining and Knowledge Discovery* 28, pp. 851–881.

Luo, Wenjie et al. (2016). "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*, pp. 4898–4906.

Ma, Qianli et al. (2016). "Functional echo state network for time series classification". In: *Information Sciences* 373, pp. 1 –20. ISSN: 0020-0255.

Ma, Tengfei, Cao Xiao, and Fei Wang (2018). "Health-ATM: A Deep Architecture for Multifaceted Patient Health Record Representation and Risk Prediction". In: *SIAM International Conference on Data Mining*, pp. 261–269.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

Maier-Hein, Lena et al. (2017). "Surgical data science for next-generation interventions". In: *Nature Biomedical Engineering* 1.9, pp. 691–696. ISSN: 2157-846X.

Malhotra, P. et al. (2018). "TimeNet: Pre-trained deep recurrent neural network for time series classification". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pp. 607–612.

Marteau, P. (2009). "Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.2, pp. 306–318.

Marteau, Pierre-François (2019). "Times Series Averaging and Denoising from a Probabilistic Perspective on Time–Elastic Kernels". In: *International Journal of Applied Mathematics and Computer Science* 29.2, pp. 375–392.

Martinez, C et al. (2018). "A deep reinforcement learning approach for early classification of time series". In: *European Signal Processing Conference*.

Masic, Izet (2008). "E-learning as new method of medical education". In: *Acta informatica medica* 16.2, p. 102.

Mathis, Florian, Hassan Ismail Fawaz, and Mohamed Khamis (2020). "Knowledge-driven Biometric Authentication in Virtual Reality". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–10.

McNatt, S.S. and C.D. Smith (2001). "A computer-based laparoscopic skills assessment device differentiates experienced from novice laparoscopic surgeons". In: *Surgical Endoscopy* 15.10, pp. 1085–1089.

Means, Barbara et al. (2009). "Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies". In:

Mehdiyev, Nijat et al. (2017). "Time Series Classification using Deep Learning for Process Planning: A Case from the Process Industry". In: *Procedia Computer Science* 114, pp. 242 –249.

Mikolov, T. et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *International Conference on Learning Representations - Workshop*.

Mikolov, Tomas et al. (2013). "Distributed Representations of Words and Phrases and Their Compositionality". In: *Neural Information Processing Systems*, pp. 3111–3119.

Mittelman, R. (2015). "Time-series modeling with undecimated fully convolutional neural networks". In: *ArXiv*. eprint: 1508.00317.

Mota, Paulo et al. (2018). "Video-Based Surgical Learning: Improving Trainee Education and Preparation for Surgery". In: *Journal of surgical education* 75.3, pp. 828–835.

Nawrocka, Agnieszka and Joanna Lamorska (2013). "Determination of food quality by using spectroscopic methods". In: *Advances in agrophysical research*.

Neamtu, Rodica et al. (2018). "Generalized Dynamic Time Warping: Unleashing the Warping Power Hidden in Point-Wise Distances". In: *IEEE International Conference on Data Engineering*.

Newell, Alejandro and Jia Deng (2020). "How Useful is Self-Supervised Pretraining for Visual Tasks?" In: *The IEEE Conference on Computer Vision and Pattern Recognition*.

Nguyen, T. L., S. Gsponer, and G. Ifrim (2017). "Time Series Classification by Sequence Learning in All-Subsequence Space". In: *IEEE International Conference on Data Engineering*, pp. 947–958.

Niitsu, Hiroaki et al. (2013). "Using the Objective Structured Assessment of Technical Skills (OSATS) global rating scale to evaluate the skills of surgical trainees in the operating room". In: *Surgery Today* 43.3, pp. 271–275. ISSN: 1436-2813.

Nwe, T. L., T. H. Dat, and B. Ma (2017). "Convolutional neural network with multi-task learning scheme for acoustic scene classification". In: *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, pp. 1347–1350.

Nweke, Henry Friday et al. (2018). "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges". In: *Expert Systems with Applications* 105, pp. 233 –261.

Olias, Raquel et al. (2006). "Sodium dodecyl sulphate-polyacrylamide gel electrophoresis of proteins in dry-cured hams: Data registration and multivariate analysis across multiple gels". In: *Electrophoresis* 27.7, pp. 1288–1299.

