Deep learning for spatio-temporal multidimensional signals : an application to transport mode detection

Hugues Moreau

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LYON

Opérée au sein de :

l'École Centrale de Lyon

Ecole Doctorale 512
InfoMath

Spécialité de doctorat : Informatique

Soutenue publiquement le 17/12/2021, par :

Hugues Ivan Louis MOREAU

Deep Learning for Temporal Multidimensional Signals - an application to Transport Mode Detection

Devant le jury composé de :

Liming Luke CHEN Professeur, Université d'Ulster	**Président**
Hongying MENG Professeut, Université de Brunel	**Rapporteur**
Xi ZHAO Professeur, Université de Jiaotong de Xi'An	**Rapporteur**
Alice OTHMANI Maître de conférences, Université Paris Est	**Examinateur**
Marielle MALFANTE Ingénieur chercheur, CEA LIST	**Examinateur**
Liming CHEN Profeseur, École Centrale de Lyon	**Directeur de thèse**
Andréa VASSILEV Ingénieur de recherche, CEA LETI	**Invité**

Numéro d'ordre NNT : 2021LYSEC51

Acknowledgements

I never really understood the heartfelt messages that usually open the manuscripts I read. From outside, it felt pretended, as if it was an impersonal list of tributes a PhD candidate owed to their colleagues. I never saw how wrong I was, until I had to reflect to my own work in its entirety. I realize that a thesis is never a single person's work, and how much the presence and personalities of my colleagues taught me.

Please know that my gratitude is not feigned. For the most part.

Thank you Andrea, for your critical eye. Without you I would not be half as rigorous as I am today. Thank you also for your availability and your sincere willingness to help.
Thank you Liming for helping me to take a step back on my work, and for guiding me through the arcane of academics.
Thank you Viviane, for being so understanding. I did appreciate working with you.
Thank you Christelle, for helping correcting the trajectory of the thesis, and avoiding what could have been a disappointing descent into mediocrity.
Thank you to my immediate colleagues for the multiple interesting technical discussions and insights about the problems I had to solve: Thibault, Alex, Nicolas, Jérôme, Adrien B., Thomas; and in particular to Dimitri, Adrien V., and Régis, for helping me proofread this manuscript. Thank you to all my colleagues in CEA in general, for their warm welcome and their helpfulness.

Thank you all.

Abstract

Transport Mode Detection (TMD) is a classification problem where the goal is to infer the transport mode of a user from GPS signals or inertial sensors, with applications such as carbon footprint tracking, mobility behaviour analysis, or real-time door-to-door smart planning. Traditionally, the method for solving this problem involved training a Machine Learning classifier on handcrafted features.

In this thesis, we will tackle Transport Mode Detection using Deep Neural Networks, a class of algorithms which offers the possibility to learn the features automatically from data. By attempting to use Deep Learning on TMD, we will tackle several different research questions: firstly, whether to preprocess the signals by computing a spectrogram, or stick to a one-dimensional sequence. We will show that computing a spectrogram does simplify the problem, thereby helping the network not to overfit on a simple problem. The second question to answer is data fusion, or, how to merge the data from different sensors. We propose a benchmark of different data fusion methods used with Deep Neural Networks and conclude that no method outperforms the others. Lastly, we will focus on Canonical Correlation Analysis to show that, when it is applied to features of deep neural networks, the canonical components are equal to the classification components of the network.

Résumé

La Détectiuon du Mode de Transport (Transport Mode Detection, TMD) est un problème de classification dont le but est de déterminer le mode de transport emprunté par un utilisateur à partir de signaux GPS ou de capteurs inertiels, et dont les applications vont de l'estimation d'empreinte carbone à l'analyse de comportement de déplacements, en passant par la planification d'itinéraire en temps réel. Traditionnellement, la résolution de ce problème passait par l'entraînement d'un classifieur avec des descripteurs calculés en fonction de connaissances pré-établies du domaine.

Dans cette thèse, nous nous attaquerons au problème de la Détection du Mode de Transport à l'aide de réseaux de neurones profonds, un type d'algorithme qui apprend à calculer les descripteurs les plus adaptés au problème à résoudre. Ce faisant, nous rencontrerons plusieurs questions de recherche : d'abord, nous chercherons à savoir s'il nous faut passer par un spectrogramme, ou si le réseau peut traiter les signaux bruts. Nous montrerons que, dans notre cas, calculer un spectrogramme permet de simplifier le problème de classification à résoudre, ce qui évite au réseau de surapprendre. La deuxième question à résoudre est de savoir comment intégrer les informations en provenance de différents capteurs, un problème appelé fusion de données. Nous proposons une évaluation de différents algorithmes de fusion de données par réseaux de neurones et concluons que, pour notre problème, aucune méthode ne dépasse significativement les autres. Enfin, nous nous intéresserons à une opération appelée Analyse des Corrélations Canoniques (Canonical Correlation Analysis, CCA) pour montrer que, lorsqu'on l'applique aux descripteurs appris par des réseaux de neurones, les composantes canoniques sont égales aux composantes de classification.

Contents

1 Introduction **1**
- 1.1 Context and Motivation . . . 1
  - 1.1.1 Temporal signals . . . 1
  - 1.1.2 A brief overview of recent Computer Vision history . . . 1
  - 1.1.3 A collaboration with an applied research laboratory . . . 2
- 1.2 The problem of Transport Mode Detection and its application . . . 3
- 1.3 The proposed approach and contributions . . . 4
- 1.4 Outline of the manuscript . . . 4

2 Related works, datasets, and baseline **6**
- 2.1 State of the art . . . 6
  - 2.1.1 Data cleaning . . . 6
  - 2.1.2 Point-level feature computation . . . 8
  - 2.1.3 Segmentation . . . 9
  - 2.1.4 Trajectory-level feature computation . . . 9
  - 2.1.5 Classification: classical Machine Learning . . . 10
  - 2.1.6 Classification: Deep Learning . . . 10
  - 2.1.7 The problem of the evaluation . . . 12
- 2.2 Datasets . . . 14
  - 2.2.1 The GeoLife dataset . . . 14
  - 2.2.2 The SHL 2018 challenge . . . 17
  - 2.2.3 SHL 2019 and 2020 challenges . . . 17
  - 2.2.4 The TMD Dataset . . . 17
- 2.3 Baselines . . . 19
  - 2.3.1 Transport Mode Detection as a classification problem . . . 20
  - 2.3.2 GeoLife Baseline . . . 21
  - 2.3.3 SHL Baseline . . . 27
- 2.4 Conclusion . . . 35

3 Preprocessing **36**
- 3.1 Introductory example: How padding segments can disturb the learning process . . . 36
- 3.2 An overview of preprocessing in the literature . . . 39
  - 3.2.1 Audio processing . . . 40
  - 3.2.2 Failure prediction in rotating machines . . . 41
  - 3.2.3 Physiological signals . . . 42
  - 3.2.4 Human Activity Recognition . . . 43
  - 3.2.5 Transport Mode Detection . . . 45
  - 3.2.6 Conclusion of the literature study . . . 46
- 3.3 Evaluation of preprocessing methods . . . 48
- 3.4 Understanding why spectrograms are more effective . . . 52
  - 3.4.1 Which frequencies are useful for classification ? . . . 52
  - 3.4.2 Computing the average of gradients . . . 53

3.5 Conclusion . . . 62

4 Global Pooling **63**
4.1 The different types of global pooling . . . 63
4.2 Evaluation metrics . . . 64
4.3 Results . . . 65
4.3.1 Comparison of the alternatives to the flatten step . . . 65
4.3.2 Comparison with the state of the art . . . 66
4.4 Conclusion . . . 68

5 Data fusion **69**
5.1 Data Fusion modes in deep learning . . . 69
5.2 An inventory of fusion modes . . . 70
5.2.1 Early fusion . . . 70
5.2.2 Intermediate fusion . . . 71
5.2.3 Late fusion . . . 76
5.3 A benchmark of fusion modes . . . 78
5.4 Decorrelated networks . . . 81
5.4.1 Principle . . . 81
5.4.2 Experimental protocol . . . 82
5.4.3 Results . . . 85
5.4.4 Why did the decorrelation loss not help . . . 86
5.5 Evaluations on the test set . . . 87
5.6 Conclusion . . . 87

6 A study on Canonical Correlation Analysis **88**
6.1 Introduction . . . 89
6.1.1 Notations and definitions . . . 89
6.1.2 Deep features . . . 90
6.1.3 A general presentation of Canonical Correlation Analysis . . . 91
6.2 Related works . . . 92
6.3 An application of canonical correlation analysis to deep features . . . 93
6.3.1 Measuring the canonical correlations to quantify the similarity between sensors . . . 94
6.3.2 Demonstrate that the network keeps the power . . . 95
6.3.3 Influence of the network initialization . . . 97
6.4 The equality between the first canonical components and the class components . . . 98
6.4.1 Introduction: A glance at canonical variables . . . 100
6.4.2 Projection experiments . . . 102
6.4.3 An explicit measurement of subspace similarity . . . 106
6.4.4 Partial conclusion: the proximity between class and canonical components . . . 116
6.5 The causes of the equality . . . 117
6.6 How the equality between class and canonical components implies that a CCA fusion is ineffective 122
6.7 An implementation of CCA fusion with SHL . . . 122
6.8 Varying the layer where features are extracted . . . 124
6.9 Conclusion . . . 125

7 Conclusion **128**
7.1 Summary of the contributions . . . 128
7.1.1 Preprocessing of input segments . . . 128
7.1.2 Global Pooling methods . . . 128
7.1.3 Data Fusion . . . 129
7.1.4 Canonical Correlation Analysis for data fusion . . . 129
7.2 Future work . . . 129
7.2.1 Semi, self, or unsupervised learning . . . 129
7.2.2 Domain Adaptation . . . 130

A Methodology: Computing a F1 score from an Intersection over Union **156**

B Automatic sensor selection **158**
B.1 Data fusion for sensor selection 158
B.2 Three known scenarios 158
B.3 Conclusion 160

C Random Search for hyperparameters on the GeoLife dataset **161**
C.1 Experimental setup 161
C.2 Results 162
C.3 Influence of the architecture 166
C.4 Conclusion 168

D The curious behaviour of the spectrogram of the orientation **169**
D.1 An observation: irregular learning of the network 169
D.2 Why does the network learn so irregularly ? 170
D.3 Why does the behaviour depend so much on the random seed ? 170
D.4 Why are we even talking about this ? 171

List of Figures

2.1 The outline of a classic Transport Mode Detection algorithms, along with a few examples of each step. . . . 7
2.2 An illustration of the meaning of the axes in a smartphone: the referential of the sensor (x_b, y_b, z_b) and the de-rotated, or NED (*North, East, Down*) coordinate system. Reproduced from [54]. . . . 8
2.3 A few examples of monomodal portions of trajectories in the GeoLife dataset. The start of the trajectory is set to be the origin . . . 14
2.4 The histogram of all the orientations between two successive GPS points in the GeoLife dataset. The fact that the 0°, 90°, 180°, and 270° values are over-represented indicate that most of the trajectories follow the orientation of the grid-like road system. . . . 15
2.5 The histogram of the durations of each monomodal segment in the GeoLife dataset. As expected by the intuition, the Walk segments are typically quite short (less than 15 minutes), while the train segments can be extremely long (more than three hours). . . . 15
2.6 Some example of signals from the SHL 2018 dataset, for each of the 8 classes and for three sensors: a) norm of the accelerometer b) norm of the magnetometer c) barometric pressure . 18
2.7 A scatter plot of the samples from the accelerometer measurements of the TMD dataset, depending on their number of points and total duration. The diagonal lines illustrate the iso-frequencies (average frequency in the sample), in Hz. . . . 19
2.8 The outline of our methodology. As we cannot explore the whole search space, we simply explore along a few chosen directions, changing one type of option at a time. . . . 20
2.9 The separation of a trajectory with the GeoLife dataset. Each trip is composed of triplegs; each tripleg correspond to a single transport mode. When two consecutive points are distant more than a certain threshold (chosen to be equal to 20 minutes in our case), we break it into two neighboring trips. . . . 21
2.10 An overview of 200 distributions obtained by splitting by users. Depending on the realizations, the proportion of each class can vary greatly: in the validation and test sets, the first quartile is always at least twice lower than the third quartile. In extreme cases, one set can completely lack one class. . . . 23
2.11 The architecture of the GeoLife baseline model . . . 24
2.12 The confusion matrix of the GeoLife model, on the test set. We can see that there is a high confusion between the classes "Car&taxi" and "Bus": merging together these classes will improve the performance considerably. . . . 27
2.13 The cumulative proportion of each class versus time. The class proportion is computed on a moving window of 1,500 samples, with a stride of 50. We can see the end of the dataset is devoid of Car segments. . . . 28
2.14 The histogram of the classes in each set (lines) for each splitting (columns). The train/validation split made by the organizers of the challenge is balanced, but it does not take the chronological order into account. A train/val split does, but the train set lacks run segments. A val/train split of the samples is both rigorous and balanced enough. As the test set is already split by the organizers of the challenge, its content does not depend on the splitting. . . . 28

2.15 An illustration of the preprocessing step with the norm of the accelerometer data from a running segment. The 2.3 Hz frequency band appears in the middle of the spectrogram due to the log scale for the frequencies. This 2.3 Hz frequency is approximately the frequency at which one foot touches the ground. The use of the log-energy on the bottom right-hand corner allows to better displays the 1.15 Hz band, which is the period of the right leg movement (with the SHL 2018 dataset, the phone is kept in the right pocket of the user). 29
2.16 The architecture of the baseline SHL model 30
2.17 A bar plot of each individual sensors corresponding to the results in table 2.5. The error bars denote the standard deviation over five random initializations of the network. 31
2.18 An histogram displaying the values of the three axes of the Gravity sensor. We can see that the phone is upright when the user walks (the extreme values in the negative region of the y axis are mainly from walk segments); the phone lies flat, with the screen facing up or down, when the user drives (the car segments are extreme values of the z-axis, both positive and negative); and that the phone is not often on the side (the extreme values of the x-axis are not the most represented). See fig. 2.2 for an illustration of the meaning of the axes. Note that the phone was in the user's pocket when the dataset was recorded. 32
2.19 An example of signal that shows how the slight noise in the norm of the gravity discloses the class for the most obvious modes (here, a Bike segment). 32
2.20 The computation of the orientation quaternion. 33
2.21 The confusion matrices of models using (a) the norm of the accelerometer (b) the y axis of the gyrometer (c) the norm of the magnetometer (d) the w axis of the orientation quaternion, with the SHL validation dataset. 34

3.1 The different kinds of padding. Zero-padding simply adds zeros until the maximum length is reached, Reflection reverses a copy of the segment and adds it at the end, while wrapping simply duplicates the segment until the maximum length is reached. 37
3.2 The validation *accuracy* versus the size of each segment (the shorter the segment, the more zeros it will be padded with). Adding zeros is not particularly detrimental to the classification performance, which means the network learnt to ignore the zeros, missing potentially relevant information. Intervals bins are obtained using equidistant separations between 0 and the 90-th percentile 38
3.3 The main hypothesis we want to verify ([152]): leaving the data intact is worse when the number of samples is low, and better when the dataset is large. 39
3.4 The type of preprocessing used, depending on the total duration, number of samples, or number of subjects of the dataset (N/M: Not Mentioned; combination denotes the use of several types of features). Figure generated using the code and data from [196]. We would like to thank Y. Roy for providing us access to the code. 43
3.5 A bar plot representation of table 3.4. Best view in color. 49
3.6 The average power spectrum per class (only half of the spectrum is shown). Note the closeness of the fundamentals for the Walk, Run, and Bike classes, the Dirac comb shape of the Run spectra, and the sharp components at 21, 25, and 30 Hz for Bus signals (corresponding respectively to 1260, 1500 and 1800 rpm, the usual rotation speeds of an engine). Best view in color 50
3.7 An example of discontinuity. Note how the periodic components at the end of the segment leave patterns that remain noticeable with the log-power, and not with the raw power. 51
3.8 An illustration of the data degradation process: we set eight consecutive lines to zero in the 48×48 spectrograms. 52
3.9 The F1 per class of a network that was trained on clean (a) or degraded (b) data and evaluated on degraded data, depending on the frequency band removed by the degradation. The width of the curves denote the standard deviation across five random initializations. The log scale for the frequencies make the intervals uneven when expressed in Hertz, while they had the same size (eight pixels) on the spectrogram. 53
3.10 the reason why we compute the gradient of a log-probability: for many samples, the gradient of the probability are too low to account for in an average. 54

3.11 An example of saliency map with a single Run segment. 55
3.12 An illustration with artificial data of what we want be careful to when averaging the gradients: summing different versions of the same motif, at different time steps, might destroy it. 55
3.13 An illustration of the axis shuffling with a spectrogram from a Run segment. On top of each spectrogram is the name of the shuffled axis 56
3.14 The gradient of each class. 57
3.15 A focus on some gradients from figure 3.14. 58
3.16 The histogram of the predictions on the validation set after we added the gradient for a given class a certain amount of times. To display this graph, we re-weighted the samples so that the bars appear balanced when the dataset is untouched. 59
3.17 the three experiments we will lead to show the network classifies some classes linearly. 60
3.18 The results of the three experiments described in fig. 3.17. The first experiment (a) shows the Running segments can be classified linearly with little error; while the two others demonstrate that the network actually behaves linearly. 61

4.1 Why a global pooling method allows a network to process inputs with different shapes. This idea originally comes from Computer Vision ([142]) , but was unknown in TMD. 64

5.1 the three early concatenations 71
5.2 An overview of the difference between frequency and time concatenations, and feature concatenation. As the features learnt in the final feature map still retain some spatial consistency (a), a network using frequency and time concatenations (b) can still distinguish between the features from each sensor. The main difference with feature concatenation (c) is that the network can now learn sensor-specific convolution filters, which was impossible with the two early fusion methods. 73
5.3 A sanity check of our attention mechanism. We can see that, when a network was trained with obstructed samples, it learns to ignore constant portions of the spectrogram by assigning them no attention 75
5.4 An illustration of the architectures of the baseline attention (5.4a) and the selective fusion (5.4b). 76
5.5 The mean and standard deviation of different late fusion methods, as a function of the proportion of epochs where the sensor-specific models are learnt separately. 78
5.6 A naive way to force the features to be complementary. 81
5.7 An illustration of why we cannot use a simple L1 norm to separate the features of two bases. The same point P has two coordinates $X_1 = (x_1, y_1, z_1)$ and $X_2 = (x_2, y_2, z_2)$ in the red and blue coordinate systems (respectively), and the corresponding features do not match (*e.g.* $x_1 \neq x_2$). Note that in this illustration, the two bases correspond exactly, that is, it is possible to recompute exactly the features (x_2, y_2, z_2) using x_1, y_1, z_1; and inversely. In the general case, a complete alignment of the bases might not be possible, but the CCA offers mathematical guarantees that we obtain the best alignment possible. 82
5.8 The complete architecture of the networks we forced to produce decorrelated features. 83
5.9 The reason why the features that are used to compute a decorrelation loss are extracted before any dropout is applied. Dropout, by removing some of the features at random, destroys the correlation between realizations. 84
5.10 The performance of a neural network using a decorrelation loss computed using the theoretical expression of the CCA. 85
5.11 The performance of a neural network using a decorrelation loss computed using the a DeepCCA network. 85
5.12 The scalar products between the classification and decorrelation losses at every batch, on the classic CCA network 86
5.13 The scalar products between the classification and decorrelation losses at every batch, on the two-layer, deep CCA network 86

6.1 The class components are the n_c vectors formed by the lines of the weight matrix W of the last FC layer. . . . 90
6.2 The extraction of deep features . . . 90
6.3 An illustration of the CCA operation. X_1 and X_2 are two 2-dimensional feature spaces representing the same five samples (two points sharing the same colour correspond to the same sample represented with different features). Note that if B_1, B_2 are rotations in our example, they can be any base change in the general case. . . . 91
6.4 Each figure displays the correlation coefficient between each of the couple of canonical variables, generated with the SHL dataset. Two curves (whether dotted or plain) sharing the same colour were computed using the same pair of networks, before (dotted) and after the training (plain). The different colors represent different couples of networks (we do not compute an average because the number of components we kept to get to a full-rank feature matrix changes between initializations). . . . 94
6.5 Each figure displays the average correlation coefficient between each of the couple of canonical variables, generated with the SHL dataset. . . . 95
6.6 the correlation between each of the features and the power of the input signal . . . 96
6.7 The correlation between different versions of trained and untrained networks, with SHL data. 98
6.8 The principle of the equality between class components and the first canonical components on a three-class problem. The colours in the different feature matrices denote the different information about the three classes. The feature vectors will undergo a matrix multiplication (denoted by the arrows under the left matrix); and the rows of the matrix the features are multiplied by are the class components. Similarly, the arrows under the matrix of the canonical variables represent matrix M in equation 6.2. . . . 99
6.9 An illustration of the problem of the comparison of two families of vectors (components), F and G. (6.9a) shows we cannot compare the elements one-to-one, and (6.9b) shows the need for a robust measure (for instance, the dimension of the subspace spanned by family $F \bigcup G$ would not be a good measure). . . . 99
6.10 The first 16 pairs of canonical variables between two initializations of Resnet-50 models trained on CIFAR-10. The colour indicates the class. . . . 100
6.11 The first 16 pairs of canonical variables between two initializations of the accelerometer model. The colour indicates the class. . . . 101
6.12 The first 16 pairs of canonical variables between an accelerometer and a gyrometer models. The colour indicates the class. . . . 101
6.13 The first 16 pairs of canonical variables between an accelerometer and a magnetometer models. The colour indicates the class. . . . 102
6.14 The principle of the subspace projection experiment, illustrated with the projection on the most correlated components (red curve in fig. 6.15): $P_1 = B_1^{-1}.I_n^{n_s}.B_1$ projects X_1 onto a linear space with dimension n_s. The value of n_s is a parameter we will modify in our experiments. 103
6.15 The performance of the networks after projecting their features on subspaces with varying dimensions, on the CIFAR (a) and SHL (b) validation sets. The top row indicates the validation performance of the network as-is, while the bottom row indicates the performance when retraining the classification layer on a projected training set. For each curve, the experiment was repeated 5 times, and the standard deviation is given by the width of the curve (which is sometimes too small to see). The dotted line highlights the performance with the n_c most correlated components. Best view in color. . . . 104
6.16 The classification performance of classification layer using features projected on a subspace with varying dimension, when the CCA is computed thanks to data from another sensor. As in fig. 6.15, one can see that the performance with the n_c most correlated components is close to the performance with all components. The graphs in the diagonal were generated using the same protocol as the first row of graphs in fig. 6.15b. The dotted line highlights the performance with the n_c most correlated components . . . 105

6.17 Why a simple classification measurement is not enough. In this example, the projection on the canonical components (yellow arrow) does not change the prediction of the network for the green sample, for both predictions (pink arrows) land on the same side of the decision boundary. In this example, the logits (the coordinates of the projections of the points on the blue plane) are changed by the projection on the first canonical components. Despite the canonical components (red) being quite far from the class components (blue plane), the classification accuracy is unchanged. 107
6.18 An illustration of the projection on CCA components with toy data. Figure 6.18a illustrates three steps equivalent to the projection on the n_s canonical components (whitening, orthogonal projection, and inverse whitening), while fig. 6.18b illustrates how the kernel is not orthogonal to the image of the projection 108
6.19 Why the class components are relatively unaffected by the projection on the first canonical components, despite appearing quite far from its image: the components are closer to the image than to the kernel of the projection 109
6.20 An illustration of the measure of the angle. Component C_1 (yellow, behind the red plane) is quite close to the y axis, which means it is almost orthogonal to the subspace spanned by the canonical components (xOz plane, in red). Hence, the angle (in green) between this component and its projection on the plane (in red) will be high. On the other hand, component C_2 is closer to the plane, ant the angle between this component and its projection is smaller. We compute an average for all class components, and use this average to characterize the proximity between the subspace and the class components 110
6.21 The average angle between a class components and its projection on the first (a) or last (b) n_s components. 111
6.22 An illustration of the principle of the random projections experiment. When two subspaces are close (*e.g.* the yellow and green), the norms of the projections of a random vector will be close to each other. Inversely, when the subspaces are far from each other, the correlation coefficient between the norms will be low. 112
6.23 The correlation between the norms of random vectors projected on the subspace of class components and their projection on the first (a) or last (b) n_s components. 112
6.24 The distance between the class components and the first n_s components of the image of diverse projections. We display the average over three runs, the width of the curve denotes three times the standard deviation. 114
6.25 The Frobenius distance between the image of the projection and the class components, with SHL data. We display the average over three runs, the width of the curve denotes three times the standard deviation. 115
6.26 The distance between the class components and the first n_s components of the kernel of diverse projections. We display the average over three runs, the width of the curve denotes three times the standard deviation. 116
6.27 The Frobenius distance between the kernel of the projection and the class components, with SHL data. We display the average over three runs, the width of the curve denotes three times the standard deviation. 116
6.28 the different types of correlation between samples illustrated with synthetic data: inter-class correlation (left versus right) and intra-class correlation (top versus down) 117
6.29 The average of the absolute values of the correlations between (a) the first 10 CCA components (b) the 10 class logits (c) 10 directions chosen with a random orthogonal projection, with three networks trained on the CIFAR dataset with a different seed. 118
6.30 The average of the absolute values of the correlations(a) the first 8 CCA components (b) the 8 class logits (c) 8 directions chosen with a random orthogonal projection, with the SHL dataset. For each sensor, we create three different initializations of a network using the sensor. As expected by the intuition, the accelerometer and gyrometer feature and predictions are closer to each other than they are close to the magnetometer. 118
6.31 An example of class shuffling on synthetic data (the colour represents the class). Random shuffle destroys the correlation, and class-shuffle allows to destroy intra-class correlations while keeping inter-class clustering intact. 119

6.32 The distance between the class components and the first n_s components of the image of diverse projections computed on clean or shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation. 119
6.33 The Frobenius distance between the the class components and the image of the projection computed from clean and shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation. 120
6.34 The distance between the class components and the first n_s components of the kernel of diverse projections computed on clean or shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation. 120
6.35 The Frobenius distance between the the class components and the kernel of the projection computed from clean and shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation. 121
6.36 The performance when projecting the features from every layer using the sensor on top on the n_c most correlated components. The CCA is computed from the sensor on the left, using features from the same layer. The experiment is repeated across three network initializations, the standard deviation is given by the width of the curve. We pay attention not to use twice the same initialisation when using twice the same sensor. 125

B.1 The weights of each sensor in the three scenarios for different automatic sensor selection methods. Error bars indicate the standard deviation between each of the five runs. The values of the weights in the last scenario are decreasing because we sorted them (as the sensors are equivalent, we wanted to see if the network listened to only one sensor). 159

C.1 Swarm plots representing the influence of several hyperparameters from random search. Each black dot represents one neural network. Subfigure (a) represents an example of significant choice: N = 3 blocks is strictly better, performance-wise. (b) and (c) are examples of unimportant hyperparameters: using savitzky-golay filters, for instance, is the same as using no filtering. 163
C.2 Swarm plots representing the influence of several hyperparameters from random search. Each black dot represents one neural network. 164
C.3 The swarm plots representing the influence of all hyperparameters from random search. Each black dot represents one neural network. 165
C.4 The classic (left) and residual (right) architectures we used for the Random Search. The classic architecture comes directly from [104], while the second one is inspired from ResNet [150]. 166
C.5 Swarm plots detailing the influence of the architecture, in terms of performance (a), number of weights (b), and number of operations (c). Each black dot represents one neural network . 167
C.6 The performance of each type of network as a function of the number of parameters. 168

D.1 The validation F1-score of 30 initializations of the same model working with full-size spectrograms of the Ori_w signal 170

List of Tables

2.1 An overview of the labeled data in the datasets from the literature. The notation in bold **3**× and **4**× for the durations and number of data points refers to the fact that several measurements are simultaneous, using phones placed at different positions at the same time. The distances from the three SHL challenges were estimated proportionally to the duration of the complete SHL dataset (15,000 km for 2,800 h, [117]). 16
2.3 The chosen hyperparameters for the training of both models 24
2.2 The number of triplegs in each subset of the GeoLife dataset. 24
2.4 An overview of the most recent works using the GeoLife dataset. Along with the performance metric (as provided by the cited works), we outline the qualitative particularities that imply the method might have higher (green text) or lower (red text) performance in a real-life scenario. The asterisks (*) denote the values we recomputed using the reported per-class results from the publications. See appendix A to know how we computed an average F1 from the per-class IoU in [65]. 26
2.5 The validation F1-score per signal. The best result is in bold. 31

3.1 The validation F1-score of each type of padding. Zero-padding is particularly detrimental to the model performance. 37
3.2 A summary of the publications that compared the one-dimensional convolutions on raw data to two-dimensional convolutions on spectrograms, scalograms, or the Fourier Transform of the signal. The lines are ordered by number of samples, approximately 47
3.3 A summary of the most common representation found in every domain we studied. 47
3.4 The validation F1-score (%) per preprocessing method. For each signal, the highest result is in bold, and the second highest result is underlined 49
3.5 The validation F1-score of a network trained on clean data and evaluated on spectrograms whose axis were shuffled (see fig. 3.13). We repeated the evaluation with five random initializations of the network. 56

4.1 The effectiveness of each kind of pooling, in terms of performance, computational resources, and training and inference time. For each result, we display the average and the standard deviation, over 5 runs 66
4.2 The comparison of the performance, number of weights, and number of operations required for a single inference (forward pass) of the networks in the literature. This is a reproduction of table 2.4, which we presented in chapter 2, except that we added the number of weights and parameters. The question marks (?) denotes the publication which did not leave enough details to obtain a precise estimation of the number of weights and operations. `AD` denotes the presence of additional data. 67

5.1 The mean and standard deviation of the validation F1-score of each method, for different fusion modes 80
5.2 The results on the held-out test set. We repeated our training process five times and display the average and standard deviation 87

6.1 A summary of the notations we will use in this chapter 89

6.2 The distances we use to compute the proximity between class and canonical components. $U \in \mathbb{R}^{n \times m_U}$ and $V \in \mathbb{R}^{n \times m_V}$ are two orthonormal bases for two subspaces of $\mathbb{R}^n$, $||.||_F$ is the Frobenius norm, $||.||_*$ is the nuclear norm, and Tr is the trace of a matrix 113
6.3 The parameters of the SVM classifier 123
6.4 The results of a CCA fusion, using diverse combination of sensors with the SHL dataset. The table displays the average and standard deviation over five random runs 123

C.1 The search space of random search for the GeoLife architecture 162

Chapter 1

Introduction

1.1 Context and Motivation

1.1.1 Temporal signals

A *temporal signal* (or sometimes signal) , simply the recording of a physical quantity over time. This measurement may come from any source: audio (barometric pressure sensors), seismographs, inertial sensors, physiological signals, *etc.* Since the 19th century, theories such as signal processing produced analytical tools in order to understand a signal's properties and modify them.

Historically, these tools were mostly used for domains where the signals were already measured, such as automation or audio processing. However, the increasing availability of data, in addition to the development of algorithms encoding these operations in off-the-shelf libraries, make it possible to apply signal processing algorithms in any domain where a measure is regularly repeated. In particular, by extracting a meaningful and compact representation of often high-dimensional temporal data, signal processing proved valuable for the Machine Learning community. For instance, we will see that one way to classify the transport mode of a user is to train a Machine Learning model (*e.g.*, SVM) on the spectral coefficients of the signal. But for the last few years, these interpretable features have been questioned by the apparition of a type of algorithms which is able to learn the feature extraction step from raw data.

1.1.2 A brief overview of recent Computer Vision history

In the years 2000, the classical approach in classification of images was quite different from what it is today. It consisted of mainly two steps: first, compute handcrafted features to encode the information present in the image (Local Binary Patterns, Scale Invariant Feature Transform, *etc.*). Then, use a Machine Learning model (mainly SVM) to classify this series of features. If Convolutional Neural networks already existed [1], the advent of the kernel methods somewhat made CNNs temporarily obsolete.

This was until 2008, when a team of researchers decided to gather a huge dataset for Image classification. They collected enormous amounts of natural images from the Internet and organized the annotation process using Amazon Mechanical Turk. The project involved more than 50,000 people across 167 countries [2], all this in order to produce a dataset that would act as a reference in the Computer Vision community. In order to compare the different image classification algorithms, they also organized from 2010 and onwards a yearly challenge, the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), for which participants had to classify one million images from the dataset.

In 2012, the first ranking submission to the challenge was the one by Krizhevsky *et al.* [3] who, thanks to Convolutional Neural Networks, outstandingly won the competition. Given that the sole neural network participation ranked first, and given the wide margin by which this submission won [4] (the error rate went from 25 % to 16 %), it was not long until the research community experimented with neural networks, and, upon seeing the good results of these methods, switched to a new standard. Deep neural networks quickly became the go-to approach, for image classification and many other applications. One interesting property which explains this popularity is the generalization capability of these deep models: a network

trained on ImageNet (a *pretrained* network) will prove useful for many other Computer Vision problems. More importantly, this utility goes beyond the cases where one can simply use the network as-is: to make use of a pretrained network, one can decide to use its weights as a starting point to begin the learning process on the new problem [5]; or, they can remove the classification layer of the network, to use the features produced by the network as encodings for the images they want [6, 7]. These models are so useful that the deep learning libraries Pytorch and Tensorflow include functions to download ImageNet-pretrained networks for the practitioners to use as they choose.

Nowadays, the research in Computer vision focuses mostly on Deep Neural Networks, with new architectures for classification [8, 9], training processes that require lesser amounts of (labelled) data, such as transfer learning or self-supervised learning [10], or even the generation of realistic images, with the advent of Generative Adversarial Networks (GANs) [11].

Without even reaching the current state of the art in computer vision, creating a Deep model which is as versatile and powerful as the ImageNet models would prove useful for many real-life applications and industrial researchers. Among them is the LSSC, the laboratory which financed this thesis.

1.1.3 A collaboration with an applied research laboratory

The thesis is organized in collaboration with an applied research institution named CEA, *Commissariat à l'Énergie Atomique et aux Energies Alternatives* (Commission for Nuclear and Alternative Energies). Originally created after the Second World War to develop the nuclear energy for industrial applications, this organization became a tool for the French state to develop its industry, and in particular, to close the gap between academic research and the industrial world. Nowadays, this institution is a country-wide actor in diverse research domains (engineering, telecommunications, nuclear physics, *etc.*), and it is a recognized actor in many applied research communities.

We will spare the reader the whole organization chart of departments, services, and laboratories that the CEA is made of, to focus on the laboratory I was in. The LSSC (*Laboratoire Signaux et Systèmes de Capteurs*, Sensor Systems and Signals Laboratory) is a team of fifteen research engineers who work within a department that specializes in embedded sensors. The whole department works like an R&D service for private companies: when an industrial actor has an idea about a technological product they want to develop and would like to obtain a prototype, they contact the CEA. To do so, the organization either design the appropriate sensors internally or make use of existing ones. The LSSC, in turn, is tasked with the design of the software that will process the information from these sensors. Many of the sensors produce temporal sequences of measures, and the typical treatments involve signal processing and Machine Learning. Other activities of the laboratory include the production of studies or the participation in research projects financed by public research agencies. To give a few ideas of the variety of the type of problems the laboratory has to solve, here are a few examples of projects delivered by the LSSC in the last years:

- the classification of stress type from physiological signals (heart rate, electro-dermal conductance, *etc.*) [12]
- the detection of early dysgraphia using recordings of a child's writing [13]
- the tracking of cable shape of cable transportation from cable tension signals [14]
- the classification of the transport mode of a person using smartphone measurements [15]

When they face a classification problem, the research engineers usually rely on classical Machine Learning and handcrafted features. In particular, when they deal with problems that are under-represented in the literature, the laboratory may not be able to afford to have an engineer focus on creating problem-specific features. Benefiting from a deep model producing general-purpose features, like in Computer Vision, would prove extremely valuable for the laboratory, if not for the whole research community. This thesis will not be enough to reach this goal, but it will nonetheless provide a few useful indications in this direction. We will try to pave the way for smarter use of neural networks with temporal signals.

As we cannot work on all possible problems, we decided to focus on a single classification problem, a problem called Transport Mode Detection (TMD). We will concentrate our experiments on this problem,

and we will use the literature to know which conclusions apply to the general case, and which are proper to our problem.

The choice of this problem relies on several reasons:

- Firstly, it is a problem involving temporal signals where some temporal features (frequency bands) are important, without solving the problem entirely.
- It is multimodal, in the strictest definition of the word (*i.e.*, the problem does not involve a single sensor with several channel, but different sensors).
- Lastly, having a practical solution was interesting for the CEA.

1.2 The problem of Transport Mode Detection and its application

Transport Mode Detection (TMD) is a family of classification problems in which an algorithm has to predict the transport mode of a given user, using several signals. The exact list of possible transport modes may vary depending on the application (most research papers use between four and eight modes), but most applications include at least the most common ones, such as Walk, Bus, Train, or Car, for instance. The signals are collected from an embedded device (either the sensors of a mobile phone, or a dedicated device), and processed by a TMD Algorithm. In practical applications, this algorithm is implemented either in a client-server fashion (the device sends the raw data to a server that runs the transport Mode Detection) or directly runs on the embedded device. The algorithm may run either in real-time (predicting the transport mode instantaneously), or return a delayed prediction, after the user ended their trip. The practical implementation of the algorithm translates into specific constraints for the programmer: if the algorithm runs in an embedded device or in real-time, it should have low computational requirements; if the algorithm runs in a client-server fashion, privacy guarantees are appreciable [16].

When returning delayed predictions, the algorithm most usually starts from a complete trajectory, partitions it into single-mode segments, and predicts the transport mode of each of these segments. Some algorithms directly work with a trip that may contain several transport modes, and return the sequence of all the transportation modes the user took. Other algorithms use only classification, that is, they return only a single class. This may happen either because the trajectory is known to contain only a single mode from the start; or because the algorithm makes its predictions on segments so short they can assume the segment to be unimodal with little errors (typically, a segment duration of one minute is enough, given that the typical duration between two mode transitions is 20 minutes). In the next chapter, a review of the literature will allow us to explain how one uses Machine Learning to this goal.

This field has several applications, such as smart trajectory planning, city-wide transportation studies, or even carbon footprint estimation. An example of the latter application is the project led by the LSSC before the beginning of the thesis, in collaboration with another French institution, the IFPEN (*Institut Français du Pétrole et des Énergies Nouvelles*, French institute for petrol and new energies). This organization wanted to develop an app for anyone to estimate the greenhouse gas and pollutant emissions of their car trajectories, and help the users reduce their CO_2 emissions. Concretely, the application detects the beginning and end of every user trip or waits for user input to classify the transport modes of a trajectory. After the end of the trip, the app uses a Machine Learning model to predict the transport mode of the user and computes an estimation of the greenhouse gas emitted during the travel given the trajectory duration, the type of vehicle involved, and the driving style of the driver (if applicable). After the project, the institute continued developing this app on their own and, nowadays, it is available for download under the name Géco air for Android and Apple[1].

We could mention that Transport Mode Detection is already implemented by default in Android phones [17]: Android proposes an API available for use by any application which was granted permission by the user[2]. However, we do not know how does this API behaves internally, or what features or models it relies on. The closest official information comes from a page from 2015 which mentions the use of Machine Learning

[1] Sadly, the author of this manuscript did not receive any compensation for the promotion of this product.

[2] This applies only for applications developed after mid-2020, which means the applications that have not been updated before this date still do not need any permission to use the activity recognition.

and Bluetooth signals [18]. A study from 2018 [19] shows that the performance of this API is far from perfect. Another imperfection of this implementation is the fact that the class granularity is rough: among the eight labels returned by the API, there are six TMD classes (still, walking, running, bicycle, in a vehicle), but all the vehicles belong to a single class. This means that despite existing practical implementations, TMD is not a solved problem. We will see in the next chapter all the recent improvements in the academic domain.

1.3 The proposed approach and contributions

In this thesis, we are investigating how the paradigm of deep learning, which proves successful in computer vision, can be used and adapted to handle the TMD problem while dealing with various temporal signals. The proposed approach is therefore Deep Learning-based as opposed to traditional Machine Learning approaches which require manually handcrafting features.

In doing so, we have to address several scientific questions:

- *How to preprocess temporal signals to use deep neural networks ?* We study the bibliography to find which representations are used in different subfields of deep learning and perform a comparison of different representations ourselves. Finally, we try to understand what does the computation of a spectrogram bring to our model.
- *How to merge data from different sensors ?* After an empirical benchmark of different fusion methods in the literature, we study in detail one fusion method we found in the literature, an algorithm based on a statistical operation named Canonical Correlation Analysis.

Our attempt to answer these questions led to the following contributions:

- We exhibit one reason why spectrograms are better than the raw data for a neural network: they allow to simplify the problem of Transport Mode detection.
- We introduce the use of global pooling in the domain of deep learning for Transport Mode Detection, which allows us to use convolutional networks with inputs of any size.
- We establish a benchmark of different data fusion algorithms in the literature.
- We demonstrate that using Canonical Correlation Analysis to perform data fusion, as done in literature do, is equivalent to a much more simple operation: considering the class logits of the networks.

1.4 Outline of the manuscript

The present manuscript will try to cover the different types of problems a practitioner faces. Usually, when a research team tries to design an algorithm to solve a practical problem, they have to answer several questions: which sensors to use, how to preprocess the data, how to choose hyperparameters for the network (including how to choose an architecture), how to make sure they evaluate their model properly etc. Without giving definitive answers, we will cover each of these subjects and give some indications for future researchers. The organization of the manuscript will reflect these choices.

First, chapter 2 will present the state-of-the-art, the datasets we use (the GeoLife and SHL datasets), and the two baselines our work will use in the next chapters.

In chapter 3, we will look at the problem of the preprocessing. After a brief introduction showing how simple preprocessing step (the choice of a good padding for shorter segments) can change the performance, we will spend some time to know whether one should use their one-dimensional sequence of data points as is, if they should compute the one-dimensional spectrum of the signal, or if they had better using time-frequency diagrams. To do so, we will first review the type of preprocessing used with one-dimensional signals in different fields of application, before trying to do the comparison ourselves. Finally, we will show that the spectrograms (the time-frequency diagrams) allow simplifying the TMD problem by making the classification of the easiest classes linear.

When the sensors and the form of the input are chosen, one decision that often comes to practitioners is the choice of an architecture for the network. Chapter 4 will be devoted to one particular decision: the choice of a pooling function for the network.

A decision that also comes with the architecture selection is the choice of the architecture to merge the data from different sensors, a problem called *data fusion*, which we will cover in chapter 5. We will start by implementing several data fusion architectures from the literature to compare them and conclude that, in our case, most data fusion methods are equivalent. We will try to improve on the different methods by forcing the networks to learn complementary features, and we will show that in fact, the networks decide better by themselves what is the optimal amount of correlation between their feature.

However, the tool we use to produce complementary features, Canonical Correlation Analysis (CCA), is not devoid of interest. The final chapter will be devoted to the study of this operation. Mainly, we will demonstrate that when we apply CCA to the features from a neural network, this operation implicitly recomputes the classification logits which are the inputs of the softmax layer. The main implication is that a data fusion relying on CCA can safely be replaced by a much more simple operation, which is to the sum of logits of each network before computing the probabilities.

Chapter 2

Related works, datasets, and baseline

This chapter is in three parts: section 2.1 will present an overview of the literature in Transport Mode Detection (TMD). Then, section 2.2 will review the most important TMD datasets available in the literature. Finally, section 2.3 will focus on the two datasets we used. For each of them, we will show the network we will build our experiments upon.

2.1 State of the art

Transport Mode Detection is a well-studied subject in the research community. Similarly to the literature, this thesis will focus on TMD from GPS or inertial sensors (accelerometer, gyrometer, magnetometer), even though other sensors are available: some works focus on Transport Mode Detection using the signal received from radio towers [20], Wifi signal [21], potentially along Bluetooth [22, 23], or even sound [24, 25]. The first two are not very present in the literature because of their limited performance, while the sound is rarely used because of important privacy concerns. Hence, we will focus on the two types of sensors that are the most commonly used in the literature: GPS signals and inertial sensors.

As figure 2.1 shows, many works follow a common structure: Cleaning, Point-level feature computation, Segmentation, and either Trajectory-level classification using classical Machine Learning algorithms; or using a pure Deep Neural Network.

The next pages will detail what these steps consist of, and what are the different alternatives used in the literature. This section will *not* try to compare the different alternatives, mainly because there are multiple datasets or problems to consider (the exhaustive list of reasons that present an effective comparison will be provided at the end of the section). However, we will compare the publications relying on the same dataset when presenting our baselines, in section 2.3.

2.1.1 Data cleaning

When dealing with signals that are are known for being noisy (such as GPS signals [26]), most research works use Kalman filters [27], Savitzky-Golay filters [28], particle filters [29], or outlier detection thanks to clustering techniques [27] to clean the data. Some approaches also remove the trajectory from the dataset if its speed or acceleration is above a realistic threshold given the class of the trajectory (for instance, the maximum speed and acceleration for the 'bus' class are set to 120 km/h and 2 m/s^2 respectively in [28]).

For inertial sensors, we notice the use of Butterworth filters [30], or undersampling [31] to work with smaller amounts of data. But many choose to simply ignore this step (see [32, 33] for instance).

The possible options to clean a signal seem to come mainly from signal processing. In addition, given that the works that ignore the cleaning step do not report suffering from the noise in the data, we think there is little to gain by improving the cleaning of the signal. Combined with the fact that the signal processing research domain is well-theorized, and much more stable than the Machine Learning or Artificial Intelligence one, we expect to see little improvements of this step in a near future.

The relevance of this step might not even be obvious at a first glance because it is sometimes done by those who gather the dataset as part of the *curation* step (such as the SHL dataset we will present later).

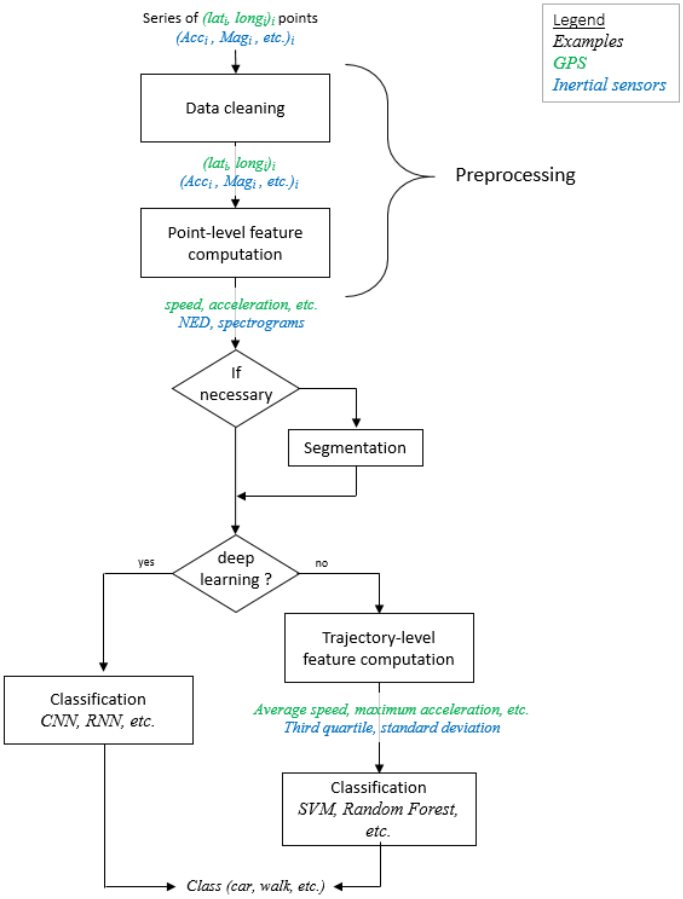


Figure 2.1: The outline of a classic Transport Mode Detection algorithms, along with a few examples of each step.

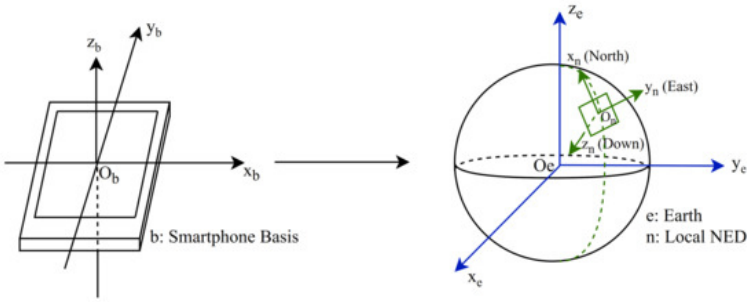


Figure 2.2: An illustration of the meaning of the axes in a smartphone: the referential of the sensor (x_b, y_b, z_b) and the de-rotated, or NED (*North, East, Down*) coordinate system. Reproduced from [54].

We include it here for two reasons: First, researchers might still have to perform cleaning themselves in some cases (the GeoLife database, which we will present later, is one example). Second, forgetting it might result in a mismatch between the academic and industrial practices. Even if it is more convenient for academic comparisons to share clean datasets, we must keep in mind that the practitioners will have to deal with this additional work [34].

2.1.2 Point-level feature computation

This step consists in computing a representation of the input data which encodes the information more explicitly than the raw representation. This is where the differences between GPS and inertial sensors are the most important. In fact, every problem has its own point-level processing.

GPS signals: When dealing with GPS data, researchers often convert the *(lat, long)* into more significant values. Those features are computed at each timestamp. Speed and acceleration are used universally. Other point-level features include distance [35, 36, 37], jerk (the time derivative of acceleration [27, 37]), or delta-bearing (the angle difference at each time step, sometimes called *bearing rate*) [27, 37].

The works of Endo *et al.* [38] and Zhu *et al.* [39] are fairly original: the compute heatmaps of the GPS measurements. They use a fixed-size grid centred on the barycenter of the GPS. Endo *et al.* [38] demonstrated that the discretization into grid cells helps the algorithm to be resilient to noise.

Apart from these two exceptions, the speed, bearing, and their time derivatives are the only point-level features that have seen any use in the literature since the emergence of transport mode detection in the years 2000. The improvements in the feature computation all belong to the trajectory-level features (section 2.1.4) or, more importantly, to the architectures of deep networks and their training (section 2.1.6). Hence, we expect this step to keep being stable throughout the years for GPS signals.

Inertial sensors: With inertial sensors, the most common preprocessing is the computation of the norm (or magnitude) of the triaxial signal: $norm = \sqrt{x^2 + y^2 + z^2}$, [31, 30, 32, 40, 41, 42, 43, 44]. As these sensors measure in their own referential (see fig. 2.2), there might be a need to express them with a referential that does not depend on the movements of the phone. To do so, researchers often convert into the *North-East-Down* (NED hereafter) coordinate system, using the real-time orientation [45] (sometimes under the name *derotated features* [46]). The NED signals are often considered in addition to the raw signals from the sensor's referential (see fig. 2.2 for an illustration of this referential), [33]. In addition, the Euler angles are sometimes added to the orientation quaternion [45, 47].

We mention seeing the gradients [46] and integrals of signals [48, 49]. We also notice the computation of the Fourier Transform [25, 33] or Spectrograms (two-dimensional diagrams consisting of sequences of Short-Term Fourier Transform [31, 50, 25, 51, 52, 53])

Why is computing a FFT not part of the trajectory-level feature computation step: The border between point-level and trajectory-level features is not crystal clear (the spectrogram, for instance, offers an interesting in-between), and one could argue that computing the Fourier spectrum of a signal is already computing trajectory-based features. For our application, the clearest difference between the two is in the dimensionality of the data: the number of point-level features may change if the input sequences have different lengths, while the number of trajectory-level features remains constant. For example, the full spectrum has a number of

points that is equal to the number of points of the input trajectory in the temporal domain, while the power in a series of fixed frequency bands will always output the same number of values, no matter the frequency resolution of the input sequence. Hence, in this work, the Fourier transform is part of the point-level features, and the energies in different frequency bands are considered as trajectory-level features.

The rare publications that do justify their choice of a point-features use either an *a priori* reasoning on the contribution of each type of features (like we will do partly in section 2.3.3), a measure of the consumption of each sensor, or an empirical measure of the performance. This last criterion necessitates particular caution: there would be nothing wrong with a research domain being purely empirical, but measuring the generalization ability of a model is not as straightforward as it seems, and we will see that many works can be criticized on that aspect. Note that we will dedicate a chapter (chapter 3) to the choice between raw temporal data and spectrograms.

2.1.3 Segmentation

The goal of segmentation is to separate a potentially multimodal trip into several portions with a single transport mode. This step is often skipped, most research works (including us, or [28, 55]) prefer using the ground truths to be sure to work on segments containing one transport mode. This corresponds to a simpler version of the real-life problem, where we do not consider the errors added by the segmentation algorithm. Those who segment using the data typically rely on walk detection (a small walk is usually necessary between two modes, [35]) or the PELT algorithm [56, 57]. See [58] for interesting considerations on segmentation algorithms in trajectory analysis.