Ordóñez, Francisco Javier and Daniel Roggen (2016). "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition". In: *Sensors* 16, p. 115.

Oregi, Izaskun et al. (2018). "Adversarial Sample Crafting for Time Series Classification with Elastic Similarity Measures". In: *International Symposium on Intelligent and Distributed Computing*, pp. 26–39.

Owen, Paula and R Foreman (2012). "Powering the nation: Household electricity using habits revealed". In: *Energy Saving Trust/DECC/DEFRA*.

Padua, Flavio et al. (2010). "Linear sequence-to-sequence alignment". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.2, pp. 304–320.

Pan, S. J. and Q. Yang (2010). "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10, pp. 1345–1359.

Pan, Weike et al. (2011). "Transfer Learning to Predict Missing Ratings via Heterogeneous User Feedbacks". In: *International Joint Conference on Artificial Intelligence*, pp. 2318–2323.

Papernot, N. and P. McDaniel (2018). "Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning". In: *ArXiv*. eprint: 1803.04765.

Papernot, Nicolas et al. (2018). "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library". In: *ArXiv*.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2012). "Understanding the exploding gradient problem". In: *ArXiv*. eprint: 1211.5063.

— (2013). "On the Difficulty of Training Recurrent Neural Networks". In: *International Conference on Machine Learning*. Vol. 28, pp. III–1310–III–1318.

Pelletier, Charlotte, Geoffrey I Webb, and François Petitjean (2019). "Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series". In: *Remote Sensing* 11.5, p. 523.

Petitjean, François et al. (2014). "Dynamic time warping averaging of time series allows faster and more accurate classification". In: *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, pp. 470–479.

— (2016). "Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm". In: *Knowledge and Information Systems* 47.1, pp. 1–26.

Petitjean, François and Pierre Gançarski (2012). "Summarizing a set of time series by averaging: From Steiner sequence to compact multiple alignment". In: *Theoretical Computer Science* 414.1, pp. 76 –91.

Petitjean, François, Alain Ketterlin, and Pierre Gançarski (2011). "A global averaging method for dynamic time warping, with applications to clustering". In: *Pattern Recognition* 44.3, pp. 678 –693.

Poggio, Tomaso et al. (2017). "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review". In: *International Journal of Automation and Computing* 14.5, pp. 503–519.

Polavarapu, Harsha V et al. (2013). "100 years of surgical education: the past, present, and future." In: *Bulletin of the American College of Surgeons* 98.7, pp. 22–29.

Pravilovic, Sonja et al. (2014). "Wind power forecasting using time series cluster analysis". In: *International Conference on Discovery Science*. Springer, pp. 276–287.

Qiu, Z. and X. Gu (2018). "Multi ROI and Multi Map Networks for Accurate and Efficient Pedestrian Detection". In: *International Joint Conference on Neural Networks*, pp. 1–7.

Rabatel, Julien, Sandra Bringay, and Pascal Poncelet (2011). "Anomaly detection in monitoring sensor data for preventive maintenance". In: *Expert Systems with Applications* 38.6, pp. 7003–7015.

Rajan, D and JJ Thiagarajan (2018). "A Generative Modeling Approach to Limited Channel ECG Classification." In: *IEEE Engineering in Medicine and Biology Society*. Vol. 2018, p. 2571.

Rakhshani, Hojjat et al. (2019). "Neural Architecture Search for Time Series Classification". In: *IEEE International Joint Conference on Neural Networks*.

Rapp, Allison K et al. (2016). "YouTube is the most frequently used educational video source for surgical preparation". In: *Journal of Surgical Education* 73.6, pp. 1072–1076.

Ratanamahatana, Chotirat Ann and Eamonn Keogh (2005). "Three Myths about Dynamic Time Warping Data Mining". In: *SIAM International Conference on Data Mining*, pp. 506–510.

Rowe, Alistair C. H. and Paul C. Abbott (1995). "Daubechies wavelets and Mathematica". In: *Computers in Physics* 9.6, pp. 635–648.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3, pp. 211–252.

Sainath, T. N. et al. (2013). "Deep convolutional neural networks for LVCSR". In: *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8614–8618.

Sakoe, H. and S. Chiba (1978). "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1, pp. 43–49.