Some researchers work on a *semantic segmentation* problem: semantic segmentation is, like classification, a type of supervised problem, as they both require some amounts of labelled data to create a model able to make predictions on unseen data. However, classification is a type of problem where the model returns only a single prediction per sample. For semantic segmentation, the model returns a localization of the different classes present in each input sample, to a resolution that can go up to one label per input data point. This problem is well studied in computer vision and other research domains, and multiple Machine Learning models or neural networks exist to return more than one prediction. Surprisingly, however, most of the works in the TMD literature do not use any of these semantic segmentation-specific models. Instead, they apply a classification algorithm to small windows of the trajectory [30, 59, 57, 31, 45, 42, 60, 61, 62]. Note that these approaches are only possible because the data we use is sequential in nature: using moving windows might not be feasible with two-dimensional (or even three-dimensional) data for complexity reasons. Some [59, 63] predict the mode using partitions of the segment into windows of different lengths (each window length creates a new partition of the segment), and obtain a prediction at each timestep with a majority voting. For those who output a sequence of predictions, an additional step is sometimes present: a correction of the prediction sequence using either an explicit transition matrix [59, 61], rules [62, 64] or a HMM [31, 45, 42, 60].

We found only one publication using semantic segmentation techniques in the literature: Li *et al.* [65] used an architecture similar to UNet ([66]) in order to process in one forward pass the trajectories with potentially several transport modes. They also use a semi-supervised training and a post processing step to smooth the predictions. We will see that many ideas in the recent TMD literature are directly adapted from the Computer Vision literature after a delay of several years. Hence, we expect to see a development of the semantic segmentation in TMD in the next few years.

2.1.4 Trajectory-level feature computation

For classical Machine-learning approaches, *trajectory-level feature computation* has two objectives:

- Get back to a fixed-size feature vector, usable by machine learning models: the number of point-level features is often proportional to the length of the input trajectory, while the number of trajectory-level features is constant.
- Extract meaningful information from the features: for example, with GPS signals, the standard deviation of the speed is more meaningful than the speed of, let us say, the fourth point of the trajectory.

The computation of the mean, standard deviation, minimum, and maximum of each point-level feature is universal. In the case of the noisy GPS signals, the maximum and minimum are often replaced by percentiles, which are more robust to errors [27, 67]. Other used operations are the computation of the median [68], interquartile range [69], the number of zero crossings of the temporal signal [70, 48], kurtosis [69], frequency energy bands [71], autocorrelation coefficients [67, 72], or maximum and index of the maximal coefficient of the autocorrelation function [73]. Some of these rarer trajectory-level features are computed using the spectrum of the point-level feature signal. This is the case of the most important frequency [74, 31], the center of gravity of the spectrum [73], spectral entropy [31, 70, 69], the coefficients of the wavelet decomposition [75], or cepstral coefficients [76].

Some trajectory-level features do not rely on a specific point-level feature, such as the stop rate and direction change rate [35], tripleg duration [77]. Several works also improve the predictions using features from external sources, like the weather [78] or the closeness to train lines [79] and/or bus stops [80, 81, 82]. The additional information needed to compute these last three features is obtained from sources such as Open Street Map [80, 82] or Baidu Map [81]).

Given the large majority of features used, some works decide to use automatic feature selection, whether by adding features one by one [27, 70, 83], by a global analysis of the correlations between features [31, 84, 44], or using PCA [85, 86].

2.1.5 Classification: classical Machine Learning

For classical Machine-learning approaches, the state of the art is Random Forests [27, 87, 70, 87, 62, 88, 44] and SVM [89, 82, 70]. Other classifiers include XGBoost [51, 74, 90, 91], decision trees [92, 93] NaiveBayes [71], Multi Layer Perceptron [38, 81, 71, 87], KNN [67, 85], HMM [94], Gradient Boosting [48, 75], logistic regression [95], statistical models [93], or even rules [96] and fuzzy rules [97, 98]. Many works [77, 48, 99, 100] compare several Machine Learning classifiers. In these works, XGBoost, Random Forest, and SVM are often among the best performing classifiers.

One may also use an ensemble of different Machine Learning classifiers [31, 84, 48]. After each model made its prediction, the final decision is made with a Hidden Markov Model [31, 84], a Multi-Layer Perceptron [48], or a simple majority voting [41].

Sometimes, the classification is hierarchical: a first classifier distinguishes two groups of classes from each other (for instance, the motorized modes, such as car, bus, train, against the others), and a second and third classifier are trained to recognize the class on each subgroup. This is done to simplify each classification problem, and can be noticed in [59, 40, 76, 90, 101].

Surprisingly, when researchers report the reasons to choose a model, they mostly mention performance alone, measured using the accuracy, the F1 score, or even the AUC. Apart from the hierarchical classifiers and (fuzzy) rules, domain knowledge is never used to choose or even design a classifier. It seems that the only source of improvement to this step is the advances in Machine Learning. However, the progress in this direction has slowed down since the advent of deep learning: it seems that neural networks have become the most active source of inspiration for TMD.

2.1.6 Classification: Deep Learning

Within deep learning-based approaches, there is a great diversity of neural networks: we note the existence of Convolutional Neural Networks [102, 103, 28, 104, 57, 105, 37, 52, 47, 106], which extract representations that are independent from the position of the motif along the temporal axis. We see Recurrent Neural Networks (RNNs, [33, 107]), or LSTMs [108, 109, 110, 43, 111] (a specific kind of RNNs). The architecture may even integrate both convolutional layers and recurrent layers, such as in [112, 99, 39, 113, 114, 115, 78]. In these situations, the convolutional layers are often closer to the input data, to extract the features: we saw only one counterexample (that is, one work where the LSTM layer is closer to the input than the convolutional layers are [116]), and with limited results [117].

This organization of the networks (convolutional layers first, and recurrent layers afterwards) is consistent with the role of each type of network:

- *Convolutional layers* are more suited to extract representations from raw data. The convolution layers learn representations which are invariant to the location of the motif, while the pooling layers provide

valuable dimension reduction

- *Recurrent networks* are more suited to classify sequential information.

RNNs and LSTMs are not often used on raw data ([108, 109, 43] are notable exceptions). Most of the time, they are used either on handcrafted features ([46, 33, 107, 95]) or after a series of convolution layers ([112, 39, 113, 114, 115, 78]). This mirrors the use of RNNs in the general literature: For instance, in Natural Language Processing (before the advent of transformer architectures), the recurrent neural networks used features from word embedding such as Word2Vec [118], which were trained in an unsupervised setting. To sum up, in TMD, extracting meaningful features from the raw data seems to be the prerogative of Convolutional networks.

In some occasions, ensemble of deep models are used [28, 37, 51, 46]. We can also notice the use of attention mechanisms, [119, 120]. However, these works may use different names, and/or different implementations from the "baselines" of attention [121, 122]

We do not notice the presence of ensembles of RNNs. This may be due to the fact that Recurrent architectures take longer to train than convolutional ones, due to their low degree of parallelism [122].

We should also mention the works of [105] who used Generative Adversarial Networks (GAN) to generate samples from the least represented class in order to produce a balanced version of the dataset. They compare the performance of an ensemble of CNNs trained on the augmented dataset to a single CNN trained on an oversampled version of the original dataset and notice the ensemble method has a higher performance for all of the five classes they considered. In a similar fashion, [123] created a conditional GAN for which a classifier makes a prediction on the sample, and the discriminator sees both the prediction of the classifier and the sample to say whether it is real or fake. In addition, [124] considered a particular type of GANs, where the discriminator did not have to only predict if the sample was real or fake, but it also had to predict the class of the real samples. Hence, the discriminator's output had $5 + 1$ classes (Walk, Car, Bike, Bus, Train, for real samples, and Fake). Finally, we should mention the works of [125], who used Adversarial Autoencoders, [126], a variant of the classic Autoencoder where the discriminator has access to the latent vector of the autoencoder and learns to distinguish it from a random Gaussian variable so that the Autoencoder fools the discriminator and generates latent representations that follow this distribution.

Some [102, 38, 81, 57, 110, 127, 65] use autoencoders to extract features from trajectories but, curiously, only four [57, 110, 127, 65] make use of additional unlabeled data. These works are called *semi-supervised*: that is, they use samples without their labels to teach their models to process the data, along with a fraction of labelled samples to learn a classification boundary. Making use of the unlabeled data is useful in many industrial applications because collecting the labels is often the most expensive part of the dataset creation process. In the case of TMD, collecting unlabeled data allows the user not to label their transport mode constantly, which might translate into more frequent recordings. The fact that so little publications made use of unlabeled data is surprising in TMD because the most famous GPS transport mode database (the GeoLife database) contains a majority of unlabeled samples. In a similar fashion, the works of [55], who uses *unsupervised* clustering with a convolutional autoencoder in order to create features from trajectories. They realize that the clusters they computed align fairly well with the classes of the dataset, even though they never made use of the labels to compute the clusters.

We also mentioned how some publications used data from other sources (the positions of bus lines [81] or the weather [78]). However, to the best of our knowledge, none used GPS data from other sources to help training a semi-supervised model.

Currently, this step is the main source of innovation in TMD. It benefits most from Computer Vision, which is the main driving force in the research in Deep Learning. We can even notice that the improvements in deep learning take a few years before being adapted in TMD: the semi supervised work were published in 2018 ([57]), 2020 ([127]), and 2021 ([65]); and the works involving GANS date from 2019 ([124, 125]) and 2020 ([123, 105]). This is why we think one obvious direction of research in TMD is to keep bringing to this domain the innovations from the general Deep Learning community, and in particular, Computer Vision. Given that the recent trends in computer vision involve using GANs [11], training the networks using self-supervised losses [10], the use of transformer architectures [9], or semantic segmentation [128], we could expect these domains to emerge in TMD, possibly after a few months or years of acclimation.

However, there is one domain which we think will not appear in TMD: the use of simulated data. In Computer Vision, many works generate data with graphical engines to increase the amount of data they

train their model with. When we look at simulation data in Computer vision, we can see that the creation of simulated data relies on realistic simulated images obtained from graphical software like Blender, from graphical engines of video games, both including rich libraries of objects and textures. The important point is that the software used for the creation of artificial images is already available, with open licenses and a documentation allowing easy reuse. In the case of signals from GPS or inertial sensors, several sensor-specific reasons prevent an effective simulation:

1. The GPS signals should, ideally, incorporate the noise of the real signals that is due the reflection of the signal on buildings.

2. For accelerometers or gyrometers, if it is possible to model the movements of a human skeleton, a realistic simulation should also take into account the fact that the sensor is not fixed rigidly to the user's body. Instead, a typical smartphone is usually in the user's pocket or bag, and a realistic simulation should model the bouncing of the sensor's support.

3. For the signal from magnetometers, a realistic simulation should take into account the different perturbations: presence of electrical engines of trains or subways, fluctuation of the magnetic field of the Earth inside the metallic cabin of a vehicle, *etc.*

In all three cases, the physical laws governing the phenomena are well known. However, there is a substantial difference between knowing the theory and making a practical simulator: before using artificial data, the TMD research community should learn how to generate artificial data realistically and efficiently. One could also think of using GANs to generate artificial data, but this approach is, to the best of our knowledge, currently unexplored in TMD.

The efficient generation of artificial samples is a technological lock that prevents to adapt the use of simulated data in TMD. Hence, before even trying to import the literature on the use of synthetic data from Computer vision, one should additionally develop a realistic simulator. The other domains (GANs, transformers, focusing on semi-supervised, self-supervised, or even unsupervised networks) are easier to adapt from Computer Vision. This is why we think that the research in the next years will focus on using less (labelled) data, before trying to leverage the power of artificial datasets.

2.1.7 The problem of the evaluation

Even though every classification algorithm is systematically evaluated to be compared to the state of the art, in practice, comparing approaches of different works is not as easy as one could think, for four reasons:

1. The first one is the fact that there is no ImageNet for TMD: many researchers conduct a study in which they sample new data (by leading an entire data collection campaign) and perform some Machine Learning, but without publishing their dataset [87, 79, 129, 130, 82, 92, 131, 40, 132, 133, 134, 110]. This makes the comparisons harder: if, for instance, the data comes from a city with lenient speed limits, the cars and buses will tend to drive faster, and the distinction between motorized modes and bikes will be easier. In addition, if the data is recorded under the direct supervision of a researcher (such as in [130], for example), chances are that the subject will be aware they are surveyed, and act differently than they would have in their daily lives (this is the *Hawthorne effect*, [135]). These two examples illustrate the need to compare performances on the same dataset, as one cannot quantify the effect of these differences, nor can they enumerate all possible sources of performance discrepancy between two datasets.

2. Even when two papers work on the same dataset, they might not use the same set of classes: for instance, in one of the datasets we will use (the GeoLife dataset), many classes originally present in the set form an extreme minority of the samples (motorbike, car, boat). This is why the creators of the dataset advise the researchers to remove some classes and to merge others [89]. However, given the varying applications of TMD, some researchers decided to merge more classes than the original recommendation. This leads to a situation where, despite working on the same dataset, two publications work on different problems. To know how a change in the categories employed would affect the results, one could look at the confusion matrix (fig. 2.12). For instance, if two modes are frequently confused

with each other, merging them would cause the model's performance to increase. This applies to the merger of {car & taxi} with bus [28, 57, 109] and, to a lesser extent, to the merger of the classes train with subway. A similar reasoning could be made about the removing of the two rail modes (train and subway), as [108] do (leaving only four modes, {walk, bike, car, bus}): not only are these modes no longer confused with each other, but they are not even confused with the car or the bus. As one could expect, the number of classes employed correlates negatively to a model's performance (see table 2.4 in section 2.3.2).

3. The third hurdle to an efficient comparison is the use of multiple scores to evaluate the classifiers. Some works (including recent ones [127, 55]) use the accuracy, that is, the proportion of samples that are correctly classified. Yet, the accuracy is biased towards the most frequent classes [136]. To illustrate it, let us use the famous example of a classification system that considers a given individual, and predicts the presence of a rare disease that touches 0.1% of the population. If this system always predicted that the individual was healthy, it would have an accuracy of 99.9%, despite being obviously flawed. In TMD, the classes are not unbalanced to this degree, but the disproportion is large enough to make the accuracy score unreliable (see fig. 2.10 and 2.14, along with table 2.2). When considering the Transport Mode Detection publications using GPS signals, the works that use an appropriate measure are the slight minority: one work computes the AUC [108], and some research publications use the F1-score [27, 28, 28, 105]. Worse, we sometimes even see some direct comparison between different scores: a recent survey Sadeghian *et al.* [137] (table 2) cited the results from [28, 57] by putting their resulting F1-score in the "accuracy" column of their table (which also contains true accuracy scores).

4. Finally, the last hurdle is an efficient splitting between the training, validation, and test set. Not all research works use a three-set separation, many works preferring to only use one train and one validation set, to compare their validation score against the state of the art. This leads to an obvious risk of overfitting, as the hyperparameters one chose might be specific to the dataset the model was evaluated on to some extent (this point is further reinforced by the fact that many publications do not display any standard deviation, see table 2.4). The second downside is that the creation of a test set leaves a bit less data for training, which means a model trained without a test set benefits from a training size that is inappropriately larger than what it should be. We should note that in several publications, the validation set (the set that is used to choose the Machine Learning algorithm or calibrate the hyperparameters) is called *test* set [38, 127]. Even for those who do use a separate test set to compare their results against the state-of-the-art, the separation between the sets is primordial. With the two datasets we will use, computing the training, validation, and test set is not straightforward, and there are pitfalls one must be careful of in order to avoid hidden overfitting sources. We will present these in the next section.

Normally, a test set is used to evaluate the generalization ability of a model. Each time one uses any dataset to select a model, they have a small chance to select improvements that are proper to this very dataset, instead of choosing a model that generalizes well. We can use a test set to choose between a handful of publications, because the probability of choosing a model that is better on this particular test set is low. However, if every publication uses the test set to select the best model, the probability to overfit to this set increases significantly. In other words, if we compare publications on the validation set, our comparison will be biased towards the publication that did the most evaluations on the validation set.

This is why the organization of a challenge, with common rules and hidden test set, is so important: the fact that the test set is unavailable is a strong guarantee against overfitting, no matter how rigorous the participant are, or how much the validation score of the publication is. A challenge guarantees to avoid a situation where there are so many qualitative precautions to keep track of that numerical results lose any meaning (a situation that somewhat happened for the GeoLife dataset, see section 2.3.2).

When working with these datasets, we will use the F1-score that takes the imbalance into account, and will follow the recommendations of the GeoLife dataset creators [89] when multiple choices of classes are present. However, comparing the approaches relying on GPS signals to the literature will be difficult. When we will work with a dataset from a challenge, we can at least choose to be in the exact conditions of the challenge, using the 'hidden' test set (which labels have been released since) only once, to compare ourselves against the participants. In the next sections, we will present the datasets the publications base their results

on, including the two datasets we used (the GeoLife dataset, and the SHL 2018 challenge), and, the baselines that most of the experiments rely on.

2.2 Datasets

This section is devoted to the presentation of the main public datasets in TMD. If we only chose to use two of them (the GeoLife and the SHL 2018 datasets), we explain why we did not choose the others. Table 2.1 summarizes the different datasets.

2.2.1 The GeoLife dataset

The GeoLife database [89, 138, 139] was collected between 2007 and 2012. The GPS signals of 180 participants living in five different cities of China were recorded during their commutes, in order to study their behaviours when travelling. Unfortunately, the need for labelled data did not appear immediately, and only one tenth of the trajectories of the database is labelled (in the subsequent, we will only refer to the labelled data, unless otherwise specified). The dataset is an ensemble of trajectories, each trajectory being a series of $(latitude, longitude, timestamp)$ points. Each labelled trajectory has one or more transport modes, and each change between modes has an associated timestamp, so that each point can be attributed a label. The transport modes (classes) in the dataset are: *walk, bike, car, taxi, bus, train, subway, boat, airplane, motorcycle*. An overview of the dataset is available in table 2.1. We follow the recommendations of the GeoLife user guide [89], removing the classes, *boat, airplane*, and *motorcycle*, and merging together the classes *taxi* and *car*. Figure 2.3 gives some examples of trajectories in the dataset. One can see that most of them follow the straight lines of the street networks (most streets are oriented either on the North to South axis or on a West to East axis) using the histogram of the angles in fig. 2.4. Finally, a histogram of the different classes' trajectories durations is available in fig. 2.5. One important thing to note is that the data points are not sampled at the same rate: some trajectories have a sampling rate of 1 or 2 Hz, while some others can go down to 0.02 Hz on average.

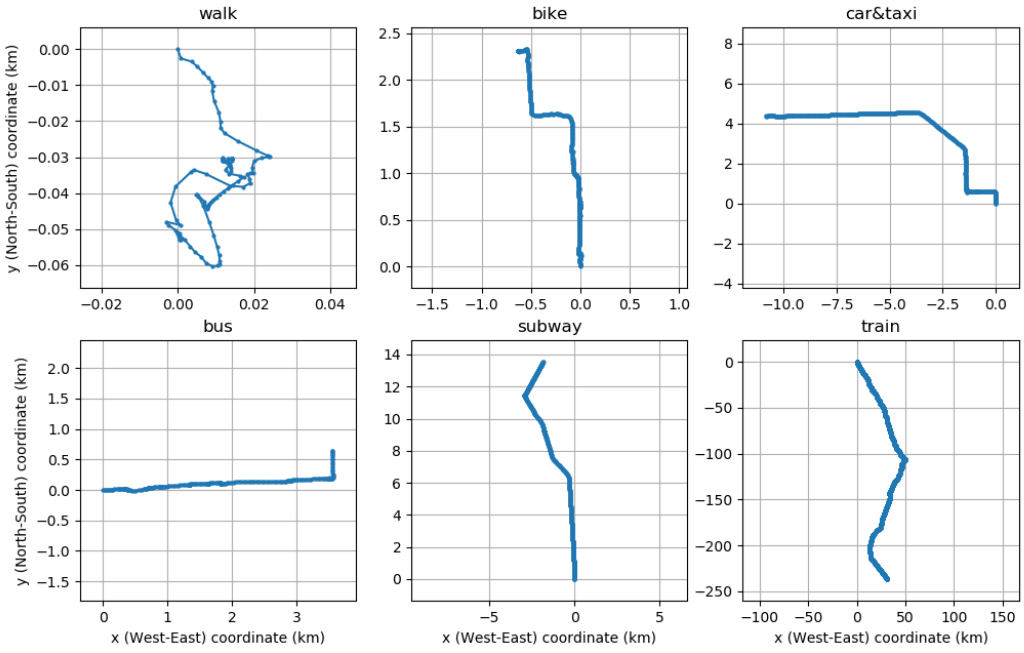


Figure 2.3: A few examples of monomodal portions of trajectories in the GeoLife dataset. The start of the trajectory is set to be the origin

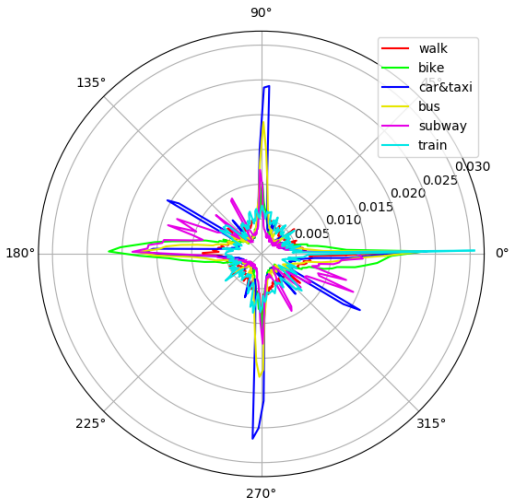


Figure 2.4: The histogram of all the orientations between two successive GPS points in the GeoLife dataset. The fact that the 0°, 90°, 180°, and 270° values are over-represented indicate that most of the trajectories follow the orientation of the grid-like road system.

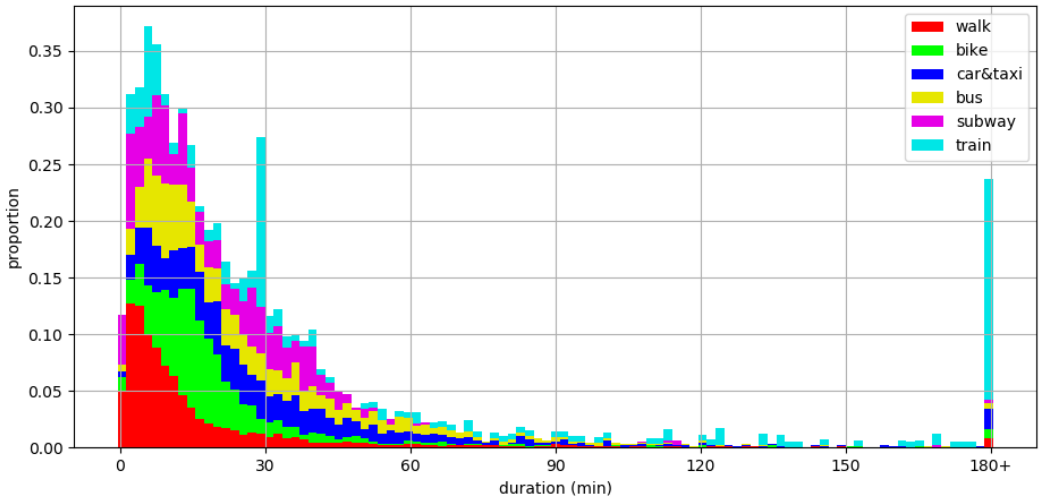


Figure 2.5: The histogram of the durations of each monomodal segment in the GeoLife dataset. As expected by the intuition, the Walk segments are typically quite short (less than 15 minutes), while the train segments can be extremely long (more than three hours).

Dataset	GeoLife [89]	SHL 2018 [140]	SHL 2019 [141]	SHL 2020 [117]	TMD [19]
Number of users	69	1	1	3	16
Total duration	5,000 h	272 h	**3**× 272 h	**4**× 312 h	32 h
Total trajectories length (estimated)	116,000 km	1,200 km	500 km	1,700 km	unknown
Average interval between two data points	7s	0.01s			between 0.2 and 10s (depending on the sensor)
Median interval between two data points	2s	0.01s			between 1s and 10s (depending on the sensor)
Sensors	GPS	Accelerometer, Magnetometer, Gravity, Linear Acceleration, Gyrometer, Orientation quaternion, Pressure			Accelerometer, Magnetometer, Gravity, Linear Acceleration, Gyrometer, Orientation quaternion, Pressure, Sound, Light, Step detector, internal sensors, *etc.*
Position of the sensor	unconstrained	Pocket	Pocket, Bag Torso	Pocket, Bag Torso, Hand	unconstrained
Number of channels	3	20			59
Total number of trajectories	$10,000$	$16,000$	**3**×$200,000$	**4**×$220,000$	248
Total number of data points in the database (for each channel)	2.5×10^6	9.6×10^7	**3**×3.2×10^7	**4**×1.1×10^8	1.4×10^6 to 3.3×10^6 (depending on the sensor)
Classes	Walk, Bike, Bus, Car & Taxi, Subway, Train	Still, Walk, Bike, Run, Bus, Car, Subway, Train			Still, Walk, Bus, Car, Train

Table 2.1: An overview of the labeled data in the datasets from the literature. The notation in bold **3**× and **4**× for the durations and number of data points refers to the fact that several measurements are simultaneous, using phones placed at different positions at the same time. The distances from the three SHL challenges were estimated proportionally to the duration of the complete SHL dataset (15,000 km for 2,800 h, [117]).

2.2.2 The SHL 2018 challenge

The Sussex-Huawei Locomotion challenge 2018 is a competition organized by the University of Sussex, UK, in collaboration with Huawei. The participants were given a series of 16,310 consecutive annotated recordings of embedded sensors and had to classify some 5978 samples of a test set. Each recording is 60-seconds long, and contains data from 7 sensors, and 20 channels: three axes (x, y, z) for the accelerometer, magnetometer, gravity, linear acceleration (acceleration minus gravity), and gyrometer; one orientation quaternion (x, y, z, w), and a recording of the barometric pressure. The accelerometer, gyrometer, magnetometer, and pressure are said to be *real* sensors, because they were recorded directly from a sensor, in contrast to the linear acceleration, gravity, and orientation, which are said to be *virtual* sensors (that is, sensors whose value are computed by the phone system). Each signal was recorded at $100Hz$, so that one sample to classify is a set of 20 vectors of size $60 \times 100 = 6000$ points. There are 8 classes available: Still, Walk, Run, Bike, Car, Bus, Train, Subway, and figure 2.6 displays some examples of signals from samples of all classes. The participants had to design an evaluation protocol to train and evaluate their models on the 16,310 annotated samples, and submitted their predictions on the 5978 test samples for evaluation. Then, the organizers of the challenge evaluated all these predictions and established a ranking of the participants, before releasing the annotations on the whole dataset. Our protocol mimics this setting: the annotations on the 5978 test samples are used only once in chapter 5.

2.2.3 SHL 2019 and 2020 challenges

The data used to set up the SHL 2018 challenge come from a larger dataset. This dataset gathers the data of three users, who were asked to record the data from four phones at once: one in their pocket, one in their hand, one in their bag, and one on their torso. They also equipped the users with a RGB camera taking a picture every 30 seconds, so that they could review the photos to correct any labelling mistakes.

In 2018, 2019, and 2020, three challenges were organized with different subsets of this dataset. In 2018, the challenge was a simple classification problem: the training and testing data are samples from the first user, using only the phone in their pocket. In 2019, the challenge was a transfer learning problem: the organizers released the data from the pocket, bag, and torso phones of the first user, and asked the participants to train a model predicting the transport mode using the data from the phone in the user's hand [141]. To help with the validation, a small amount of hand-phone data was also provided.

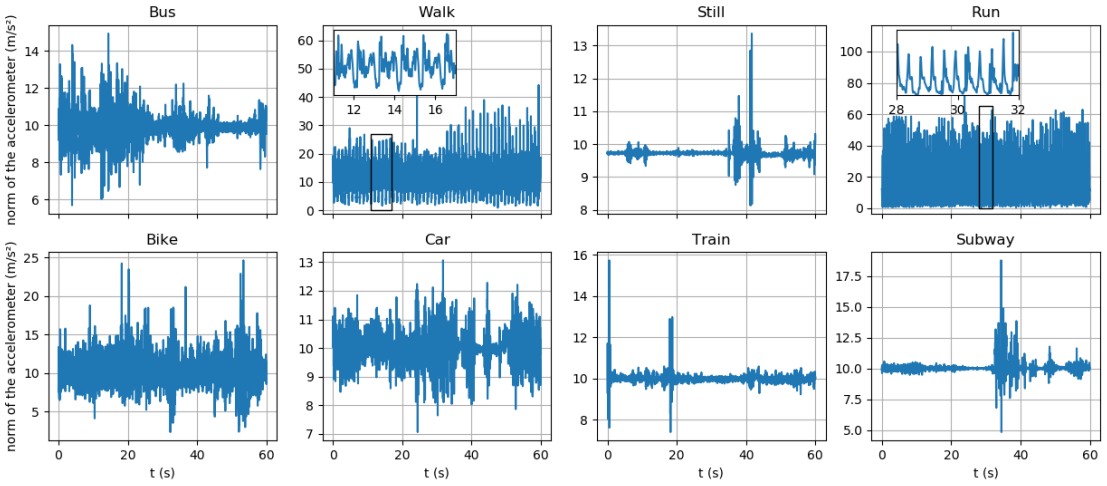
In 2020, the challenge was still a transfer learning problem, but this time, the participants were to transfer to new users. The organizers released the data from the first user (four positions), along with a small amount of data from the two other users, and asked to predict the transport modes of the two other users [117]. Using the same set of classes as the 2018 challenge, the best test F1-scores of the SHL 2020 challenge were 88.5 %, 79.0 %, and 77.9 %. As the first-ranking participation used the labelled data from the second and third users to train their final model (this data was originally intended for validation only), we can say that a F1-score of 77.9 % to 79.0 % is more representative of the real-life performance of the best algorithms.

These challenges are interesting but, given that they include a transfer learning dimension, we chose not to explore them. In this manuscript, we will focus on well-known supervised classification.

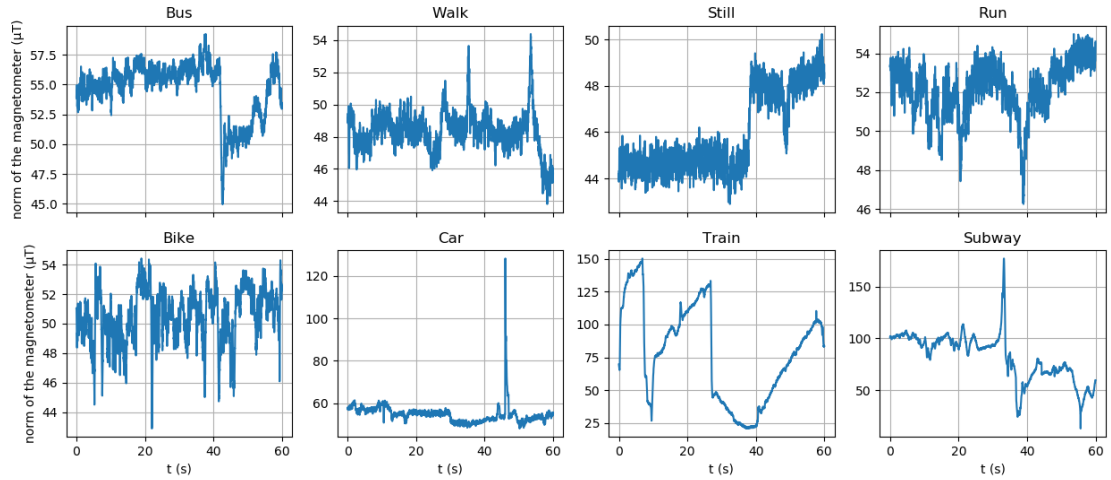
2.2.4 The TMD Dataset

In 2017, the University of Bologna started recording a dataset to act as a benchmark for transport mode detection. They asked sixteen volunteers to record the data from their phones' sensors using an application they created for the occasion, and recorded this data to publish it the following year [19]. They adequately named their dataset TMD (for Transport Mode Detection). This dataset contains about thirty hours of data, shared approximately equivalently between five modes: Still, Walk, Car, Bus, and Train.

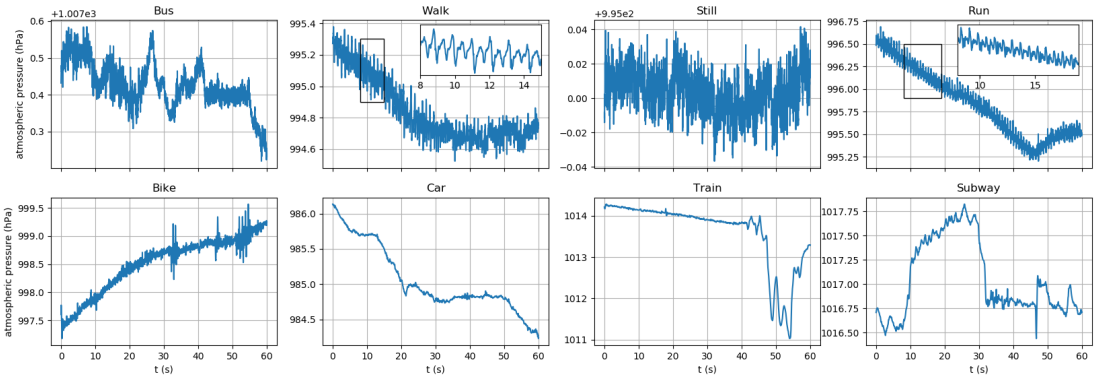
In addition to the user imbalance (similarly to the GeoLife dataset, each user has their own class balance), this dataset suffers from an extreme frequency imbalance: contrary to the SHL dataset (where the sampling frequency is 100 Hz across the whole dataset), the TMD dataset has a sampling frequency which ranges from 0.2 to 200 Hz (fig. 2.7). This is why we did not make use of this dataset: if we had to filter only the segments with a similar sampling frequency, we would have been left with an extremely small dataset. And we did not consider the fact that each segment is not necessarily sampled regularly (which fig. 2.7 does not show).



(a)



(b)



(c)

Figure 2.6: Some example of signals from the SHL 2018 dataset, for each of the 8 classes and for three sensors: a) norm of the accelerometer b) norm of the magnetometer c) barometric pressure

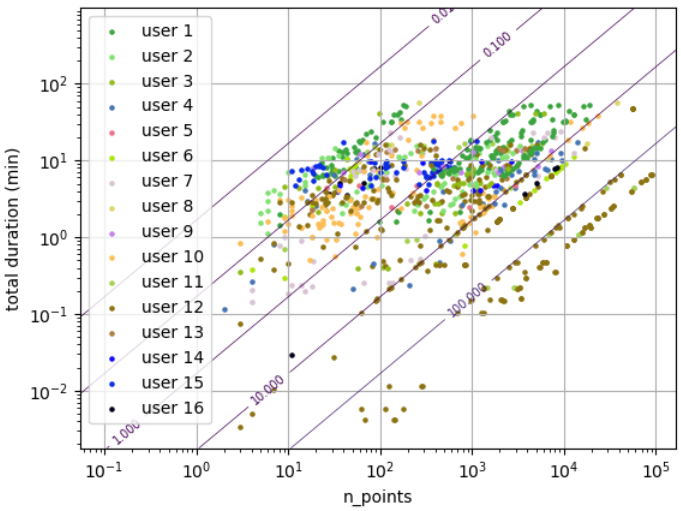


Figure 2.7: A scatter plot of the samples from the accelerometer measurements of the TMD dataset, depending on their number of points and total duration. The diagonal lines illustrate the iso-frequencies (average frequency in the sample), in Hz.

There are ways to use convolutions on irregularly sampled data (like point-clouds [128, 142]), and the use of these types of networks could have offered a solution. They are, however, slightly out of scope for our work. We could also think of leveraging this diversity through Transfer Learning, or Few-shot learning: the general idea would be to find a way to leverage both the huge amount of data in the SHL challenges and the diversity of the TMD dataset. We are not sure exactly how one should proceed, however, and this might be an option for future work. But for now, let us focus on the work we did perform. The next section presents the networks that tackle TMD on the GeoLife and SHL datasets.

2.3 Baselines

In this thesis, we focus on the GeoLife and SHL 2018 datasets. Compared to the state of the art, and contrary to many publications in TMD and elsewhere, we do not aim to improve a classification score by introducing an new algorithm or computational step. Instead, we will focus on evaluating the choices made in the literature, so that a new practitioner does not have to evaluate them all empirically.

For each dataset, we develop an approach, a *baseline* which will serve as a starting point for our other experiments. After a general introduction of our framework (section 2.3.1), we will present the two datasets we focused our work on the GeoLife (section 2.3.2) and the SHL 2018 datasets (section 2.3.3)

Methodology

In the next chapters, we will evaluate several types of choices: type of preprocessing, architecture, sensors used, *etc.* In order to make the comparisons, we will rely on one model per dataset which we present here. Figure 2.8 illustrates our experiments: for every type of parameter choice (each choice corresponding roughly to a chapter), we will change only the concerned parameters and leave all the others equal as their baseline versions.

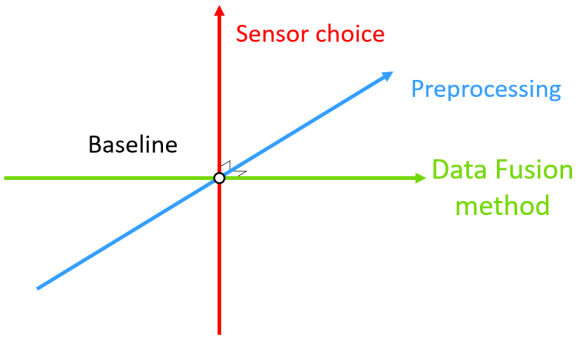


Figure 2.8: The outline of our methodology. As we cannot explore the whole search space, we simply explore along a few chosen directions, changing one type of option at a time.

Why choosing a CNN ? This thesis focuses on processing temporal signals with deep neural networks. One could think that the Recurrent Neural networks are more suited to this problem, yet, we use Convolutional Neural Networks in all our experiments. The reason for this is that we are mainly interested in the computation of temporal features. As we mentioned in section 2.1.6, RNNs (and LSTMs) are mostly used to process high-level information, with a relatively low number of temporal steps. In other words, the inputs of these networks are much higher-level abstractions than the raw data. On the other hand, computing features from raw data requires handling data with no abstraction whatsoever. For instance, a segment of the SHL dataset has 6,000 points, and one point in itself has very little meaning. This is why we will focus on CNNs, whose convolutions demonstrated their ability to extract lower-level features in several different domains [143, 144, 145].

2.3.1 Transport Mode Detection as a classification problem

Our goal is to use some labelled data in order to learn to assign a transport mode to unknown data. Using Machine Learning vocabulary, this is a classification problem: we use a certain amount of labelled samples to train a classifier that will predict the transport mode used during the recording of an unseen segment: $class(segment_i) \in \{walk, bike, bus, car, train, subway, etc.\}$

Some additional preprocessing is applied so that the data can be fed to a machine learning model. To train and test properly the model, the dataset will be split into train, validation and test sets. A fraction of the data with the associated labels will be used to train the machine learning model to predict the class of an unseen segment. This is the *train set*. We usually have a wide choice when selecting preprocessing functions. In addition, machine learning models generally involve several hyperparameters, *e.g.*, number of filters, or number of layers, that need to be chosen. These hyperparameters are optimized and chosen in evaluating the variants of the machine learning models on previously unseen segments, the *validation set*. Once we have chosen every possible parameter using the validation set, we use the last part of the dataset (*test set*) to evaluate the generalization of the learned model against the state of the art. The following

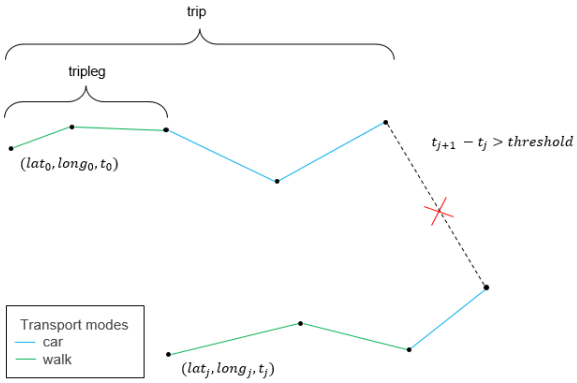


Figure 2.9: The separation of a trajectory with the GeoLife dataset. Each trip is composed of triplegs; each tripleg correspond to a single transport mode. When two consecutive points are distant more than a certain threshold (chosen to be equal to 20 minutes in our case), we break it into two neighboring trips.

algorithm shows how we use the three datasets:

Require: Three datasets X_{train}, X_{val}, X_{test}, for training, validation, and testing; a list of hyperparameters sets L_h to evaluate.
  $best\_score \leftarrow 0$
  $best\_hyperparameters \leftarrow \emptyset$
  for h in L_h **do**
    Create a deep learning model with hyperparameters h
    Train the model on X_{train}
    Evaluate the model on X_{val}, measure the F1-score $score_{val}$
    if $best\_score < score_{val}$ **then**
      $best\_score \leftarrow score_{val}$
      $best\_hyperparameters \leftarrow h$
    end if
  end for
  Create a deep learning model with hyperparameters $best\_hyperparameters$
  Train the model on X_{train}
  Evaluate the model on X_{test}, measure the F1-score $score_{test}$
  return $score_{test}$ for comparison with the state-of-the-art

2.3.2 GeoLife Baseline

Preliminary definitions

Figure 2.9 illustrate how we get to a classification problem: the dataset contains a series of *trajectories* (each trajectory being a series of measurements points, such as $(lat, long)$ for GPS signals. Each trajectory has to be divided into *trips*: series of points that are likely recorded in one go (we chose GPS points such that two consecutive points are distant by less than 20 minutes, as in [35]). Each trip is made of one or several *triplegs*: series of points sharing a single transport mode. This setting corresponds to a more simple version of the problem, where we know triplegs to have only one mode. In real-life applications, we could consider applying segmentation algorithms like in [81, 57, 35].

The triplegs might still have different lengths, but a model sometimes requires all inputs to have the same shape. When this is the case, we cut triplegs into fixed-shape *segments*. When we use a model that can deal with arbitrary-sized inputs, segments are equal to triplegs. Thus, a model only classifies segments.

Preprocessing

We begin by computing the speed and acceleration of each point. This way, our data is not dependent on the precise location of the trajectory. We remove the data points which acceleration or speed are deemed unrealistic given the annotated transport mode (we reused the values from [28]). We considered adding the bearing [28], but it turned out using this feature did not increase the performance of our model. As the sampling rate is irregular, we interpolate linearly our data points (T = 2s), so that a difference between two consecutive points always has the same meaning. We realized when writing these lines that the interpolation step was absent from the literature and that we should have at least evaluated it before moving on. However, the comparisons we will make in the next chapters all use the interpolated dataset, which means they are still relatively valid.

We do not apply any other cleaning or filtering, for we found these to be unnecessary during the Random Search (see appendix C).

Data Preparation and Splitting

Etemad [27] showed that the way the segments are split between the different sets (training, validation, test) can have a huge influence on performance: when a tripleg is split into several segments and the segments of a single trajectory can go in both the training and the test set, the trained model will be likely to have seen parts of all trajectories in the dataset, which will cause it to overfit.

In his experiments, Etemad found the F1 score can vary by 20 percentage points between a training scenario in which the users are correctly split (71 %), compared to a scenario in which the fragments of a given trajectory can go in different sets (91 %). This result is not surprising, as mobility trajectories have a high degree of regularity [146, 147, 148]. Ideally, we should train a prediction model using the trajectories from a set of users and test the generalization skills of the trained model on those of unseen users. This means that we need first to assign the trajectories of each user to one set among training, validation or test (as in [27, 38, 28, 57]). This corresponds to the most realistic setting, where an algorithm predicts the transport mode of unseen users.

As an illustration, here is a pseudo-code of the separation into users:

Require: A list L_{users} of users, each user being a list of trips.
  Randomly split the list L_{users} into three lists $\{L^{train}_{users}, L^{val}_{users}, L^{test}_{users}\}$
  for s in $\{train, val, test\}$ **do**
    $L_{tripleg} \leftarrow \emptyset$
    for each user u in L^{s}_{users} **do**
      Split the trips of u into triplegs and segments,
      Add the triplegs to the list:
      $L_{tripleg} \leftarrow L_{tripleg} \bigcup u_{split}$
    end for
  end for
  return the three tripleg lists $L_{tripleg}$ obtained for $\{train, val, test\}$

But in practice, with the GeoLife dataset, this method leads to extremely imbalanced sets, as users have different habits when it comes to transportation. In some cases (depending on the seed used to initialize the splitting process), splitting the dataset by users can even produce validation or test sets that completely lack one class. To show this, we realized 200 separations with different seeds initializing the random process, and looked at the effect it would have on the final distribution (fig. 2.10). While the median is centered around the correct distribution, the quartiles show a high variance. In the validation and test set, the third quartile is at least twice higher than the first quartile. This distribution variance is a problem for comparison, because the performance of imbalanced models will strongly depend on the distribution, even when using measures like the F1 score: if the hardest classes to classify are present more often, the recall for this class will not change, but the precision of all the classes will be lower, which will, in turn, lower the F1 score.

This is why we only split the sets by tripleg: we first separate triplegs between train (64% of the trajectories), validation (16%), and test (20%), before segmenting the triplegs into segments. This method is less realistic than splitting the sets by users, but it allows to produce sets with similar distributions consistently, given that each tripleg has a unique class. The following pseudo-code explains how we split by

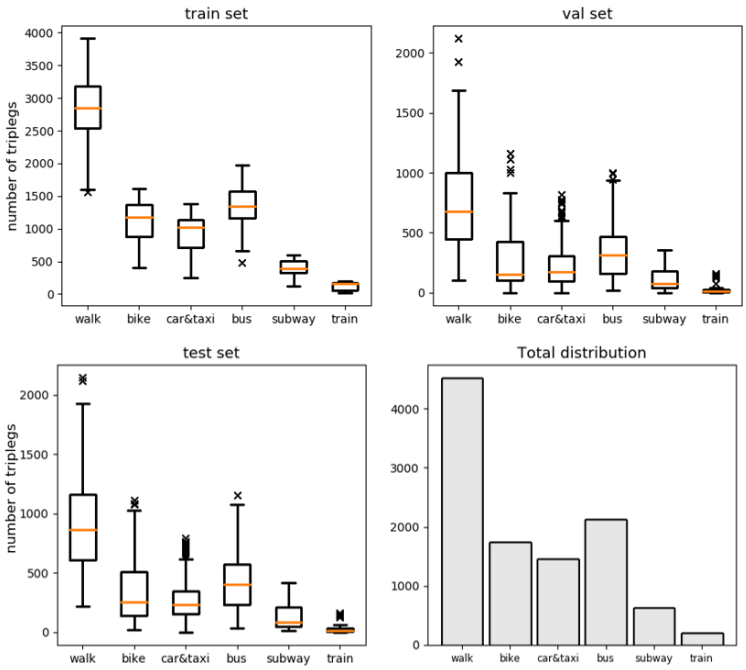


Figure 2.10: An overview of 200 distributions obtained by splitting by users. Depending on the realizations, the proportion of each class can vary greatly: in the validation and test sets, the first quartile is always at least twice lower than the third quartile. In extreme cases, one set can completely lack one class.

triplegs:

Require: A list L_{users} of users, each user being a list of trips
  Create a list L_{trips} of trips by merging all users
  Randomly split the list L_{trips} into three lists $\{L^{train}_{trips}, L^{val}_{trips}, L^{test}_{trips}\}$
  for s in $\{train, val, test\}$ **do**
    $X_s \leftarrow \emptyset$
    for each trip t in L^s_{trips} **do**
      Split the trip t into triplegs and,
      Add the triplegs to the set:
      $X_s \leftarrow X_s \bigcup t_{split}$
    end for
  end for
  return the three tripleg sets $X_{train}, X_{val}, X_{test}$

We did not realize it at the time, but there might have been a solution to split by users while still keeping a similar class distribution between sets. For instance, this problem looks similar to the knapsack problem (a classic Computer Science problem where we are given a list of items, each having a weight and a value, and we must choose the items to maximize the total value while keeping within predefined weight bounds), and we could have tried searching for solutions in the literature. Or, we could have been less subtle: the speed of the script computing the 200 splits is enough for us to use a brute-force solution, trying random splits until one is well distributed enough. Using such brute-force solutions might seem surprising and inelegant, but we must keep in mind that if elegance is desirable, it must not get in the way of actually solving the problem. However, we did not consider these options at the time of experimenting.

In any way, splitting by tripleg is different from a split by *segments*: in this case, the segments are assigned a set (among train, val, and test) at random and independently, it means the two fragments of the same trip can go in different sets. This separation is the most likely to result in data leakage and, to the best of our knowledge, is not justified in any way.

Parameter	value (GeoLife)	value (SHL)
learning rate	10^{-2}	10^{-3}
regularization parameter	3.10^{-3}	10^{-3}
batch size	128	64
non-linearity	ReLU	ReLU
optimizer	Adadelta [149]	Adam
max number of epochs	2000	50
patience	100	/

Table 2.3: The chosen hyperparameters for the training of both models

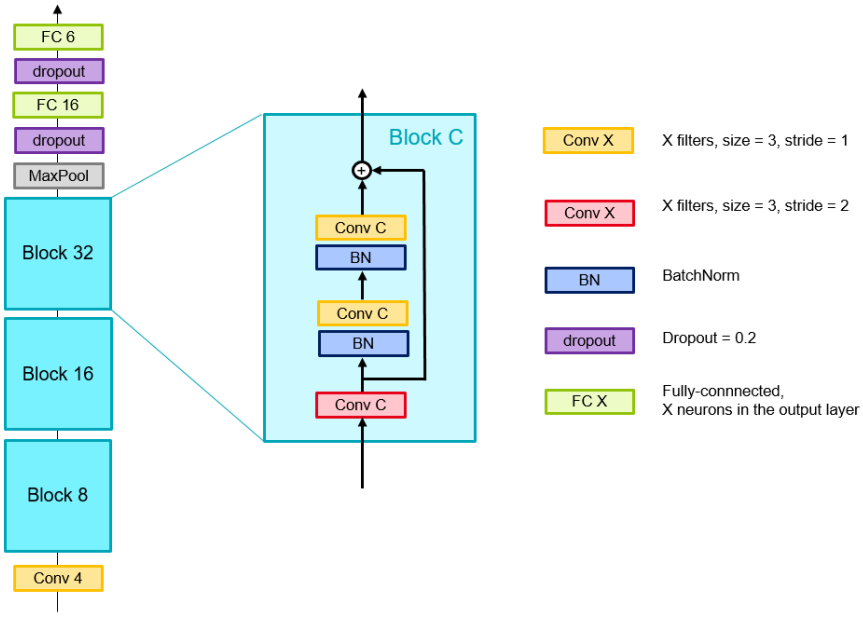


Figure 2.11: The architecture of the GeoLife baseline model

	walk	bike	car & taxi	bus	subway	train
total	4517	1731	1459	2129	632	200
train	2890	1108	934	1362	405	128
validation	723	277	233	341	101	32
test	904	346	292	426	126	40

Table 2.2: The number of triplegs in each subset of the GeoLife dataset.

Baseline architecture

The CNN on the GeoLife dataset uses the speed and acceleration features to classify a segment. The input is a one-dimensional signal (the values are stacked along the time axis), two-channel signal (one channel for the speed, one channel for the acceleration). Its architecture is inspired by ResNet [150] and is given in fig. 2.11. To train our neural network, we use a weighted version of the cross-entropy loss: the loss of every

segment is given a weight that is inversely proportional to the number of elements in this segment's class on the training set (which is proportional to the proportion on the whole dataset, see section 2.3.2). The goal of this procedure is to compensate for the class imbalance in the dataset. We also apply early stopping: we stop the training process when the loss on the validation set did not increase for more than a fixed number of epochs (chosen to be 100, see table 2.3), and we keep the model which minimized the validation loss for testing. Usually, this minimum loss is reached between epoch 100 and 600.

A nontrivial comparison with the literature

As we mentioned in section 2.1.7, comparing the performance of two publications in TMD is not easy, and the literature on the GeoLife dataset exhibits many of these discrepancies that prevent a fair evaluation: different number of classes, biased metrics, and separation between train, validation, and testing.

If our experiments in the next chapters use the 6-class problem recommended by the GeoLife creators [89], we still tried to compare the performance of this baseline model to the literature. When there are different *valid* methods for comparisons, (number of classes, F1-score versus AUC), we repeat the training and testing and display the test result each time (we allow to use the test set several times here because this will not result in a choice of a number of classes or error measure). However, we kept using the hyperparameters found using the 6-class problem. We also enumerate each element that might change translate into higher or lower performance in the real case: allowing the trajectories to have several transport modes, for instance, is an element that means the reported score will be more realistic of the performance of a real-life TMD algorithm because the error measure includes a step (the segmentation of trips into triplegs with a single class) which might decrease the score in the real case. On the other hand, splitting the training and validation sets by segments means the score is likely to be inflated, or dubious.

Note that the meaning of the scores differ slightly depending on the exact problem solved by the publication: a method for which the transport mode may vary will look at each point of the segment and count the number of *points* for which the model returned the correct class in order to compute a performance score (Accuracy, F1, AUC, *etc.*). Yet, a publication working with the hypothesis that a trip can only have a single mode only counts the number of *trips* that are classified correctly. In this manuscript, we put the two scores side by side and allow ourselves to compare, for instance, a F1 on points to a F1 on trips.