Santos, Tiago and Roman Kern (2017). "A Literature Survey of Early Time Series Classification and Deep Learning". In: *International Conference on Knowledge Technologies and Data-driven Business*.

Sanwlani, Monica and M Vijayalakshmi (2013). "Forecasting sales through time series clustering". In: *International Journal of Data Mining & Knowledge Management Process* 3.1, p. 39.

Scardapane, Simone and Dianhui Wang (2017). "Randomness in neural networks: an overview". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7.2, e1200.

Schäfer, Patrick (2015). "The BOSS is concerned with time series classification in the presence of noise". In: *Data Mining and Knowledge Discovery* 29.6, pp. 1505–1530.

Serrà, J., S. Pascual, and A. Karatzoglou (2018). "Towards a universal neural network encoder for time series". In: *Artificial Intelligence Research and Development: Current Challenges, New Trends and Applications* 308, p. 120.

Shokoohi-Yekta, Mohammad et al. (2017). "Generalizing DTW to the multi-dimensional case requires an adaptive approach". In: *Data Mining and Knowledge Discovery* 31.1, pp. 1–31.

Silva, Diego F. et al. (2018). "Speeding up similarity search under dynamic time warping by pruning unpromising alignments". In: *Data Mining and Knowledge Discovery* 32.4, pp. 988–1016.

Smith, Todd L and Scot Ransbottom (2000). "Digital video in education". In: *Distance learning technologies: Issues, trends and opportunities*, pp. 124–142.

Spiegel, Stephan (2016). "Transfer Learning for Time Series Classification in Dissimilarity Spaces". In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958.

Stefan, A., V. Athitsos, and G. Das (2013). "The Move-Split-Merge Metric for Time Series". In: *IEEE Transactions on Knowledge and Data Engineering* 25.6, pp. 1425–1438.

Strodthoff, Nils and Claas Strodthoff (2019). "Detecting and interpreting myocardial infarction using fully convolutional neural networks". In: *Physiological Measurement* 40.1, p. 015001.

Susto, Gian Antonio, Angelo Cenedese, and Matteo Terzi (2018). "Time-Series Classification Methods: Review and Applications to Power Systems Data". In: *Big Data Application in Power Systems*, pp. 179 –220.

Susto, Gian Antonio et al. (2015). "Machine learning for predictive maintenance: A multiple classifier approach". In: *IEEE Transactions on Industrial Informatics* 11.3, pp. 812–820.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *Neural Information Processing Systems*, pp. 3104–3112.

Szegedy, C. et al. (2014). "Intriguing properties of neural networks". In: *ArXiv*.

Szegedy, C. et al. (2015). "Going deeper with convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

Szegedy, Christian et al. (2017). "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *AAAI Conference on Artificial Intelligence*.

Tan, Chang Wei, Geoffrey I Webb, and François Petitjean (2017). "Indexing and classifying gigabytes of time series under time warping". In: *SIAM International Conference on Data Mining*, pp. 282–290.

Tan, Chang Wei et al. (2018). "Efficient search of the best warping window for Dynamic Time Warping". In: *SIAM International Conference on Data Mining*, pp. 225–233.

Tanisaro, P. and G. Heidemann (2016). "Time Series Classification Using Time Warping Invariant Echo State Networks". In: *IEEE International Conference on Machine Learning and Applications*, pp. 831–836.

Tao, Lingling et al. (2012). "Sparse Hidden Markov Models for Surgical Gesture Classification and Skill Evaluation". In: *Information Processing in Computer-Assisted Interventions*, pp. 167–177. ISBN: 978-3-642-30618-1.

Taylor, James W, Patrick E McSharry, and Roberto Buizza (2009). "Wind power density forecasting using ensemble predictions and time series models". In: *IEEE Transactions on Energy Conversion* 24.3, pp. 775–782.

Tobiyama, S. et al. (2016). "Malware Detection with Deep Neural Network Using Process Behavior". In: *IEEE Annual Computer Software and Applications Conference*, pp. 577–582.

Tripathy, R.K. and U. Rajendra Acharya (2018). "Use of features from RR-time series and EEG signals for automated classification of sleep stages in deep neural network framework". In: *Biocybernetics and Biomedical Engineering* 38, pp. 890–902.

Tsipras, Dimitris et al. (2018). "There is no free lunch in adversarial robustness (but there are unexpected benefits)". In: *ArXiv* 8.