As table 2.4 shows, the four GeoLife baselines look acceptable, but there are so many of these exceptions to consider that the numerical score loses its meaning. We also display the confusion matrix of the GeoLife model on the 6-class problem in figure 2.12. As one can see, some classes are fairly easy to distinguish from the rest (the Bike, Train, and Walk classes), while others couple of classes are harder to discriminate (car & taxi versus bus and, to a lesser extent, subway versus walking or train).

model	reported score	classes	remarks
LSTM + embedding [108]	94.5 % AUC	4	No mention of the splitting No test set
Our GeoLife Baseline	97.1 ± 0.3% AUC	4	/
Convolutional LSTM [78]	80.67 % F1	4	No additional information No test set
Convolutional LSTM [78]	83.97 % F1	4	Additional information: weather No test set
Our GeoLife Baseline	87.1 ± 1.1% F1	4	/
Convolutional Auto Encoder [57]	76.4 % F1	5	The trajectories are not segmented Additional information: Unlabeled GeoLife data No test set
Convolutional Auto Encoder with skip-connections [65]	80.4* % F1 67.7 % $\overline{IoU}$ (average Intersection over Union)	5	The trajectories are not segmented Additional information: Unlabeled GeoLife data No mention of the splitting No test set
Fully-connected Autoencoder [81]	93.44 % F1	5	The trajectories are not segmented Additional information: Bus stops position Incorrect splitting No test set
Unsupervised Convolutional Autoencoder [55]	80.5 % Acc.	5	They did not use any labels to compute the clusters No mention of a validation set
CNN ensemble (7 models) [28]	84.0 % F1	5	No test set
semi-supervised LSTM ensemble [127]	91.5 ± 0.41% Acc.	5	No test set
LSTM + Wavelet features [109]	91.9* % F1 92.7 % Acc.	5	No mention of the splitting No test set
Our GeoLife Baseline	83.9 ± 1.1% F1	5	/
Random Forests [27]	71 % F1	6	No test set
Our GeoLife Baseline	81.8 ± 0.9% F1	6	/
AE + Logistic Regression [38]	67.9 % Acc.	7	No test set
Our GeoLife Baseline	74.1 ± 0.7% F1	7	/

Table 2.4: An overview of the most recent works using the GeoLife dataset. Along with the performance metric (as provided by the cited works), we outline the qualitative particularities that imply the method might have higher (green text) or lower (red text) performance in a real-life scenario. The asterisks (*) denote the values we recomputed using the reported per-class results from the publications. See appendix A to know how we computed an average F1 from the per-class IoU in [65].

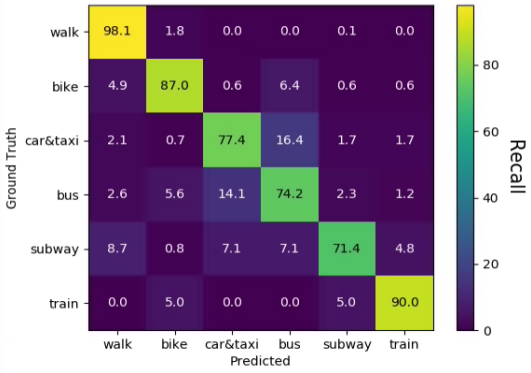


Figure 2.12: The confusion matrix of the GeoLife model, on the test set. We can see that there is a high confusion between the classes "Car&taxi" and "Bus": merging together these classes will improve the performance considerably.

2.3.3 SHL Baseline

Separation of the training and validation sets

With the SHL 2018 challenge, the test set is already split from the rest, but the separation between train and validation was left to the participants. We choose to have 13,000 samples in the training set, and 3,000 samples in the validation set. This division corresponds roughly to an 80/20 separation. However, similarly to the GeoLife dataset, there is a caveat to consider before running the first training. This time, it is not about user separation (there is only one user in the database), but about the trip separation problem which holds here too.

As noticed by Widhalm *et al.* [42], if one randomly assigns segments to either of the sets, the segments in the validation set will be too close to segments in the train set: for each segment in the validation set, chances are that at least one segment from the training set was recorded right before or after it. They even show the autocorrelation coefficient of the standard deviation of the norm of the accelerometer is equal to 25% of the maximum at $t = 10min$, which means the training and validation samples must be well-separated.

A random split leads to possible overfitting: as the validation set is too close to the train set, the validation performances will be unrealistic. To illustrate, in the experiments from [42], the differences between a correct, rigorous splitting and a random one are significant, between 7 and 11 percentage points in F1-score.

In order to have a realistic evaluation, the segments of the training set must be as far as possible from segments from the validation set. The first idea to split the data in a time-consistent fashion is to sort the database chronologically, take the first 13,000 samples for training, and leave the last 3,000 samples for validation. However, this method leads to a high imbalance between the training and validation set. The reason behind this is the following: the distribution of the different transport modes changes with time (see fig. 2.13 for an illustration). In particular, the 'Car' class is absent from the end of the dataset. If one chooses to take the last 3,000 for validation, the validation set will practically lack this particular class. To achieve better class balance, we still sort the samples chronologically, but we use the *first* 3,000 samples for validation, and the *last* 13,000 samples for training.

Preprocessing

The 2018 SHL Challenge asked candidates to give one prediction per timestamp (that is, to output 6,000 predictions per sample) but, as only 4% of the samples of the database have more than one mode, we work on a simpler problem: each sample in the dataset is assigned a single transport mode, which is the mode at $t = 30s$.

We did not consider cleaning the data, for two reasons:

- Firstly, cleaning was found to be ineffective on the GeoLife dataset (appendix C), and GPS signals are more noisy than embedded sensors [26].

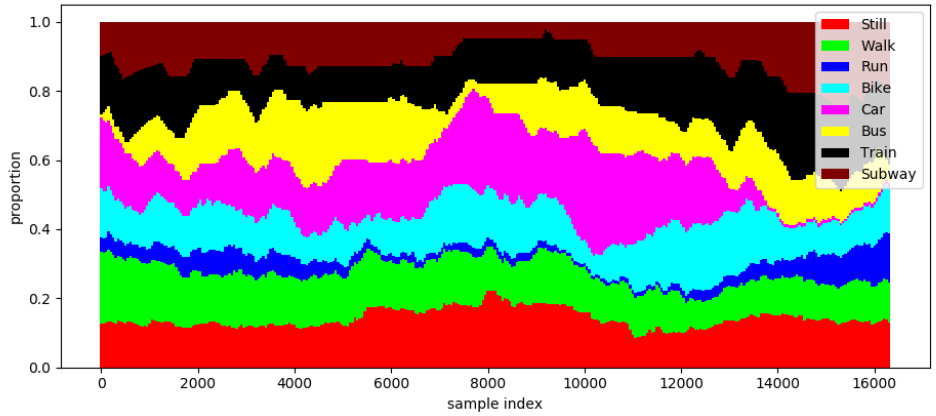


Figure 2.13: The cumulative proportion of each class versus time. The class proportion is computed on a moving window of 1,500 samples, with a stride of 50. We can see the end of the dataset is devoid of Car segments.

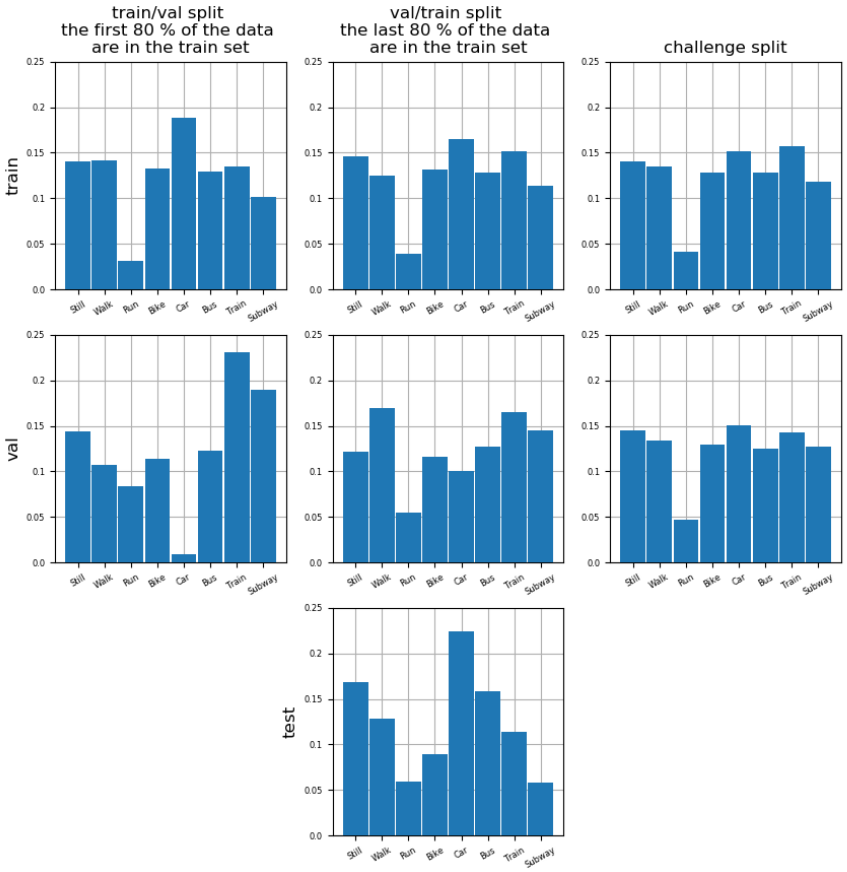


Figure 2.14: The histogram of the classes in each set (lines) for each splitting (columns). The train/validation split made by the organizers of the challenge is balanced, but it does not take the chronological order into account. A train/val split does, but the train set lacks run segments. A val/train split of the samples is both rigorous and balanced enough. As the test set is already split by the organizers of the challenge, its content does not depend on the splitting.

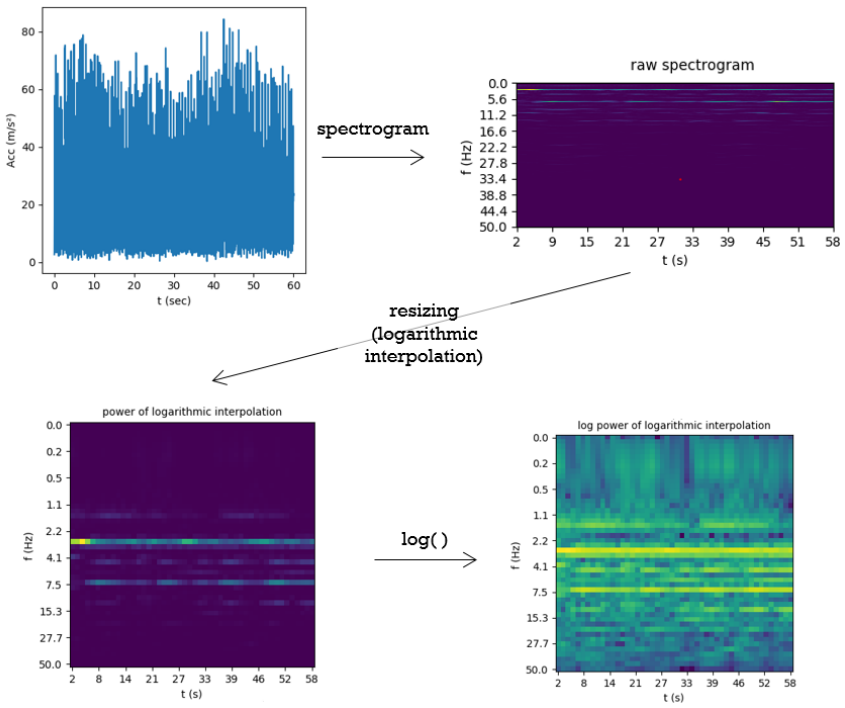


Figure 2.15: An illustration of the preprocessing step with the norm of the accelerometer data from a running segment. The 2.3 Hz frequency band appears in the middle of the spectrogram due to the log scale for the frequencies. This 2.3 Hz frequency is approximately the frequency at which one foot touches the ground. The use of the log-energy on the bottom right-hand corner allows to better displays the 1.15 Hz band, which is the period of the right leg movement (with the SHL 2018 dataset, the phone is kept in the right pocket of the user).

- Secondly, no publication working on the SHL dataset found that cleaning the data helped a classification algorithm. Some works do apply some filtering, but they either do it to accelerate the computation (*e.g.*, subsampling [31, 45]), or they apply the filtering without evaluating this step ([30, 76])

We repeat the preprocessing protocol in [50]. Each temporal signal is first converted into a spectrogram using a moving Short-Term Fourier Transform (STFT) window. The samples are 6,000 points-long *segments* (60 seconds at 100 Hz), and we use 5 seconds-long windows with 4.9s overlap. We obtain spectrograms with 550 points on the time axis, and 250 points on the frequency axis. Then, the spectrograms are rescaled into 48×48 pixels. If the choice of a smaller resolution is done in order to reduce the complexity of the problem, we assume that this precise resolution was chosen to fit exactly the architecture of the network: with the successive size reductions due to convolution and pooling steps, a 48×48 spectrogram fits exactly and leaves no 'leftover pixel line'. The time axis is rescaled linearly, while the frequency axis is rescaled using a logarithmic interpolation (similarly to the mel scale). This allows to give more importance to the lowest frequencies as walking, running, and cycling generate sharp components between 1 and 2 Hz (see chapter 3).

We also rely on the architecture in [50]: the Convolutional Neural Network has three convolutional layers and two fully-connected layers, as fig. 2.16 shows. The convolution filters are 3×3, in order to process the two-dimensional spectrograms. The hyperparameters used are given in table 2.3.

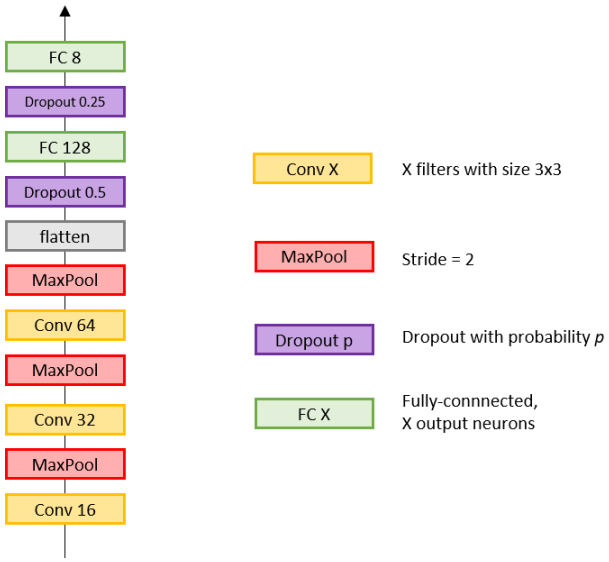


Figure 2.16: The architecture of the baseline SHL model

Training protocol

To generate the comparisons between parameters or choices, the network is trained for 50 epochs on the training set, before being evaluated on the validation set. Each evaluation is repeated 5 times (with a new random seed every time), the mean and standard deviation of the validation F1 score are given as a result. The hyperparameters for the training can be found in table 2.3. Once the hyperparameters are found, we test the best method against the state of the art by training the model with the union of the train and validation set, and evaluating the results on the test set. The results on the test set will appear in chapter 5, after we tried improving on the baseline.

Evaluation of the sensors

The attentive reader noticed that there are several sensors in the GeoLife dataset, and that we did not precise which sensors we would use. Our baseline network uses only one sensor as is (we will address the problem of data fusion in chapter 5), but we will not repeat out experiments with each axis of the seven sensors available.

To know which sensors to choose, we decide to evaluate all the sensors individually by training a network using one sensor at a time. For each sensor available (accelerometer, gyrometer, etc.) we consider every possible axis (x, y, z, with the possible addition of w for the orientation quaternion), in addition to the norm of these axes (computed using the euclidean norm). The norm is hoped to represent an orientation-independent version of the signals. For each signal, we compute a log-power spectrogram, using a log axis. We then evaluate them individually. Table 2.5 shows the norm of the accelerometer is the single best signal available. Even though the accelerometer is the best sensor, the linear acceleration and gravity follow closely. This is no surprise, as those three signals are closely correlated to each other. The non-negligible differences between the x, y, z axes of each sensor might be due to the fact that some orientations of the phone are more likely than others (fig. 2.18). The pressure signal is surprisingly effective at distinguishing between transport modes. This sensor manages to reliably capture the periodic disturbances from walk, run, or bike segments (see fig. 3.6 in chapter 3), and somehow retains enough information to classify most of the other modes (Still, Car, Bus, Subway, Train).

sensor	Acc	LAcc	Gra	Gyr	Mag	Ori	Pressure
x	87.24 ± 0.53	83.97 ± 0.80	85.19 ± 0.26	81.31 ± 0.57	71.14 ± 0.67	73.82 ± 1.24	
y	87.22 ± 0.72	86.22 ± 0.84	84.44 ± 0.22	81.05 ± 1.25	73.63 ± 0.82	74.37 ± 1.25	
z	84.18 ± 0.77	85.36 ± 0.69	83.34 ± 0.54	79.32 ± 1.32	73.17 ± 1.03	75.46 ± 0.51	76.35 ± 0.67
$norm$	$\mathbf{89.14 \pm 0.65}$	81.01 ± 0.50	47.67 ± 3.27	76.52 ± 0.68	66.81 ± 0.47	42.05 ± 0.97	
w						78.54 ± 1.07	

Table 2.5: The validation F1-score per signal. The best result is in bold.

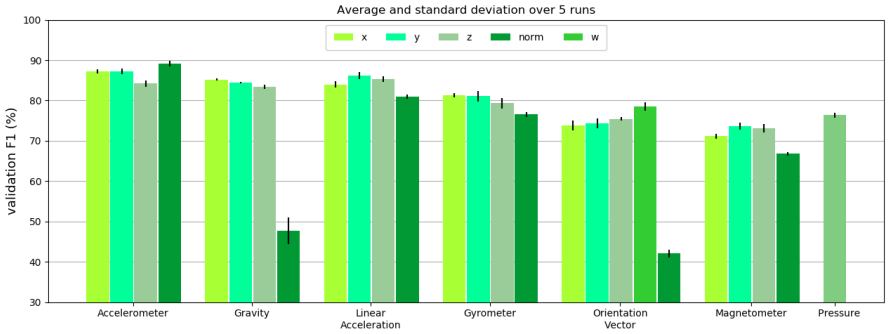


Figure 2.17: A bar plot of each individual sensors corresponding to the results in table 2.5. The error bars denote the standard deviation over five random initializations of the network.

Concerning the accelerometer, the norm of the acceleration vector has a better performance than any of its individual axes, which shows the orientation-independent signal is better than the orientation-dependent components. For the norm of the magnetometer, however, the result is the opposite: each of the individual axes could play the role of a compass, giving information about the way the phone moves, which particularly helps with most dynamic transport modes (Walk, Bike, Run). For instance, if the user is on a bike and has their leg moving with a $2Hz$ period, the y axis of the magnetometer will display a strong $2Hz$ components. When we compute the norm of the magnetometer, we obtain the strength of the magnetic field and loose this compass-like information. However, this is not a problem in our setting, because we always use the norm of the magnetometer in conjunction with the accelerometer signal, which is already efficient enough to classify the classes for which the phones moves the most.

Two other surprisingly high performance signals are the norm of gravity and the norm of the orientation vector. In theory, these two signals are constant (equal to 1 for the norm of the orientation vector, and $9.81 m.s^{-2}$ for the gravity). In practice, these values are never exactly constant: when zooming in on the most dynamic transport modes (Walk, Bike, Run), we can see some periodic patterns (fig. 2.19). The motif is quite noisy, but the frequency of the signal is the same as the fundamental of the individual axes (see fig. 3.6). These patterns may have a negligible amplitude in the temporal domain, but we use spectrograms with the log of the energy: this representation allows us to convey these small patterns to the network. With the other classes, however (Still, Car, Bus, Train, Subway), these signals look like plain noise, and the classifier is almost always wrong.

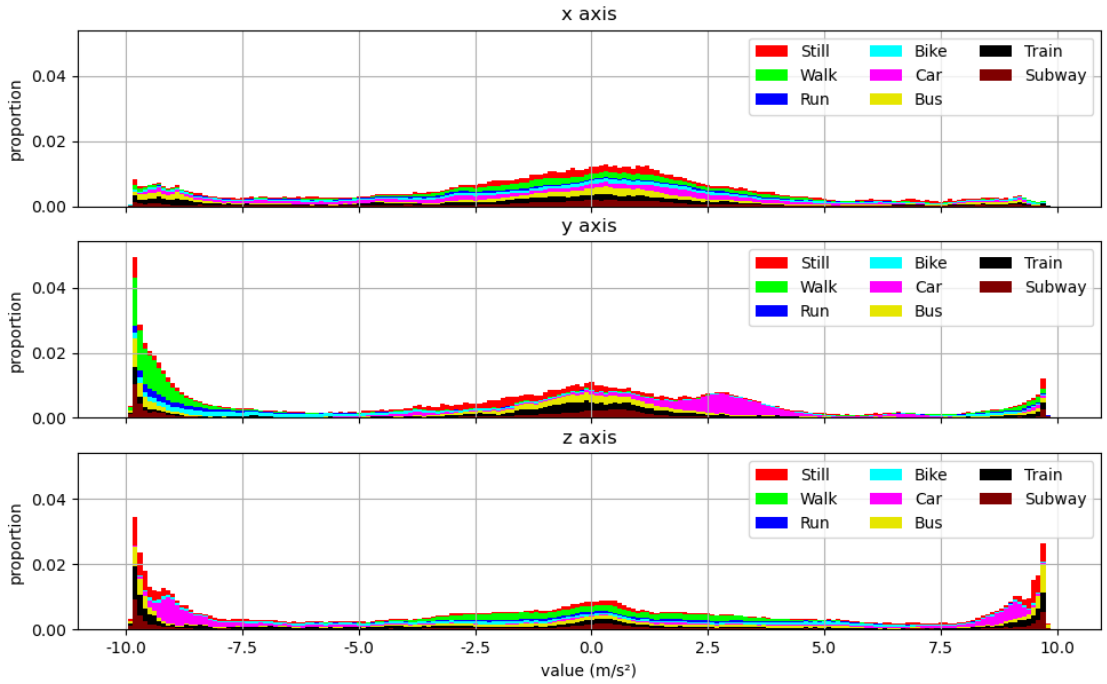


Figure 2.18: An histogram displaying the values of the three axes of the Gravity sensor. We can see that the phone is upright when the user walks (the extreme values in the negative region of the y axis are mainly from walk segments); the phone lies flat, with the screen facing up or down, when the user drives (the car segments are extreme values of the z-axis, both positive and negative); and that the phone is not often on the side (the extreme values of the x-axis are not the most represented). See fig. 2.2 for an illustration of the meaning of the axes. Note that the phone was in the user's pocket when the dataset was recorded.

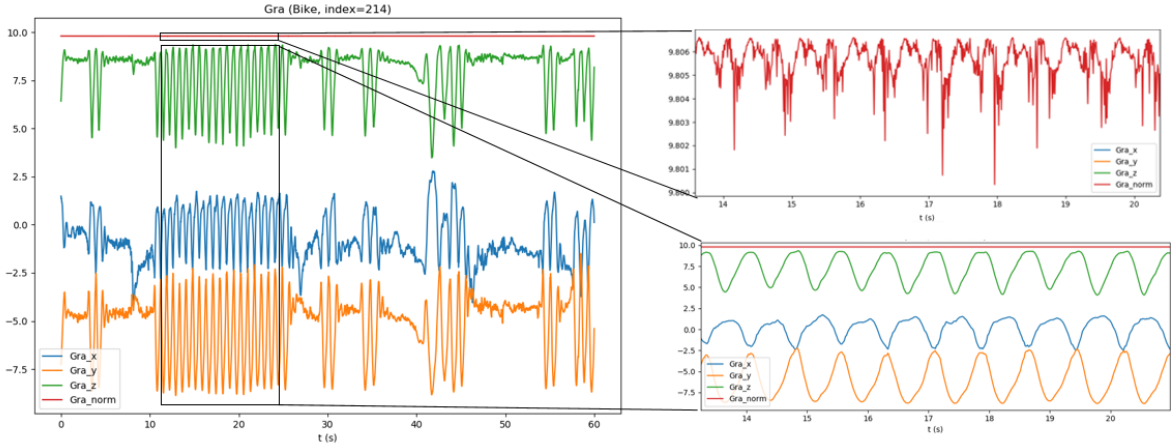


Figure 2.19: An example of signal that shows how the slight noise in the norm of the gravity discloses the class for the most obvious modes (here, a Bike segment).

Sensor choice

We start with the norm of the accelerometer ($|Acc|$), for it is the single best signal available (table 2.5). As in [50], we add the y-axis of the gyrometer (Gyr_y).

But those two sensors only measure the inertial dynamics. To add a different kind of information, we choose to use the norm of the magnetometer. This signal does not change much on the exterior, and varies greatly around metal objects and strong electrical currents. In particular, when the sensor (and hence the user) is inside a car, a bus, a train, or a subway, the value of this signal can jump up to $200\mu T$ (whereas it remains around $40\mu T$, the value of Earth's magnetic field, when the sensor is left outside). One could argue against this choice because when considered alone, the norm of the magnetometer is worse than any of its three individual axes (x, y, z, table 2.5). This is because the axes can act as a compass, thus retaining information about the dynamics of the movement. Computing the norm destroys this precious information. However, this is not a problem in our setting as we always use the magnetometer along with other inertial sensors (accelerometer and gyrometer).

As a sanity check, we add the w component of the orientation vector (Ori_w), to have an example of a virtual sensor. The orientation quaternion is a representation of the orientation of a referential that, contrary to the Euler angles, prevents the loss of a degree of freedom because of the gimbal lock. As indicated by its name, the quaternion has four dimensions, $Ori_x, Ori_y, Ori_z, Ori_w$, computed using the coordinates of a vector around which the phone rotates (see fig. 2.20):

$$\begin{cases} Ori_x = x.cos(\theta/2) \\ Ori_y = y.cos(\theta/2) \\ Ori_z = z.cos(\theta/2) \\ Ori_w = sin(\theta/2) \end{cases}$$

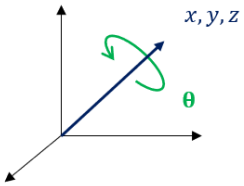
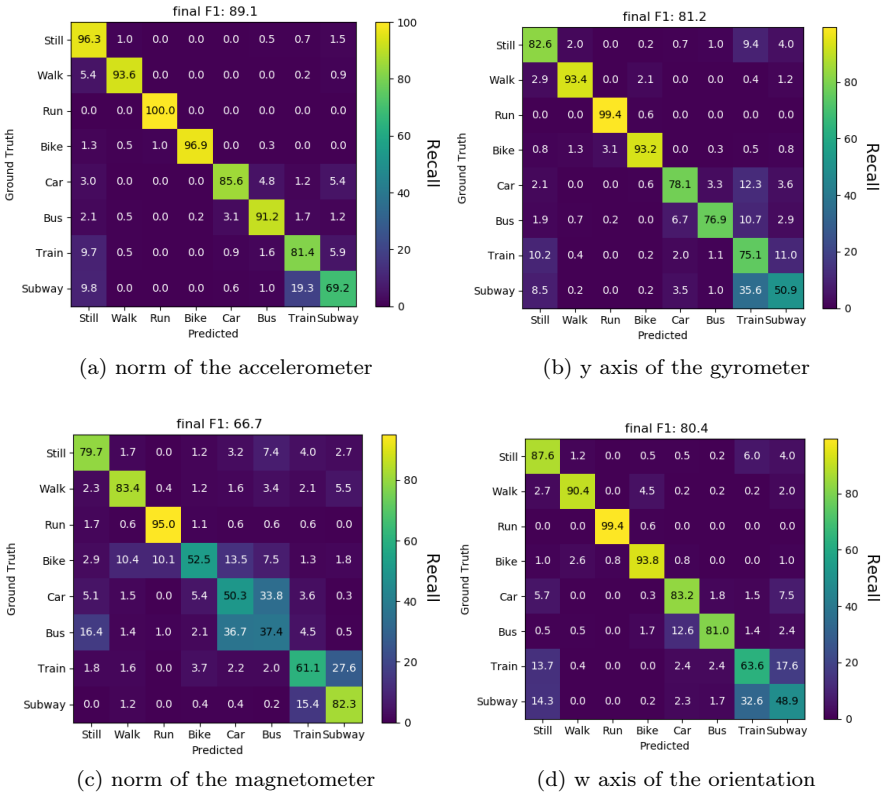


Figure 2.20: The computation of the orientation quaternion.

In short, this signal gives some information about the amount of rotation the phone records. Note that we ignore how is this sensor computed in practice, even though we assume it involves the gyrometer, magnetometer, and possibly the accelerometer. This is why, when we will consider a fusion of different sensors, we expect this signal to carry similar information to at least the gyrometer and magnetometer: adding the orientation to the triplet $(|Acc|, Gyr_y, |Mag|)$ should not improve results significantly.

The confusion matrices of baseline models using $|Acc|, Gyr_y, |Mag|$, and Ori_w individually are displayed in figure 2.21. As with Transport Mode Detection from GPS signals, the performance differs depending on the class to classify: running segments are classifier almost perfectly using the accelerometer (which is not surprising given the simplicity of the problem [140]), while train and subway classes are harder to distinguish from each other. To a lesser extent, these two modes are also confused with the Still class. Depending on the sensors, the Car and Bus classes might also be hard to tell each other apart.



(a) norm of the accelerometer

(b) y axis of the gyrometer

(c) norm of the magnetometer

(d) w axis of the orientation

Figure 2.21: The confusion matrices of models using (a) the norm of the accelerometer (b) the y axis of the gyrometer (c) the norm of the magnetometer (d) the w axis of the orientation quaternion, with the SHL validation dataset.

A focus on the results of the challenge

Contrary to the GeoLife dataset, those who participated in the challenge had to use the F1-score, and their evaluation by a hidden test set guaranteed the comparability of the results.

The most successful methods were the two submissions from the Joseph-Stephan Institute, ranking first and second on the unseen test set [31, 45]. Both approaches relied on computing a broad set of features: statistical (mean, standard deviation, skewness, kurtosis, min, max, etc.) and frequency features (frequency of the highest power component, energy in predefined frequency bands, etc.). These features were computed on raw data, norm, and de-rotated signals (aligned on the North-East-Down coordinates). A feature selection step took place before using the features for classification. The only difference between them is that the approach from [45] used a XGBoost model for classification, while [31] trained several Machine Learning models, along with a Deep Neural Network, to predict the output. Then, a Hidden Markov Model was trained to return the final prediction from the predictions of all the individual models. This allowed them to gain one and a half points: the F1-scores of these approaches were 92.41 % and 93.86 % with XGBoost and the ensemble models (respectively).

The next best participation [50] is the one that we started from for our baseline. Ito *et al.* used spectrograms, with a log-scale for the frequencies. The 'images' containing the log of the power were then given to a 2-dimensional CNN for classification. This pure-deep learning approach is the one we selected, but the participation only used two sensors: the accelerometer with gyrometer. For each classification, the spectrograms from these two sensors were concatenated along their 'frequency' axis, to form a single image that was to be classified by a single network. This approach seems unusual, but chapter 5 explains how this procedure is not as cumbersome as it seems. In the challenge, this approach ranked third, with 88.83 % F1-score on the test set.

Several other participants did submit a prediction to the challenge. Similarly to the Transport Mode

Detection literature, some used traditional ML algorithms with handcrafted features, others relied on Recurrent Neural Networks, or a combination of CNN and RNN. Wang *et al.* [140] published a complete synthesis of the challenge participations along with the results. However, they did not conduct any experiments to assess the importance of each choice that can be made.

After the end of the challenge. Since the end of the challenge, other publications have worked on this dataset. Gjoreski *et al.* [84] improved the model that scored first in 2018 [31]. By adding another neural network to the prediction models, they managed to improve the F1-score on the SHL test set by one percentage point, up to 94.9%.

Using the dataset without adopting the same constraints. Some publications also worked in a similar setting (*ie*, using the same data, without evaluating on the same train/val/test split) as the challenge:

Richoz *et al.* [25] designed a 1-dimensional CNN working on a sequence of FFT segments from the different sensors. Using the inertial sensors (accelerometer, magnetometer, gyrometer), they obtained a 79.4 % F1-score with a 1-dimensional convolutional neural network working on Discrete Fourier Transforms. Even if their evaluation methodology is different, the results will be fairly close to ours (section 3.3, chapter 3).

Qin *et al.* [115] combined handcrafted features with features from a CNN and gives them to a LSTM. The resulting classifier achieved an average F1-score of 98.0 % (this result was recomputed using table 10 in [115]). However, they explicitly stated splitting the train and validation sets using a random split from sklearn, which hides potential overfitting (see section 2.3.3 or [42])

Drosouli *et al.* [85] studied the performance of different Machine Learning models with several sets of features, and obtain a surprising F1-score of 99.5% with k-nearest neighbours on a broad set of features. Here too, we argue that their performance is likely to be overestimated, for three reasons:

- they did not mention having a correct train/val split (section 2.3.3), nor cite the publication that first noticed the problem on the SHL 2018 challenge [42].
- the best classifiers were worse when dimensionality reduction was used, yet, dimensionality reduction is often used to reduce overfitting.
- the best classifier is k-nearest neighbours, an algorithm that relies on explicitly memorizing every sample in the train dataset.

These two examples highlight one fundamental issue for a fair comparison of different methods in terms of performance: the experimental protocol; *e.g.*, dataset, number of classes, recording conditions, *etc.* matters greatly.

2.4 Conclusion

This chapter was the occasion to present the problem of Transport Mode Detection. We began by displaying an overview of the state of the art, and the typical steps a TMD algorithm follows. Then, we reviewed the available datasets, and made the distinction, between those we would use and the databases we would leave for others to work on. Finally, we presented our methodology, along with the networks we will use in our experiments.

The literature review showed that there were many improvements to be made before making valid comparisons. We will try to be cautious to these so that we do not make the same mistakes as our predecessors: we use a hidden test set, a score (the F1) that is not biased towards the most frequent classes, and to give the standard deviation to know whether a result is significant.

The way figure 2.8 illustrated our methodology prompts an idea: one could wonder if instead of optimizing the performance on each of the axes independently, we could formulate a single optimization problem that encompasses the entirety of all possible choices one could make. Alas, this is not an idea that occurred to us at the time we made the experiments. Pursuing this idea might be a possibility for future work.

Now that we have mentioned all the concepts needed to understand our works and underlined all the pitfalls we paid attention to, we can present the experiments in themselves, starting with the choice of an adequate preprocessing for the network.

Chapter 3

Preprocessing

When dealing with the SHL dataset, the input data we work with are series of 1-dimensional data points, but we switched to spectrograms. We based our approach on one of the submissions [50], but did we do well? Are we sure the network cannot learn the best hidden representation itself, as it generally does with images? We could think that a neural network needs no help to learn automatically the best representations. Alas, a neural network is not always optimal. For example, the researchers of Google who solved the protein folding problem this year [151] relied on the extensive use of domain knowledge-related representations.

After a quick example illustrating the shortcomings of one of the preprocessing steps used in the literature (section 3.1), we will focus the rest of the chapter on the use of signal processing treatments. Four ways of representing a signal exist in the literature: using the raw sequence of temporal values, computing the one-dimensional spectrum using the Fourier Transform, using a two-dimensional spectrogram (a time-frequency diagram using the Short-Term Fourier Transform), or a 2D *scalogram* (a time-frequency diagram using a wavelet decomposition of the signal). Given that the raw data and the spectrograms are the two most common representations in the literature, we will focus on these two, and try to answer the question: when should we compute spectrograms, and when should we stick to one-dimensional temporal data? We will begin by a review of the use of each preprocessing with deep learning in different domains in section 3.2. Then, we will use the SHL dataset to establish a comparison between the representations in section Section 3.3. We will see that the spectrograms are better than temporal representations, at least in our case. Finally, we will try to understand why spectrograms are better than raw, temporal data. The section 3.4 will provide some answers: the spectrograms allow to simplify the classification problem.

3.1 Introductory example: How padding segments can disturb the learning process

Before tackling the complex subject of knowing whether the spectrograms are more useful, we will show a very simple improvement to the padding some works from the literature use: if we see publications use zero-padding, we will show we can increase the performance by a few points by padding using a replicated version of the segment itself.

When we use a Convolutional neural network with segments of different sizes (*eg* the GeoLife dataset), we are often required to find a way to get back to fixed-size segments. *Padding* is the action of filling a short segment with well-chosen values so that the segment reaches a desired shape. This action can be required on two occasions:

- When the model requires a fixed-size input [28], the most common approach is to cut the triplegs that are too long, which inevitably generates segments that are shorter than the limit. This requires to pad those segments so that they have the same lengths as the others.
- When the model can deal with segments of arbitrary shape, one problem arises: to accelerate the training, the common practice is to parallelize the computation and to submit to prediction a set of 64 or 128 samples (a batch). This requires putting all segments into a single tensor, which is impossible

Padding	Validation F1
Zero	$77.7 \pm 1.5\%$
Reflection	$80.2 \pm 1.6\%$
Wrapping	$80.3 \pm 1.6\%$

Table 3.1: The validation F1-score of each type of padding. Zero-padding is particularly detrimental to the model performance.

when the segments have different lengths. One could simulate the batch computation by sending each segment one after the other and computing the weight update once after a certain number of segments are processed, but doing so would increase greatly the training times. This is why even with the baseline we presented in chapter 2 (which can use segments of different lengths) we pad all the short segments so that they reach the length of the longest segment in the batch.

There are several ways to pad short segments. One could pad using zero-values (like in [28, 57]), but we will show that this disturbs the learning process. To demonstrate it, we compare this padding to two methods consisting of padding using the data from the input segment itself (see fig. 3.1). We tried padding with a reflection of the segment (adding a reversed copy of the segment after the original), or simply repeating the segment until the maximum length is reached.

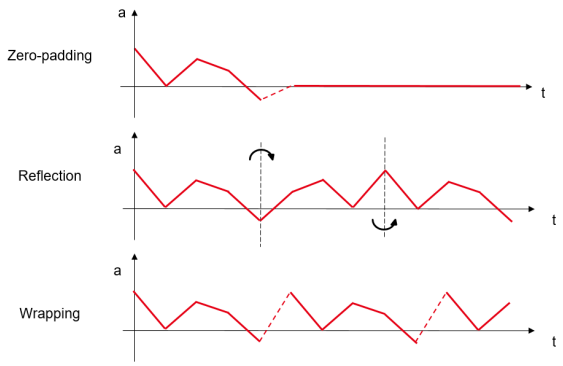


Figure 3.1: The different kinds of padding. Zero-padding simply adds zeros until the maximum length is reached, Reflection reverses a copy of the segment and adds it at the end, while wrapping simply duplicates the segment until the maximum length is reached.

Using the GeoLife baseline architecture, we compared the three kinds of padding shown in fig. 3.1:

- *Zero-padding*, where zeros are added to the shorter segments until they reach the correct length, as in [28, 57].
- *Wrapping*, where segments are padded using their own data
- *Reflection*, which consists in padding the segments using a time-reversed version of the segment itself.

As table 3.1 shows, padding with zeros is particularly detrimental to the performance of our model. However, one can wonder which padding is better between wrapping and reflection. Wrapping creates discontinuities in the data, but this is the only artefact it introduces: otherwise, wrapping only uses data from the segment itself. On the other hand, reflection removes some of the meaning of the data (*i.e,* and acceleration becomes a breaking), but it does not introduce any discontinuity.

In general, padding by wrapping works well when using a global pooling operation: because convolutions can only see patterns locally, if the input data is periodic, the output of a convolution will be periodic, with the same period. When considering the whole convolution blocks, only the layers with a stride different from 1 will affect the period of the features. This means a global max-pooling layer will output the same

result whether the input segment was padded by wrapping or not padded at all. Conversely, zero-padding and reflection-padding create new patterns that might be interpreted as significant by the network. One could think that using a reflection is better than wrap the segment around because the former introduces no artificial discontinuities. However, the GeoLife model has a particularity: during the random search for hyperparameters (appendix C), we contemplated adding a cleaning step using median or Savitzky-Golay filters, and it turns out these filters did not improve the performance. This means the GeoLife network is naturally robust to noise in the data, which is why it is not affected by the discontinuities brought by wrapping the segments around.

One could wonder why zero-padding is worse than the rest. Two mutually exclusive hypotheses could be formulated to explain this phenomenon:

- A long series of zeros is interpreted as being meaningful by the model, and disturbs its *predictions*.
- The model notices the long series of zeros in the learning process, and, upon seeing they are uncorrelated with the segment's classes, somehow learns to ignore the end of a segment during the training process, which cause it to miss relevant information

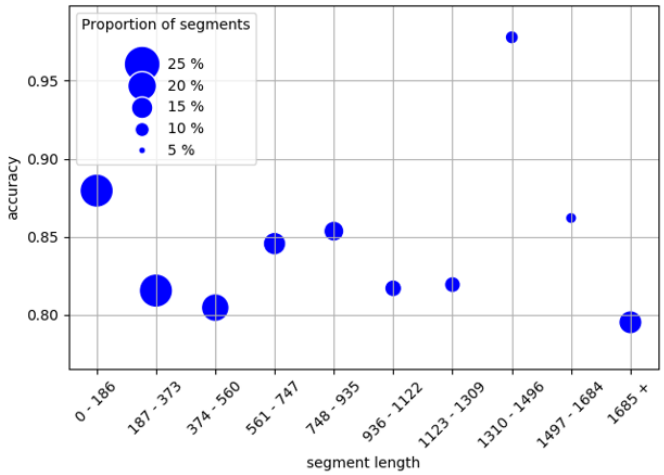


Figure 3.2: The validation *accuracy* versus the size of each segment (the shorter the segment, the more zeros it will be padded with). Adding zeros is not particularly detrimental to the classification performance, which means the network learnt to ignore the zeros, missing potentially relevant information. Intervals bins are obtained using equidistant separations between 0 and the 90-th percentile

To know which one is true, we compute the accuracy for segments with different lengths (using the fact that the shorter the segment, the more zeros it will be padded with). If the former hypothesis was true, we would see the performance to be correlated negatively with the number of zeros. The performance would be correlated positively with the length of a segment, resulting in a decreasing curve when displaying the performance versus the number of zeros. *Note:* in this experiment, and this experiment only, we computed the accuracy, because some bins are devoid of certain classes, and the F1-score is not defined when the number of samples from one of the classes is zero. Figure 3.2 shows that the curve is quite irregular, which means the model learnt to ignore zeros at the end of segments, at the price of also ignoring the end of relevant segments.

Even though some works did use zero-padding, choosing an alternative padding method is an obvious decision when it is clearly exposed as a choice. However, a less clear choice is to know if the network will see raw segments or if we apply some kind of signal processing treatment (*e.g.*, computing the spectrum), and the rest of the chapter is devoted to this preprocessing.

3.2 An overview of preprocessing in the literature

The recent advent of Deep Learning sparked enthusiasm in a diversity of domains. Quantity of researchers tried to use a learning algorithm to replace the old approaches based on domain knowledge. However, this domain knowledge does not always disappear completely. In particular, in domains that involve signal processing, we often see deep neural networks applied mostly on either the raw data, or on spectrograms. This section is devoted to a quick overview of the use of the different representations in different research domains.

If there are four representations we will focus on (raw data, FFT, spectrograms, scalograms), we would want to answer the following question: when is it more interesting to compute a representation for the network, and when should we leave the network alone? In this regard, we will consider that the FFT, spectrograms, and scalograms all consist in computing features along the temporal axis ourselves, while leaving the data unprocessed lets the network learn the optimal features to compute. One key hypothesis that will structure this study: the fact that it is better to let the network learn which temporal features to extract itself when the number of samples is high, and that it is better to compute the frequency features (FFT, spectrograms, scalograms) when the number of samples does not allow the network to find an optimum that generalizes well ([152] fig. 3.3).

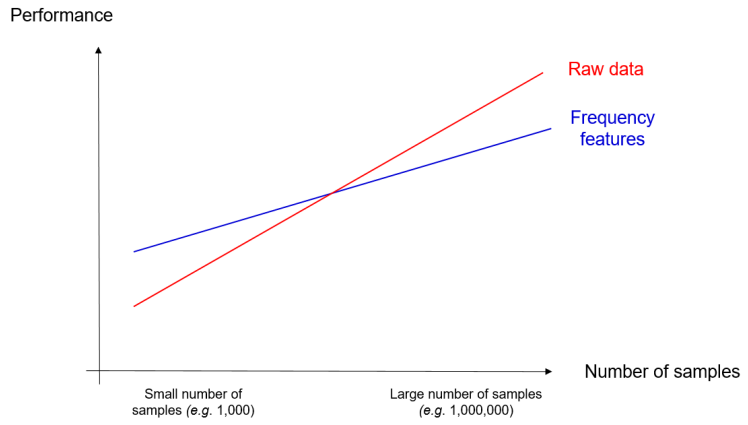


Figure 3.3: The main hypothesis we want to verify ([152]): leaving the data intact is worse when the number of samples is low, and better when the dataset is large.

When looking at figure 3.3, one particular question comes to mind:

Isn't that obvious ?

Many researchers know that classical Machine Learning models, based on handcrafted features, see their performance increase at a slower rate when the number of samples increases than deep neural networks [153]. In other words, when the number of samples increases, the slope of the curve of the performance is higher for deep neural networks than for Machine Learning algorithms relying on handcrafted features. The use of deep learning on FFT, spectrograms, or scalograms could be thought of as a middle ground between handcrafted features, for which an expert selects the interesting values a classifier will work with; and deep neural networks that compute their own features from the immense variety of the raw data.

However, the choice representation does not follow this reasoning: for instance, the Fourier transform is bijective, which means there is, in theory, as much information in the spectrum of a signal as in the original signal. In general, spectrograms often have more pixels than the input segment. We do not say that computing a spectrogram creates new information (there is actually some redundancy in the values of pixels in a spectrogram), but we argue that the idea that frequency representation reduces the amount of data the network works with is not straightforward. What changes between representations is only the semantic meaning of the data points in the segment or spectrum.

In addition, the reasoning we presented in the last paragraph and in fig. 3.3 is, to the best of our knowledge, not the subject of many research works. At first, we will try to use the literature to know whether this hypothesis is verified.

We will rely on summary papers, and when available, on direct comparisons of the performance of different representations. However, some of these comparisons (including the one we will lead in the next section) suffer from a common flaw: they work with a fixed architecture, and many of them do not say how this architecture was found. In our case, when we worked with the SHL Dataset, our baseline architecture comes from a participation that worked exclusively with spectrograms [50]. This means that the architecture is likely to be optimized for this exact representation, and that evaluating 1-dimensional temporal representations with it will result in suboptimal performances. To establish a fair comparison, we would need to reproduce the hyperparameter and architecture search for each representation (similarly to [111]). We could also look at the challenges: as they consist in giving the same time window to several teams to optimize their performance, we could compare the result each representation manages to get. However, one must keep in mind that even the challenges are not perfect, because the number of researchers working on the same participation may vary, and we do not know how much time and computational resources each team devoted to its participation.

When no comparisons are available, we will try to know which representation is used the most often (given that researchers often make many choices that are not written in publications). However, even this information is not easy to gather: many summary papers focus on neural networks architectures and disregard the network input. This information is either hidden in the detailed list of publications the review cites [154, 155], or simply absent from the review [156, 157, 158]. We will also try to mention the approaches that stand out from the rest due to their originality.

The present section does not aim to be exhaustive (there are too many research subjects to cover), but only to provide an overview of the use of spectrograms in the literature. We will try to cover four research domains we chose for their diversity: audio processing (section 3.2.1); analysis or vibration signals for rotating machinery (section 3.2.2); physiological signals (mainly EEG and ECG, section 3.2.3); and Human Activity Recognition (section 3.2.4). Finally, we will detail the use of representations in Transport Mode Detection in section 3.2.5, and conclude on some opening remarks in section 3.2.6.

3.2.1 Audio processing

In 2015-2016, reviews [159, 160] indicate that, for audio processing (whether it is speaker identification, speech transcription, *etc.*), the golden standard was use recurrent models (either Hidden Markov Models or RNNs), applied on features named *Mel-Frequency Cepstral Coefficients* (MFCC) [161, 162]. These features are obtained the following way:

- Compute the Fourier Transform of the signal.
- Compute the energy of the signal in an overlapping series of frequency bands. The ranges of the bands' scale with the logarithm of the frequency (*mel* scale).
- Compute the log of the power.
- Compute the Discrete Cosine Transform (DCT), which plays the role of an "inverse Fourier Transform".

These features have the property that any convolution filtering applied to the raw data can be separated linearly from the signal in the feature space, and they represent an example of features that are specially handcrafted for a specific domain.

During the last years, reviews noticed an increase in the amount of research works using deep neural networks [163, 164]. Some networks use the raw data [165, 166, 143], but most of them rely on some kind of spectral features such as spectral bank features [167, 168, 169], or spectrograms [170, 171]. However, the use of MFCCs still occurs in more recent works [172].

We did study two speech recognition challenges in detail: the CHiME challenge (both the fifth edition [173], and the sixth [174], for they are extremely similar), and the Airbus challenge [175].

For the fifth CHiME challenge, the organizers provided a baseline relying on MFCCs [176], and the participations that improved the results on one of the tasks either build upon this baseline ([177]), or

rely on complex knowledge-based models. For our subject of interest, no deep learning model on raw data outperformed the MFCC-based models. A similar conclusion can be drawn from the sixth CHiME challenge, for which none of the participants managed to outperform baselines using MFCCs with GMM and HMM ([174, 178]). Both challenges had about 100,000 *utterances* (an utterance is an uninterrupted sequences of words, or even sentences, which is used to count the number of samples in a speech dataset).

This conclusion also applies to the Airbus challenge [175], where the participants had to perform Automatic Speech Recognition of sentences pronounced by airport controllers and pilots, involving both technical terms and non-native speakers. The best submissions in this challenge used MFCCs, along with other problem-specific features we will not present here. The Airbus challenge has 50 hours of data, which represents about 18,000 utterances if we estimate an utterance to last 10 seconds. For both the Airbus and the CHiME challenges, the dataset size is too small for deep learning to even start being relevant.

In addition to the challenges, a publication performing different comparisons is extremely important for us. Pons *et al.* [152]. tackled audio tagging for music, a task where the model must choose appropriate tags to characterize a song: the tone, the style, the type of instruments, *etc.* This can be considered as a classification problem where we can assign multiple classes to any sample. They used prior knowledge to design two architectures, one using spectrograms, and one using temporal data. They concluded that the raw data had the best performance, but the most important conclusion is when they used smaller portions of the dataset: if the original dataset had a million songs for 50 tags, they tried using only 500,000 and 100,000 songs to train the models. With the smaller versions of these training sets, the two architectures perform identically. They concluded by saying that the raw, temporal data are better when the number of samples is high (when the network can use the mass of data to extract the features itself), but the spectrograms are comparatively better when the number of samples is low, and the network needs help extracting the relevant features. However, their results are not extremely significant, and the difference between the increase rates (the difference between the slopes of the red and blue curves in fig. 3.3) are hard to observe in the results they provide.

Finally, we should mention the works of [143], who designed a convolutional network whose first layer was particular: instead of using the classic, discrete convolution, the values of the first convolutional filters are computed using the formula of a given wavelet. Only the two parameters that compute the wavelet are learnt by gradient descent. Their network, dubbed SincNet, is able to focus on the frequencies relevant to speech understanding better than the classic CNN. Also, when the authors added noise in a specific frequency band, SincNet ignored the noise faster than the classic CNN. For us, the most interesting is not their network, but the following derived result: at the end of the training, the classic CNN did learn to ignore the noise extremely well: the average Fourier spectrum of the first convolutional filters displays a low power at the frequencies in which the noise was added. In other words, the first layer of a CNN with discrete filters can 'see' the interesting frequencies in the signal and ignore the irrelevant frequencies. Provided, of course, that this layer is long enough to capture interesting frequencies (the layers were 250-points long in the original architecture).

If we could not draw any conclusions from challenges, a review of the literature indicates that both spectrograms and temporal representations are used. One paper showed that the raw data is better than the spectrograms when the dataset has more than $500,000$ samples, and one of them tell us that one-dimensional convolutions on raw data do notice the interesting frequencies by themselves, and learn to ignore the useless ones. This seems to indicate that one-dimensional convolutions could effectively re-learn the same features like the ones we compute with spectrograms, provided the number of samples is high enough.

3.2.2 Failure prediction in rotating machines

Rotating machinery is widely used in a large number of industries. Given how extensive the utilization of this equipment is, some researchers try to design algorithms to predict a breaking from acceleration signals. In order to produce a dataset, the typical method is to use an experimental testbed: the researchers keep an engine functioning in a laboratory until one of the roll bearing breaks. Meanwhile, they place a sensor (most often an accelerometer) on the engine (or close to it), and record the vibration signals. They obtain the datasets of 1,000 to 30,000 samples in total [179, 180, 181, 182, 183], and add labels to solve one of two tasks:

- predict the Remaining Useful Life (RUL) of the bearing using the acceleration signal: if, for instance,

the network is presented a segment recorded seven hours before the bearing breaks, it should predict that the engine can keep running for seven hours before a preventive repair is needed. This is a regression problem, as the objective is a continuous value.

- predict the exact source of the fault (diagnosis), which is a classification problem.

In addition to the classic Machine learning, producing handcrafted features inspired by signal processing knowledge, this field has seen the use of several neural networks for automatic classification. According to reviews [155, 184], if some approaches use raw, temporal data (such as [185]), most networks work on spectrograms, or on scalograms. We also see the use of the Hilbert–Huang Transform (HHT, [186]), a signal processing operation aiming at finding the instantaneous frequencies of harmonics in the signal. We did find one work using convolutions with the one-dimensional FFT [187], which is rare because most publications use either the raw, 1-dimensional temporal representation or the two-dimensional spectrograms/scalograms.