Uemura, Munenori et al. (2018). "Feasibility of an AI-Based Measure of the Hand Motions of Expert and Novice Surgeons". In: *Computational and Mathematical Methods in Medicine* 2018.

Ulyanov, D., A. Vedaldi, and V. Lempitsky (2016). "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *ArXiv*. eprint: 1607.08022.

Um, Terry T. et al. (2017). "Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring Using Convolutional Neural Networks". In: *ACM International Conference on Multimodal Interaction*, pp. 216–220.

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Vedula, S. Swaroop et al. (2016). "Analysis of the Structure of Surgical Activity for a Suturing and Knot-Tying Task". In: *Public Library of Science One* 11.3, pp. 1–14.

Vercruyssen, Vincent, Wannes Meert, and Jesse Davis (2017). "Transfer Learning for Time Series Anomaly Detection". In: *Workshop and Tutorial on Interactive Adaptive Learning co-located with European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pp. 27–36.

Wang, J. et al. (2018). "Multilevel Wavelet Decomposition Network for Interpretable Time Series Analysis". In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2437–2446.

Wang, Jindong et al. (2018). "Deep learning for sensor-based activity recognition: A Survey". In: *Pattern Recognition Letters*.

Wang, Lin, Zhigang Wang, and Shan Liu (2016). "An effective multivariate time series classification approach using echo state network and adaptive differential evolution algorithm". In: *Expert Systems with Applications* 43, pp. 237 –249.

Wang, Lusheng and Tao Jiang (1994). "On the Complexity of Multiple Sequence Alignment". In: *Journal of Computational Biology* 1.4, pp. 337–348.

Wang, Oliver et al. (2014). "Videosnapping: Interactive synchronization of multiple videos". In: *ACM Transactions on Graphics* 33.4, p. 77.

Wang, Shuqin et al. (2017). "A Cycle Deep Belief Network Model for Multivariate Time Series Classification". In: *Mathematical Problems in Engineering* 2017, pp. 1–7.

Wang, W. et al. (2016a). "Earliness-Aware Deep Convolutional Networks for Early Time Series Classification". In: *ArXiv*. eprint: 1611.04578.

Wang, X., C. Li, and B. Xu (2018). "Hierarchical Tree Long Short-Term Memory for Sentence Representations". In: *International Joint Conference on Neural Networks*, pp. 1–6.

Wang, Z. and T. Oates (2015). "Spatially Encoding Temporal Correlations to Classify Temporal Data Using Convolutional Neural Networks". In: *ArXiv*. eprint: 1509.07481.

Wang, Z., W. Yan, and T. Oates (2017). "Time series classification from scratch with deep neural networks: A strong baseline". In: *International Joint Conference on Neural Networks*, pp. 1578–1585.

Wang, Z. et al. (2016b). "Representation Learning with Deconvolution for Multivariate Time Series Classification and Visualization". In: *ArXiv*. eprint: 1610.07258.

Wang, Zhiguang and Tim Oates (2015a). "Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks". In: *Workshops at AAAI Conference on Artificial Intelligence*, pp. 40–46.

— (2015b). "Imaging Time-series to Improve Classification and Imputation". In: *International Conference on Artificial Intelligence*, pp. 3939–3945.

Wang, Ziheng and Ann Majewicz Fey (2018). "Deep learning with convolutional neural network for objective skill evaluation in robot-assisted surgery". In: *International Journal of Computer Assisted Radiology and Surgery* 13.12, pp. 1959–1970.

Weber, Ron A Shapira et al. (2019). "Diffeomorphic Temporal Alignment Nets". In: *Advances in Neural Information Processing Systems*, pp. 6574–6585.

Wedge, Daniel, Peter Kovesi, and Du Huynh (2005). "Trajectory based video sequence synchronization". In: *Digital Image Computing: Techniques and Applications*, pp. 13–13.

Weiss, Karl, Taghi M. Khoshgoftaar, and DingDing Wang (2016). "A survey of transfer learning". In: *Journal of Big Data* 3.1, p. 9.

Wen, Guihua et al. (2017). "Ensemble of Deep Neural Networks with Probability-Based Fusion for Facial Expression Recognition". In: *Cognitive Computation* 9.5, pp. 597–610.