Most of the publications work using only one type of wavelet, but [188] did a comparison of the performance of a network trained on scalograms (Morlet wavelets), along with spectrograms and concluded that the scalograms were the best, then came the spectrogram, and the HHT was the worst. Another comparison is the one from [179] who designed a network similarly to [143]: the first convolutional layer uses continuous filters from a diversity of wavelet families, and the scale parameter of these wavelets are learnt by gradient descent. They compared different families of wavelets to the classic CNN using discrete filters, and concluded that the Laplace wavelets were the best, significantly above the classic CNN (which was on par with the other families they evaluated: Morlet and Mexican hat wavelets). Finally, the sinusoidal network (the network using continuous filters based on the Fourier basis) was significantly worse than the rest. However, they did not draw any conclusion relating these results to physics. The Airbus challenge has 50 hours of data, which represents about 18,000 utterances if we estimate an utterance to last 10 seconds. of their problem.

Sadly, we could not find any comparison involving the temporal representations. The fact that spectrograms and scalograms are more popular than raw representations [155] might be an indication that these representations are better, but it might also be due to the fact that researchers tried the spectrograms more often due to prior experience with these diagrams. In this domain, we cannot conclude whether the hypothesis we presented is verified.

3.2.3 Physiological signals

The human body generates a great number of signals: heart rate (ElectroCardioGram, ECG), brain waves (ElectroEncephaloGram, EEG) *etc.* Many research works try to design automated diagnostic models that use these signals to know if a given patient suffers from a disease, or sometimes to try to predict the state of the patient (sleep, emotions, *etc.*).

Both approaches are present: the use of raw signals [189, 190, 191, 192]; as well as spectrograms [193, 194] or scalograms [195]. For EEG signals, reviews [196, 197] indicate that both preprocessing methods are used in similar proportions. However, this conclusion does not apply uniformly to all types of problem. For instance, in EEG for brain-Computer Interface, most works using deep networks focus on spectrograms [198].

The extensiveness of the theoretical background surrounding these signals allows the authors to rely more on expert knowledge to create new preprocessing methods. For instance, with EEG processing, different families of frequencies are known to exist in the brain [199, 197]: alpha ($8-13Hz$), beta ($13-30Hz$), *etc.* This pushed some authors to create a preprocessing where they compute the energy in predefined frequency intervals. For each frequency band, they project the recordings of the power from the different electrodes on a 2D plane, using a projection that translates as best as possible the physical position of the electrode on the patient's head. They obtain one "image" per band, and stack the images like channels in an RGB image. This image is then given to a 2D CNN [200, 201] for classification. In this example, the role of the CNN is mainly to extract *spatial* features, because the convolution filters run on the two dimensions of the image, and not along the temporal axis. Hence, it would make sense to consider this preprocessing to belong to the *spectrograms* category, for the temporal features (the type of feature encoding we are interested in) are extracted when the energies in different frequency bands are computed.

However, the reasoning we made comes from the very specific need to know how the temporal features are computed: the existing reviews do not categorize this preprocessing as spectrograms. A thorough analysis would review each paper in detail, to know exactly where does the paper fit in, but such a work is too vast

for us to pursue, and we will only rely on the number of raw and spectrograms/scalograms publications to know whether the networks compute the temporal features themselves or not. In addition, some works are quite hard to classify, such as the creation of a 2-dimensional matrix symbolizing the similarities between two points of the temporal input signal [202].

Another hurdle to the proper accounting of the number of representations is that this number does not depend only on the performance of each method. For instance, with ECG processing, spectrograms are used less often [203], because the temporal signal is composed of an extremely stereotyped motif, and no wavelet family corresponds: trying to encode an ECG recording with wavelets would scatter on many different frequencies, even though the signal consists in repeating the same pattern with varying frequencies. Hence, the frequencies we observe with spectrograms using such bases do not translate the frequency of the heartbeat, but depend more on the shape of the motif [204]. However, we argue that this kind of information might still be useful, because one study [194] does report that 2D convolutions on spectrograms are better than a heavily optimized one-dimensional CNN for heart disease classification, on a dataset with 2,500 samples and five classes. This pushes us to question whether the absence of spectrograms (with this database at least) is due to a worse performance, or if this preprocessing was merely left unstudied because of prior reasoning.

Finally, Hussein *et al.* provide us with another evaluation that is interesting to our case: they compare the raw data to *scalograms* (they argue that scalograms are better than spectrograms). They do not mention which family of wavelets they used to compute the scalograms, but the comparison is still useful to us: they evaluated many architectures from Computer Vision (ResNet, AlexNet, VGG, *etc.*) on these scalograms, and all of them are significantly better than the one-dimensional convolutions on raw data. They used three datasets with about 100,000 samples each to detect epileptic seizure, a two-class imbalanced problem.

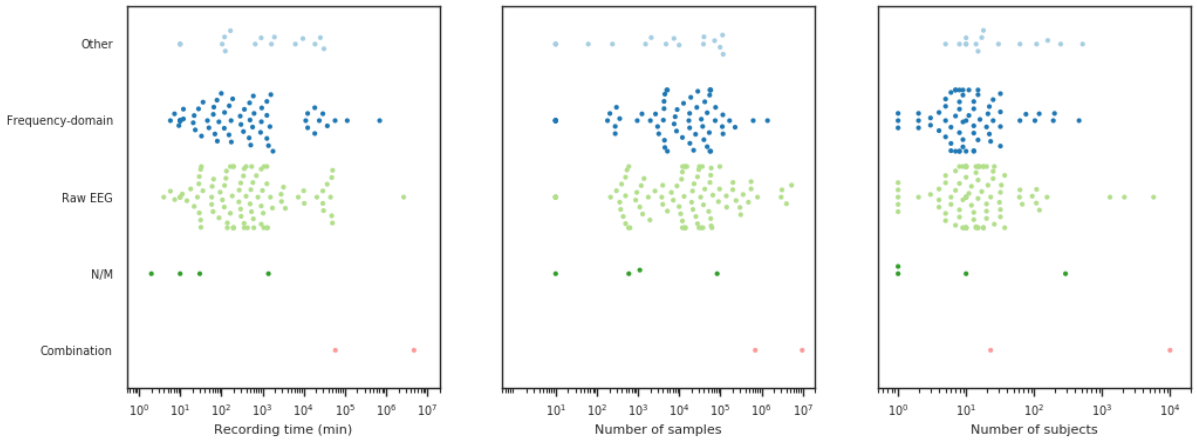


Figure 3.4: The type of preprocessing used, depending on the total duration, number of samples, or number of subjects of the dataset (N/M: Not Mentioned; combination denotes the use of several types of features). Figure generated using the code and data from [196]. We would like to thank Y. Roy for providing us access to the code.

In this domain, it seems that both approaches coexist, and figure 3.4 shows that the choice of a preprocessing method is not related to the size or complexity of the dataset. However, the extensiveness of this research domain prevents us from reading all the publications in detail, which hinders the analysis.

3.2.4 Human Activity Recognition

Human Activity Recognition (HAR) is a classification problem in which the classifier must distinguish between the activities, in a broad sense, of a subject. The classes are among the most common activities a random person can take: Walking, running, sitting, going upstairs or downstairs, playing sports, *etc.*

As transportation is part of the daily routine for most of the population, TMD can be seen as a subset of Human Activity Recognition, and some HAR datasets include classes that are transport modes (Walking,

being in a vehicle, running, *etc.*). However, the low granularity of the transport modes is often too low to use these bases for TMD, or at least without adaptation.

Many types of sensors are used for the problem, and most of them are not interesting for the question at hand: RGB or depth map videos, point clouds, position of a skeleton of the individual, etc. We will focus on accelerometers, for which computing a spectrogram makes sense.

The reviews we found [185, 205] do not even acknowledge the use of spectrograms. It is true that the majority of the publications use one-dimensional convolutions on raw signals [206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 207].

Since the publication of the reviews, some works did use spectrograms [217, 218]. However, some publications that use spectrograms are not usable for our problem, because the employed network is not known to extract features from the data (*e.g.,* Alsheikh *et al.* implemented a Deep Belief Networks [219]). Our problematic is about *deep* learning, and we want to know if the network is able to extract relevant features from the data. If the network used is unable to extract features from itself, it is obvious that the raw data will be inadequate.

With deep convolutional networks, a preprocessing that is surprisingly common is the stacking of one-dimensional segments to form a two-dimensional image [211, 220, 221]: for instance, if we start from six one-dimensional signals with $T = 100$ samples each, we obtain a 6×100 matrix to perform 2D convolutions on. In a similar fashion, [220] stacks the 1D Discrete Fourier Transform of segments into a 2D matrix which is then given to a 2D CNN. In some situations, the stacking occurs along three dimensions: Zheng *et al.* [221] uses signals from eight triaxial accelerometers. Instead of creating a one-channel $T \times 24$ image, they create a $T \times 8$ image with three channels, each channel being one axis of the sensor (x, y, z). Concatenating the signals into an image to perform two-dimensional convolutions might seem weird at a first sight, because the order of the signals has an influence. This is why Jiang and Yin [220] also introduced an algorithm that replicates the input signal and orders them in such a way that every couple of signals is represented exactly once. For instance, if we wanted to create a 2D matrix out of nine one-dimensional signals (which we will call $1, 2...9$), their algorithm would return the following concatenation of signals: $1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 3, 5, 7, 9, 2, 4, 6, 8, 1, 4, 7, 1, 5, 8, 2, 5, 9, 3, 6, 9, 4, 8, 3, 7, 2, 6, 1$. Even if the network uses 2D convolutions, we will consider this preprocessing to be equivalent to a one-dimensional convolution on raw segments.

There are also works that introduce their own preprocessing: Some [222] use images called recurrence plots [223], two-dimensional matrices where the (i, j) indicates a measure of the distance between the samples x_i and x_j in the segment. Memmesheimer [224] simply plots the signal (using libraries such as `matplotlib`), records the image, and applies a CNN to solve an image classification problem. Arigbabu *et al.* [225] start by reshaping the temporal segment into a 2D matrix (samples 1 to 10 go into the first column, samples 11 to 20 go in the second, *etc.*), and compute the *two-dimensional* FFT of the matrix. They compare their results to a 1-dimensional CNN working on raw data and show their method is better on a six-class, 10,000-samples dataset. The diversity of the preprocessing methods make the classification into clear categories harder. One might be tempted to argue to classify some of these methods into one category or the other, but, as these method are quite uncommon in the literature, and as they do not appear to be competitive in any direct comparison, we will simply not consider them any longer.

Finally, to complicate things even further, we also see publications which mention using a 2D CNN, but without giving the actual preprocessing they used [226]. For others, the lack of details even prevent us to know the type of the network in unexpected ways. For example, Chen and Xue [227] concatenate three one-dimensional Discrete Cosine Transforms of the signals into a two-dimensional matrix, and compares three filter sizes for the first layer: 13×1, 13×2, 13×3. Given that the input 'image' has a resolution of 256×3, we could think that the last filter size makes the network equivalent to a 1-dimensional CNN. However, the first convolutional layer can be configured to use padding. If the layer did use a nonzero padding, this leaves room for the 13×3 filter to move along the second dimension, and the first layer is effectively 2D. If, on the other hand, there is no padding, the network is actually equivalent to a one-dimensional CNN, where the input has three channels.

However, we can find three comparisons that are useful to us: Gholamrezaii *et al.* [228] compared the stacking of 1D raw signals and stacking of the FFT of these signals. A two-dimensional CNN performed better with the FFT on a dataset of 10,000 samples and eight classes. We should mention that they split

the datasets by users, which means their results are trustworthy.

The second comparison is the one made by Hur *et al.* [218], who used datasets with thousands of samples and six to twelve classes to compare four preprocessing methods from the literature, and they concluded that their performance order with CNN was the following (in decreasing order):

1. 2D concatenation of one-dimensional temporal signals
2. Spectrograms
3. Signal plot (with *e.g.* `matplotlib`)
4. Recurrence plot

They also introduce their own preprocessing, a modification of the concatenation of input signals, which improves slightly the results. However, we argue that their work is dubious, for reasons similar to the ones we developed in chapter 2: not only they did not mention splitting by user (there are tens of users in every dataset, all with the same class balance) or in a time-consistent fashion, but, the separation of continuous temporal recordings into fixed-size windows is done with non-negligible overlapping (33 to 50 %, depending on the dataset). This means the networks they trained have a chance of overfitting. The overfitting is even more plausible if we know that they claim to reach a performance of $100,00\%$ on one of the four datasets they considered.

Lastly, Zheng *et al.* used a 2D CNN on three types of preprocessing [221]: an image plot (*e.g.* `matplotlib`) of the signal, a spectrogram, and a concatenation into an image. They used a dataset of 7,000 samples and eight classes and concluded that the spectrogram and concatenation were equally important, and that the concatenation they used was better. One could argue that the fusion method is they used for spectrograms and signal plots is questionable: in both cases, they computed one image per axis (x, y, z) per sensor and put the images side by side. In this case, the width of the image, for instance, denotes both the time and the axis (x, y, z) of the sensor. We could think that the concatenation of 1D signals into an image is the only method where the concatenation axes (depth/channels for x, y, z and height for sensor) do not already have a meaning to understand the signal; and that this is why it is better. However, we will see in chapter 5 (fig. 5.2) that the fusion method for spectrograms and image plots is actually relevant. Here, the authors explicitly mention using non-overlapping windows, and the splitting of the samples into training and validation sets follows the chronological order (similarly to the process we explained with the SHL dataset). This means their work is likely to be trustworthy.

To conclude the part on HAR, we should say that even though most of the publications work with convolutions along the temporal axis, some publications using 2D convolutions on spectrograms do exist, even though they have been found slightly worse than the temporal representation.

3.2.5 Transport Mode Detection

We devoted a section to point-level feature computation in the second chapter, but this time, we will focus especially on temporal representations versus frequency features.

With GPS signals, we argue that all works relied on raw representations, despite the fact that no network used the raw latitude and longitude as an input to their networks. One could argue that computing a speed and acceleration is different from using raw data. However, this section primarily makes the distinction between the temporal representation and the frequency ones (FFT, spectrograms, scalograms). Given that the real-time speed and acceleration have a similar meaning to the real-time position, we think that computing the speed and acceleration still counts as using the temporal representation. If it is theoretically possible to estimate the spectrum (and thus, to compute a spectrogram) from irregular data (such as the GPS signals), we found no mention of a deep neural network trained on GPS spectrograms. As a matter of fact, some publications use spectral features to feed a Machine Learning classifier [71], or computed the coefficient from the wavelet decomposition in conjunction with the features from a neural network using temporal data [109], but these methods are the minority.

With inertial sensors, the small amount of publications simplify our analysis: to the best of our knowledge, no publications used deep networks with the TMD dataset [19]. This leaves only the three SHL challenges, which we will present in detail.

SHL 2018

In the SHL 2018 challenge, the best submission [31] used statistical features, feature selection, and an ensemble of Machine Learning classifiers. However, we are not interested in knowing whether Machine Learning outperforms Deep Learning, which is why we will systematically ignore the submissions that do not use a deep neural network in all three SHL challenges. In 2018, five submissions optimized the architecture to use raw (temporal) representations, and even the best of them (83.2% test F1-score, [140]) was 5 percentage points under the only submission that used spectrograms (88.8% test F1-score, [140]). This implies that the spectrograms bring something unique, allowing them to exceed the effect of architecture, or sensor selection. Thus, for our tests, we expect the difference between temporal and spectrogram representations to be superior to 5 percentage points.

SHL 2019

In 2019, the results are not so clear: the first deep submission did use both the FFT and the raw data, but they reported that the results were almost equivalent when only the FFT was used [120]. If we look only at deep learning approaches, the two submissions that used spectrograms ranked third and sixth [141], while the two submissions that used convolutions on temporal data ranked second and seventh. Two submissions ranked fourth and fifth with LSTMs on raw data, but we are not sure if we should take them into account: contrary to CNNs Recurrent Neural Networks are not known to extract features efficiently from raw data.

In 2019, one team [111] compared the performance of 2D convolutions on spectrograms, 1D convolutions on raw data, and LSTM on raw (temporal) data. If we are not interested in the fact that the LSTM was the best by a small margin, the fact that the spectrograms and raw representations have the same performance on this $3 \times 200,000$ samples database is valuable to us: it means that the intersection between the blue and red curves in fig. 3.3 is located around this number of samples. In addition, the final validation performance they report is fairly close to the test result after the challenge, which make their work more trustworthy. However, this team used only the Accelerometer and Linear Acceleration (Acceleration minus Gravity) for their tests.

SHL 2020

Like the previous year, the results of the 2020 challenge are hard to interpret. The winners of the challenge were the only ones to use some of the data from the unseen users as training data (the organizers intended this data to serve for validation only), which is why we will not consider their submission. The second, fourth, and seventh deep submission used spectrograms, while the third, fifth, and sixth used raw data. In this challenge, there do not seem to be a clear influence of the type of preprocessing on the final performance.

In short, for the SHL 2018 challenge ($16,000$ samples), the results clearly indicate that spectrograms are better-suited. For the 2019 and 2020 challenges ($3 \times 200,000$ and $4 \times 220,000$ samples, respectively), both representations seem to have equal performances.

3.2.6 Conclusion of the literature study

During this study in section 3.2, we tried to know whether it was best to compute the temporal features using signal processing operations (FFT, spectrograms, scalograms) when the number of samples is small, and whether a network would learn better features with the raw data when the number of samples is large. We looked at three types of indications: challenge organizations, direct comparisons, and tendencies in the research domains. The frequency of use of the preprocessing methods in diverse domains (table 3.3) does not seem to be extremely helpful in our case: some domains (*e.g.* the study of rotating machinery) rely manly on spectrograms, some (such as ECG or HAR), on raw data, and some of them, finally, see both representations be used, without a clear link to dataset size (as fig. 3.4 illustrates for EEG). The only challenge that we could consider effectively were the SHL challenges, where the 2018 edition did see the spectrograms yield better results, and the 2019 edition seemed to imply that the two representations are equivalent. This seems to confirm the conclusion by Pons *et al.*: the first challenge took place with a "small" dataset, while the two following editions used a dataset for which the two representations are similar. The different comparisons we found are presented in table 3.2. The only domain for which we can say anything is the Human Activity

Recognition, where the hypothesis from Pons *et al.* does not seem to be verified. Even is we except the works from Hur *et al.* [218] because of doubts we have on its rigour, it seems that computing frequency information does help when the number of samples is less or equal to a few thousands. Note that even if Gholamrezaii *et al.* did not evaluate spectrograms but the stacked Discrete Fourier Transforms of the signals, given the high difference between the performance of DFT and raw data (about ten percentage points on accuracy scores), we assume it is unlikely that spectrograms are so much worse than the DFT that they are actually inferior to the stacked segments (the two other comparisons found a difference of performance between two and five percentage points between the methods).

Ref.	input type	problem	which representation is better ?	number of samples	number of classes
[152]	Audio	music tagging	Temporal > Spectrograms	1,000,000	50
			Temporal = Spectrograms	500,000	50
			Temporal = Spectrograms	250,000	50
[111]	Accelerometers	TMD	Temporal = Spectrograms	$3 \times 200,000$	8
[218]	Accelerometers	HAR	Temporal > Spectrograms	50,000	9
			Temporal > Spectrograms	20,000	12
			Temporal > Spectrograms	10,000	6
			Temporal > Spectrograms	7,000	8
[228]	Accelerometers Gyrometers	HAR	FFT > Temporal	10,000	8
[221]	Accelerometers	HAR	Temporal > Spectrograms	7,000	8
[229]	EEG	Epilepsy detection	Scalograms > Temporal	144,000	2
			Scalograms > Temporal	100,000	2
			Scalograms > Temporal	126,000	2
[194]	ECG	Heart disease classification	Spectrograms > Temporal	2,500	5

Table 3.2: A summary of the publications that compared the one-dimensional convolutions on raw data to two-dimensional convolutions on spectrograms, scalograms, or the Fourier Transform of the signal. The lines are ordered by number of samples, approximately

problem	Which representation is the most common ?	Typical number of samples
Audio	Spectrograms & Temporal	1,000 to 500,000
EEG	Spectrograms & Temporal	1,000 to 1,000,000
ECG	Temporal	500 to 10,000
Rotating machinery	Spectrograms	1,000 to 30,000
HAR	Temporal	3,000 to 50,000
TMD (GPS)	Temporal	10,000
TMD (inertial sensors)	Spectrograms & Temporal	10,000 to 500,000

Table 3.3: A summary of the most common representation found in every domain we studied.

However, the fact that these works use different datasets means that we cannot compare them as-is: the threshold number of samples (after which the raw data is better) might change from one domain to another,

and even between datasets. We argue that this is because measuring the number of samples is limited, for two reasons:

- It does not take into account the complexity of the problem to solve. 50,000 sample are enough to learn to recognize written digits in images, but certainly not for general-purpose, ImageNet-like, image classification. We sometimes see a number of samples per class (we often see that neural networks should be preferred to traditional Machine Learning models when the number of samples is higher than 1,000 per class, because it is the number of images per class in ImageNet [230]), but this is still not enough: for TMD, classifying Run segments versus Still ones is much easier than, say, Train versus Subway segments.
- The definition of what constitutes 'one sample' is not as clear as it seems. When looking at the mathematical theorems that guarantee a convergence when the number of samples goes to the infinity, all of them state that the samples must be *independent* from each other. To give an example, duplicating a dataset as is doubles the number of samples, but it does not bring any additional information. This has very practical consequences: when preparing the SHL 2019 challenge, the organizers decided that the classification would be done on 5-second samples instead of 60-second ones. As a consequence, the SHL 2019 challenge has 12 times more "samples" available for each position. Does it means that the available information was multiplied by 12 in this operation ? We think not[1]. Even in the SHL 2018 dataset, the high temporal regularity of the samples [42] means that the samples are far from being independent: we said there were 13,000 labeled samples available to train the model, but as these samples are redundant, if we were to filter the samples to keep only a subset of mutually-independent segments, the number of samples should be lower. Note that this reasoning applies mainly to temporal data (especially when several samples are extracted from an single sequence, such as the SHL comparison or the EEG and HAR comparisons in table 3.2), to images extracted from a video sequence [231], or to multi-user data (a dataset where a single user is present carries less information than a dataset with a representative set of users). Finally, we sometimes see researchers performing data augmentation, and counting the number of augmented samples in the set [232] (for instance, adding the horizontal reflection of RGB images in a 10,000-sample dataset results in a 20,000-sample dataset). We do not know if the augmented versions of already-existing samples bring more or less information than new, unseen samples.

Counting the samples is not incorrect, but these reasons explain that the number of samples is both problem-specific and dataset-specific: if one drew a conclusion on a given dataset (for instance, finding the exact number of samples for which using spectrograms is equivalent to raw representations), these findings would likely not hold if they changed the objective (the classes to recognize) or the data collection process (the diversity in the samples).

Using the literature, we tried confirming the hypothesis from Pons *et al.*, according to which giving the raw, temporal segments to a network is better than computing spectrograms when the number of samples is large enough, but the spectrograms yield better performances when the number of samples is low. If the use of each preprocessing method in the literature does not answer our question, the direct comparisons and the challenges indicate that the hypothesis seems to be true. However, we did not find a single reference proving the hypothesis by itself, as the work from Pons *et al.* does not exhibit figure 3.3 in its entirety. This is why we will make a comparison ourselves.

3.3 Evaluation of preprocessing methods

This section presents the comparisons of the spectrograms, FFT, and raw data representations on the SHL 2018 dataset. We compare the different methods to preprocess four signals from the SHL dataset: $|Acc|$, Gyr_y, $|Mag|$, Ori_w. We consider the 1-dimensional temporal data, and different kinds of spectrograms. For each signal, we also compute the power spectrum using the Fast Fourier Transform (FFT), to obtain the power associated to each of the 6,000 frequencies. As the input signal was real, we obtain a symmetric curve,

[1]The following editions of the challenge still contain more data because the organizers added data from other locations (backpack, torso, hands) to the samples from 2018, which were recorded with the phone in the user's pocket. However, we argue that the mere splitting into shorter segments does not bring significantly new information.

which will go through a 1-dimensional CNN. This is intended to be halfway between the one-dimensional temporal representation and the computation of frequencies of the spectrogram representations. For the results in the 'temporal' and 'FFT' categories (and for these results only), the network uses 1-dimensional convolutions, with filters of size 3. All the other parameters remain similar to the parameters presented in chapter 2 and, in particular, the baseline network working on spectrograms relies on 3×3 convolutional filters.

mode	interpolation (frequency axis)	power scale	size (T, F)	$\|Acc\|$	Gyr_y	$\|Mag\|$	Ori_w
temporal			6000	70.20 ± 1.63	64.71 ± 2.74	7.49 ± 10.32	39.65 ± 2.26
FFT			6000	80.57 ± 1.30	74.30 ± 0.72	55.99 ± 1.53	64.27 ± 1.57
spectrogram	none	linear	550, 250	84.55 ± 1.03	71.81 ± 0.64	63.64 ± 0.96	2.29 ± 0.00
spectrogram	none	log	550, 250	$\underline{87.88 \pm 0.68}$	$\underline{79.89 \pm 1.30}$	$\underline{64.81 \pm 0.85}$	43.35 ± 33.56
spectrogram	linear	linear	48, 48	81.98 ± 0.75	58.42 ± 1.31	45.97 ± 1.88	2.29 ± 0.00
spectrogram	linear	log	48, 48	86.33 ± 1.00	77.06 ± 1.32	56.53 ± 0.68	$\underline{75.03 \pm 2.08}$
spectrogram	log-freq.	linear	48, 48	84.46 ± 0.63	69.19 ± 0.89	53.36 ± 1.41	2.29 ± 0.00
spectrogram	log-freq.	log	48, 48	$\mathbf{88.83 \pm 0.71}$	$\mathbf{82.64 \pm 0.68}$	$\mathbf{67.36 \pm 0.49}$	$\mathbf{78.39 \pm 1.79}$

Table 3.4: The validation F1-score (%) per preprocessing method. For each signal, the highest result is in bold, and the second highest result is underlined

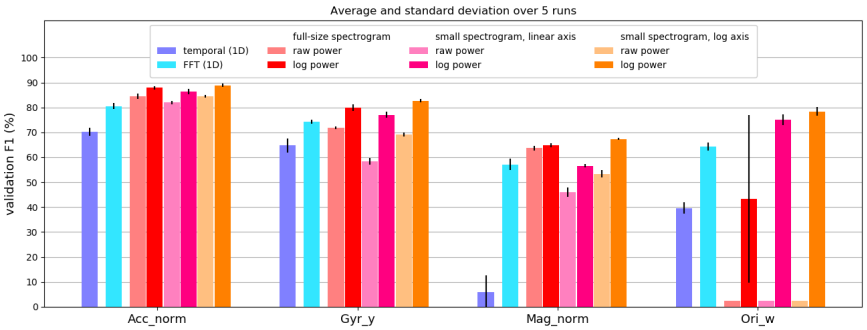


Figure 3.5: A bar plot representation of table 3.4. Best view in color.