Wilcoxon, Frank (1945). "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6, pp. 80–83.

Wolf, Lior and Assaf Zomet (2002). "Sequence-to-sequence self calibration". In: *European Conference on Computer Vision*, pp. 370–382.

Wolpert, David H, William G Macready, et al. (1995). *No free lunch theorems for search*. Tech. rep. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Xi, R. et al. (2018). "Deep Dilated Convolution on Multimodality Time Series for Human Activity Recognition". In: *International Joint Conference on Neural Networks*, pp. 1–8.

Xie, Cihang et al. (2019). "Feature denoising for improving adversarial robustness". In: *The IEEE Conference on Computer Vision and Pattern Recognition*, pp. 501–509.

Xie, Cihang et al. (2020). "Adversarial Examples Improve Image Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xu, N., G. Chen, and W. Mao (2018). "MNRD: A Merged Neural Model For Rumor Detection In Social Media". In: *International Joint Conference on Neural Networks*, pp. 1–7.

Yamada, Yasunori and Masatomo Kobayashi (2017). "Detecting Mental Fatigue from Eye-Tracking Data Gathered While Watching Video". In: *Artificial Intelligence in Medicine*, pp. 295–304.

Yang, Min et al. (2018). "Investigating Deep Reinforcement Learning Techniques in Personalized Dialogue Generation". In: *SIAM International Conference on Data Mining*, pp. 630–638.

Yang, Qiang and Xindong Wu (2006). "10 CHALLENGING PROBLEMS IN DATA MINING RESEARCH". In: *Information Technology & Decision Making* 05.04, pp. 597–604.

Ye, Lexiang and Eamonn Keogh (2009). "Time Series Shapelets: A New Primitive for Data Mining". In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 947–956.

— (2011). "Time series shapelets: a novel technique that allows accurate, interpretable and fast classification". In: *Data Mining and Knowledge Discovery* 22.1, pp. 149–182.

Yi, F. et al. (2018). "An Integrated Model for Crime Prediction Using Temporal and Spatial Factors". In: *IEEE International Conference on Data Mining*, pp. 1386–1391.

Yosinski, Jason et al. (2014a). "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*, pp. 3320–3328.

— (2014b). "How Transferable Are Features in Deep Neural Networks?" In: *International Conference on Neural Information Processing Systems*. Vol. 2, pp. 3320–3328.

Yuan, X. et al. (2017). "Adversarial Examples: Attacks and Defenses for Deep Learning". In: *ArXiv*.

Yuan, Ye et al. (2018). "Muvan: A multi-view attention network for multivariate temporal data". In: *IEEE International Conference on Data Mining*, pp. 717–726.

Zeiler, M. D. (2012). "ADADELTA: An Adaptive Learning Rate Method". In: *ArXiv*. eprint: 1212.5701.

Zhang, Chiyuan et al. (2017). "Understanding deep learning requires rethinking generalization". In: *International Conference on Learning Representations*.

Zhao, B. et al. (2017). "Convolutional neural networks for time series classification". In: *Systems Engineering and Electronics* 28.1, pp. 162–169.

Zheng, Yi et al. (2014). "Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks". In: *Web-Age Information Management*, pp. 298–310.

— (2016). "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification". In: *Frontiers of Computer Science* 10.1, pp. 96–112. ISSN: 2095-2236.

Zheng, Z. et al. (2018). "Wide and Deep Convolutional Neural Networks for Electricity-Theft Detection to Secure Smart Grids". In: *IEEE Transactions on Industrial Informatics* 14.4, pp. 1606–1615.

Zhou, B. et al. (2016). "Learning Deep Features for Discriminative Localization". In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2921–2929.

Zia, Aneeq and Irfan Essa (2018). "Automated surgical skill assessment in RMIS training". In: *International Journal of Computer Assisted Radiology and Surgery* 13.5, pp. 731–739.

Ziat, A. et al. (2017). "Spatio-Temporal Neural Networks for Space-Time Series Forecasting and Relations Discovery". In: *IEEE International Conference on Data Mining*, pp. 705–714.

Zoumpatianos, Kostas, Stratos Idreos, and Themis Palpanas (2014). "Indexing for interactive exploration of big data series". In: *ACM SIGMOD International Conference on Management of Data*, pp. 1555–1566.