Table 3.4 gives the results the evaluation of each preprocessing method. Note that the results in table 3.4 are similar to those obtained by Richoz *et al.* [25] with a different neural network architecture: with three sensors (accelerometer, magnetometer, gyrometer), they obtained a F1-score of 79.4 %. For the record, we reproduced their approach (frequency concatenation of FFT segments), with our setting (architecture presented in chapter 2, Fourier transforms of 60-seconds long segments instead of 5s), and obtained a validation F1-score of $81.44 \pm 1.06\%$. The closeness of the results is not surprising, as using sixty seconds instead of five to make a prediction actually brings little additional information [141].

Switching to the norm of the FFT is strictly better than using raw, temporal representations. This difference might be due to the fact that the power spectrum better separates patterns from noise (see fig. 3.6).

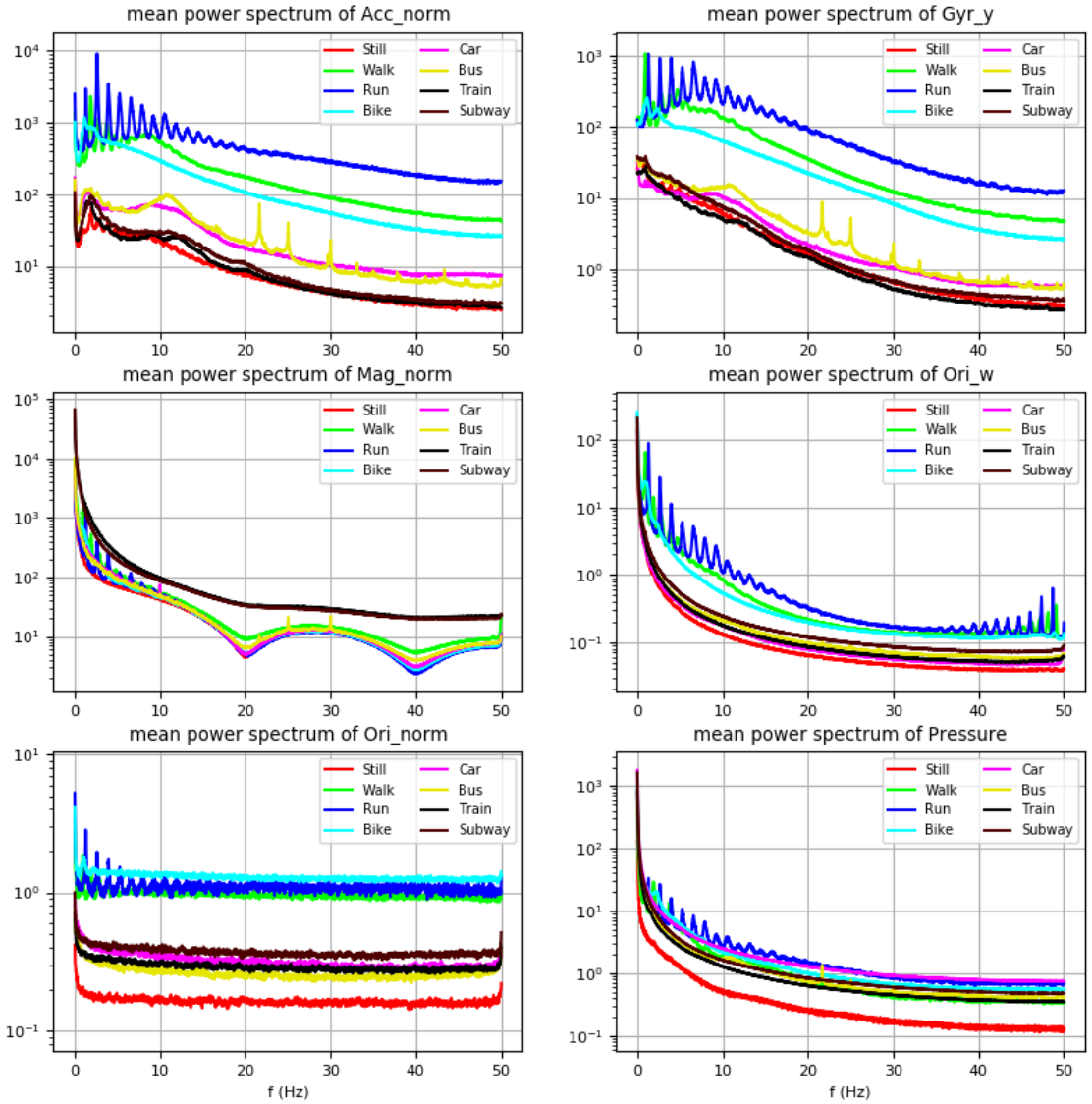


Figure 3.6: The average power spectrum per class (only half of the spectrum is shown). Note the closeness of the fundamentals for the Walk, Run, and Bike classes, the Dirac comb shape of the Run spectra, and the sharp components at 21, 25, and 30 Hz for Bus signals (corresponding respectively to 1260, 1500 and 1800 rpm, the usual rotation speeds of an engine). Best view in color

As expected, using spectrograms seems better, in most cases, than using a power spectrum, or even a temporal segment, and the difference between spectrograms and temporal data (about 25 points on average) is higher than the difference between the two optimized approaches of the challenge (five points between the spectrograms [50] and the best temporal representation [113]). One notable exception, however, is the raw, full-size spectrogram, with the orientation vector. This method has an impressive standard deviation, because two of the five initializations were failure cases that did not learn efficiently and had a F1-score of 2.8% (which is the score of a classifier that predicts the most occurring mode). The others had a F1-score of $76.4 \pm 1.6\%$, which is closer to what one could expect given the results of the other sensors. We will look at this unexpected behaviour in the appendix (chapter D).

Using the log of the power is strictly better than the raw power for the accelerometer, gyrometer, magnetometer: the average gain obtained by switching from the raw power to the log power is 7.66 percentage points. For the case of the Orientation, using the log power is even mandatory. This is due to a particularity

of the signal: contrary to other sensors, the orientation vector is subject to sharp changes: all scalars may be replaced by their opposite between one timestamp and the following one. As these scalars are defined up to a sign, the information stays the same if the whole quaternion is multiplied by -1. But the individual signals are still affected. On a frequency level, this translates into a sharp, but localized, maximum. When computing spectrograms, those maxima prevent the network from seeing anything else. Switching to the log-power allows to dampen these maxima, so that the network works with relevant information.

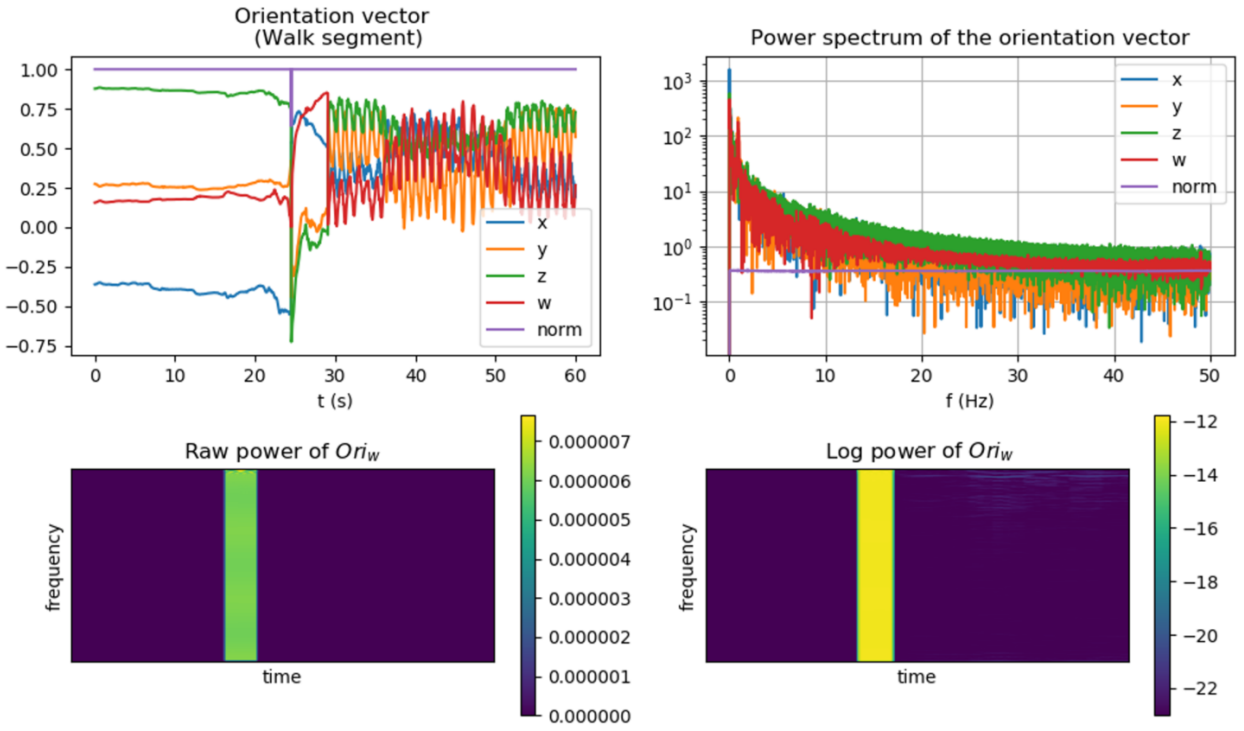


Figure 3.7: An example of discontinuity. Note how the periodic components at the end of the segment leave patterns that remain noticeable with the log-power, and not with the raw power.

Using a logarithmic interpolation for the frequency axis allows to effectively reduce the size of the data without altering the signal as a linear interpolation does. This might be due to the fact that interpolating linearly a (550, 250) spectrogram into a (48, 48) matrix erases the difference between the fundamental frequency of the Walk and Bike segments (the fundamental frequencies for the Bike, Walk, and Run classes are at 0.9, 1.15, $1.3Hz$, respectively). A log scale preserves the distinction between these modes by giving more room to the lower frequencies.

However, we should mention that this section is not entirely fair: to perform our comparison, we used the baseline architecture, a network which comes from a publication using spectrograms [50]. This means that the different hyperparameters (learning rate, number of layers, *etd.*) are likely to be optimized for spectrograms. Using these same hyperparameters for temporal representations (and FFT) is likely to yield suboptimal results, which leads to a bias in favor of spectrograms. A more rigorous approach would perform a hyperparameter search for each representation independently. The fact that we found results similar to Richoz *et al.* despite having different networks might indicate that the architecture has little influence on the performances. However, the difference in performance we found in out experiments (about 20 percentage points on average) is much higher than the difference between the best 1D CNN of the challenge (83% test F1) and the only 2D CNN (88% test F1) [140]. The results of the challenge allow us to be sure that spectrograms are better, but our experiments alone are not enough to reach this conclusion.

3.4 Understanding why spectrograms are more effective

We have seen that the spectrograms seem to be the most efficient representation for the SHL 2018 challenge, but we do not know *why*. We did display a graph (fig. 3.6) which shows some of the classification information can be found in the spectrum of the signal, but this figure, as pretty as it is, is not a guarantee that the network actually relies on this specific information. In this section, we will focus on a network using the spectrogram of the norm of the accelerometer; and try to know which information the network exploits.

3.4.1 Which frequencies are useful for classification ?

We first want to know what are the frequencies exploited by the spectrogram, To do so, we try obfuscating the frequencies (fig. 3.8): we set eight lines of pixels (corresponding to frequencies) to zero in the every spectrogram, and ask the network to predict the class of the noisy samples. We will vary the frequency interval we set to zero to see which frequencies are the most useful: the more important the corresponding frequencies, the worse the performance when this information disappears. We evaluate two type of networks:

- networks that were trained on normal data: we train one network on pristine data, and evaluate it on a degraded validation set.
- networks that were trained on noisy data: for each frequency band we remove, we train the network on a degraded version of the train set, and evaluate it on a degraded validation set. We use the same degradation (we set to zero the same frequency bands) for the training and validation sets.

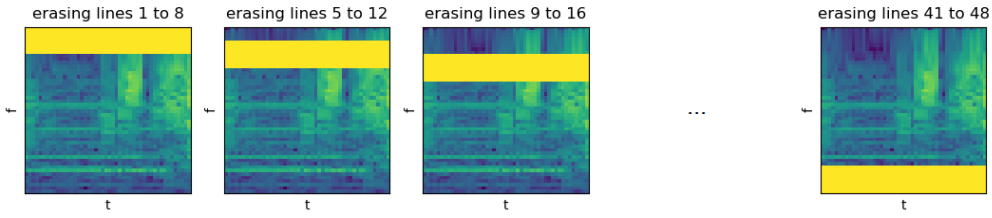
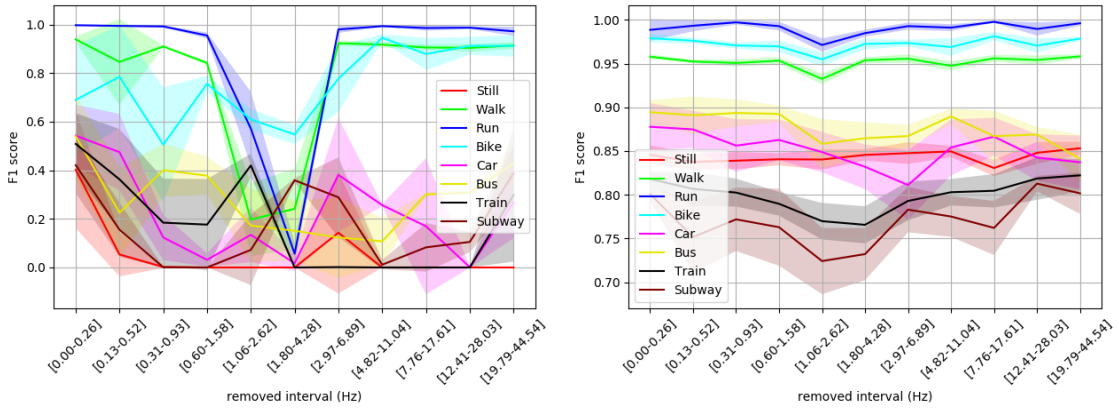


Figure 3.8: An illustration of the data degradation process: we set eight consecutive lines to zero in the 48×48 spectrograms.

If the first experiment matters the most to us (we want to understand the frequency a normal network uses), the evaluation of networks trained on degraded data allow us to see whether there is another way to solve the problem that the one the network chose. Given the strong differences in the frequencies that are proper to each class, we display the F1-score per class. Figure 3.9 displays the results. For the network trained on clean data (fig. 3.9a), the only understandable curves are the Run and the Walk ones: they are almost to their maxima for all but two of the intervals ($[1 - 2.6Hz]$ and $[1.8 - 4.4Hz]$), meaning that the network bases its predictions on these frequencies to predict if the user is walking and running. The other curves are harder to interpret given their irregularity.
For the network trained on degraded data (fig. 3.9b), we cannot draw class-specific conclusions for the opposite reasons: we can see that the network learnt to ignore the noise because the performance is generally much higher than the performance of a network that did not see the degraded data during its training. At best, we could say that there is a slight decrease in performance for the Run and Walk classes for the two bands between 1 and 4 hertz, but the difference (a few percentage points) is much smaller than the variations in the curves for the other classes, which we cannot assign meaning to. The only affirmation we can deduce is that there are enough redundancies in the spectrogram for the network to ignore partially the loss of information: the lowest average F1-score in fig. 3.9b is $86.23 \pm 1.50\%$, which is not far from the network trained and evaluated on clean data ($88.83 \pm 0.71\%$).

We cannot conclude on the data each class needs, but we can draw an additional conclusion about figure 3.9b: given that setting the $[20 - 50Hz]$ to zero during the training process does nor affect the performance much (the average F1-score of the network is $88.61 \pm 0.94\%$), we could have undersampled the signal to

$20 * 2 = 40Hz$ (the $\times 2$ factor comes from the Shannon theorem), without loosing too much information. This means that the sensors that recorded the SHL dataset could have run at 40 Hz instead of 100, saving valuable amounts of energy. A practitioner trying to implement an practical device could have this conclusion in mind to help increase the autonomy of the device.



(a) (b)

Figure 3.9: The F1 per class of a network that was trained on clean (a) or degraded (b) data and evaluated on degraded data, depending on the frequency band removed by the degradation. The width of the curves denote the standard deviation across five random initializations. The log scale for the frequencies make the intervals uneven when expressed in Hertz, while they had the same size (eight pixels) on the spectrogram.

However, when designing this experiment, we did a mistake, which was to reproduce a protocol from Computer Vision (obfuscation, [233]) without thinking why this protocol was not adapted to our signals: when setting a series of frequencies to zero, we create a discontinuity, a border in the frequencies. In other words, the difference between clean data and noisy data is a feature in itself, like the value of any other pixel of the spectrogram. We assume this is why the performance of the 'Still' class is almost always zero when the network did not learn to ignore the noise: it interprets the discontinuity in frequency as a meaningful signal, while it expects to see no (or close to zero) energy when the user is not moving. We assume that in Computer Vision, this is not as much of a concern because occlusion is a natural phenomenon in images. Some Computer Vision training protocols even hide a fraction of the image to make the network more resilient to occlusion [234].

To improve our protocol, we could have considered changing the data degradation protocol: instead of setting the desired segments to zero, we could have replaced the data with a linear interpolation of the closest valid frequencies. However, the main conclusion that we can make is that the network relies heavily on the $1 - 4Hz$ interval to detect the Run and Walk classes. The next chapter will be the occasion to precise how the spectrogram computation make the problem easier for the network to solve.

3.4.2 Computing the average of gradients

Saliency maps

Deep models usually are known to suffer from a lack of immediate interpretability [235]. To be able to know what a network relied on to classify a given image, several works produce a *saliency map*, that is, an image with the same size as the input, which values differ from zero in the regions which contribute to the network's conclusion. Let us consider a spectrogram X which is the input of our network, and choose a class y. To know what are the regions contributing to the classification as y, we generate a saliency map by computing the gradient of the log-probability of the class y: $\nabla_X log(p_y)$. We obtain the saliency map which

has the same shape as X, with both positive and negative values[2]. If one value is positive (resp. negative), it means that increasing the scalar in the corresponding position in the input increases (resp. decreases) the probability p_y.
Remark 1: Contrary to the training process of the network, in which we compute the gradient of loss with relation to the network's weights, here, we compute the gradient of log-probability with relation to the *input*.
Remark 2: in this manuscript, we only apply salience to compute the gradient of the log-probability of the class the spectrogram belongs to (ground truth), but we could have computed the gradient of another class (and in particular, the class the network predicted in case of a mistake).

One could wonder why we do not compute the gradient of the probability directly, and choose to focus on the log-probability. This protocol is common to many saliency maps publications, even the first publication computing saliency maps used the class logits. In our case, we do so because the probabilities are computed with a softmax of the logits: as fig. 3.10 illustrate, their gradient are often too close to zero. When we will add the saliency maps to each other, if we added the gradients of the probabilities, many spectrograms for which the model is certain ($p_y = 0$ or $p_y = 1$) would simply not account in the average.

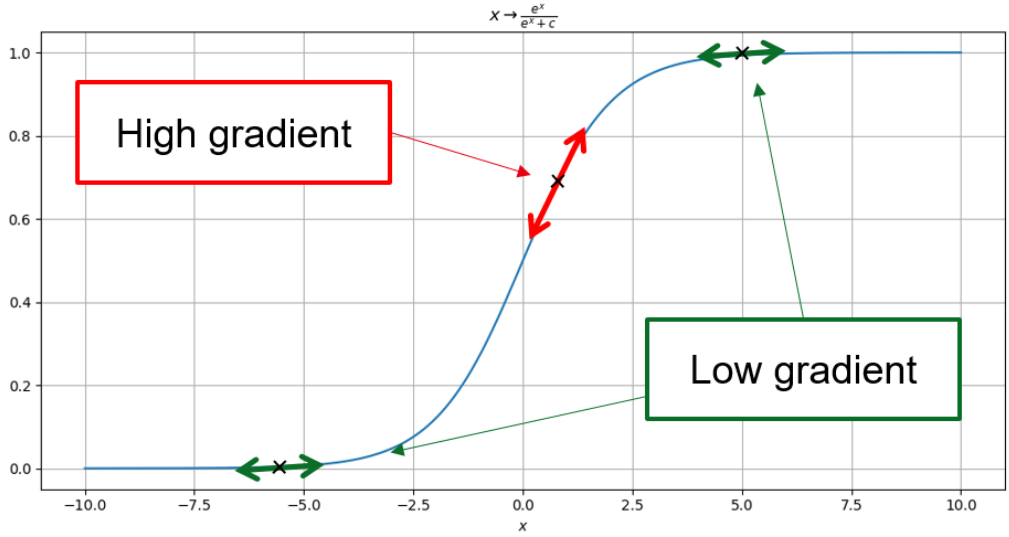


Figure 3.10: the reason why we compute the gradient of a log-probability: for many samples, the gradient of the probability are too low to account for in an average.

In the literature, there is a large diversity in the family of methods to create a meaningful saliency map. We presented the most baseline saliency map, but several alterations exist, such as removing the negative gradients [144]. We will not go through a complete inventory of the different saliency maps in the literature, because a recent paper [236], demonstrated both theoretically and experimentally that methods like DeconvNet or Guided BackPropagation (GBP) partially reconstructed the input image instead of highlighting the elements the network based itself upon. As the most basic saliency map did not suffer from this flaw, we will consider only this technique. Even though its usefulness has been questioned [237, 238, 239, 240], we will demonstrate the relevance of this method for our application.

[2]Some [144] consider the absolute value of the saliency map to differentiate between the informative regions (high absolute value) from the low-informative ones (values close to zero), a treatment we will not use.

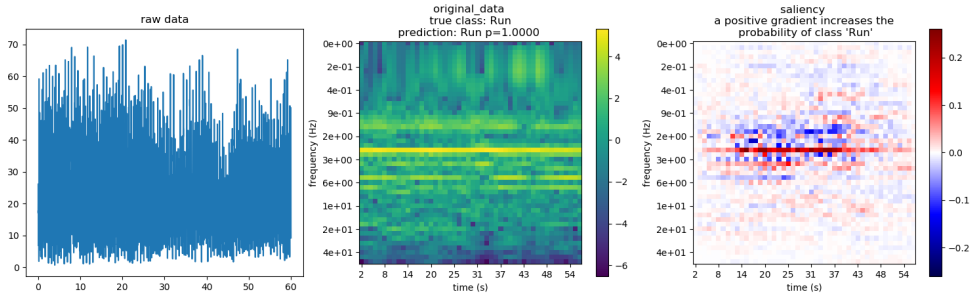


Figure 3.11: An example of saliency map with a single Run segment.

Figure 3.11 displays an example of such saliency maps. When looking at this example, it seems that the saliency map correctly translates the useful frequency band we saw being used in the previous section. However, we would like to use a method for which we do not have to look at samples one by one. We would like to compute an average of saliency maps per class, but we are not yet sure that such an average makes sense. Figure 3.12 illustrates the main problem we face: computing an average only makes sense if every input pixel has the same meaning. Given that the position of a point on the time axis does not change the meaning of the information much, a pattern might be present anywhere along this axis. By summing the 48×48 values of the spectrogram, we run the risk to sum different instant of the same pattern, hereby destroying it. This problem is not a concern for the frequency axis, because two patterns which have different positions on the frequency axis have different meanings.

To summarize, we need to make sure that no interesting patterns on the time axis are destroyed. In the next section, we will demonstrate that the temporal variations we observe in the gradients are not used much by the network.

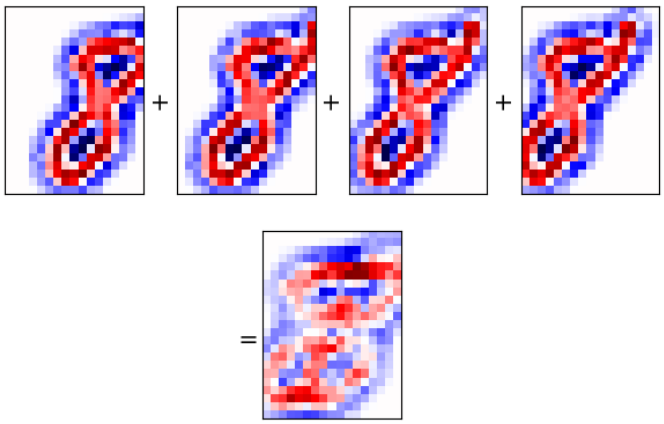


Figure 3.12: An illustration with artificial data of what we want be careful to when averaging the gradients: summing different versions of the same motif, at different time steps, might destroy it.

Hypothesis verification: the network uses mainly the frequencies

To verify whether the time axis is relevant, we will use a data degradation that leaves one axis intact: the *shuffle* (see fig. 3.13 for an illustration). We select all the lines (resp. columns) of the spectrogram, and shuffle their order at random (with uniform probability, independently for each spectrogram). By doing so, we completely destroy any pattern that appeared on this axis. Then, we ask a network trained on clean data to predict the class of degraded versions of the validation samples. Given that the shuffle operation made the axis useless, if the network still obtains a good performance, it means that the respective positions of the lines (resp. columns) do not play an important role in the network's prediction. Inversely, the lower the performance, the more important the position on the axis we just shuffled. We also consider shuffling both

axes to provide an additional verification.

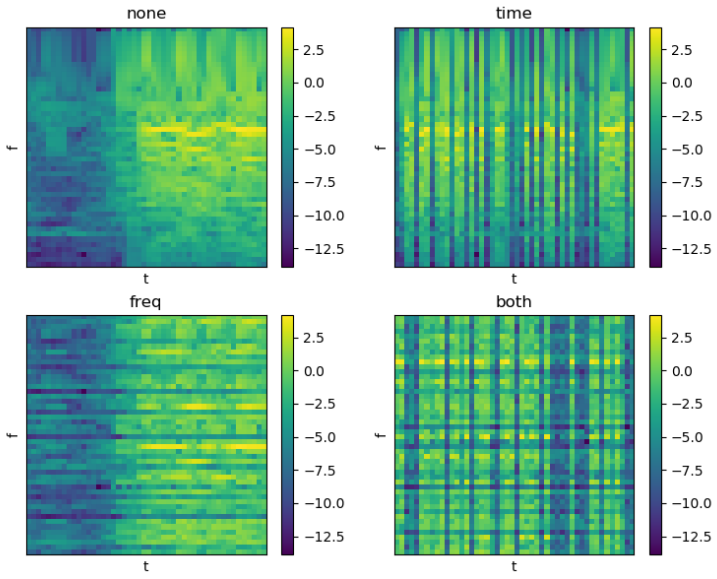


Figure 3.13: An illustration of the axis shuffling with a spectrogram from a Run segment. On top of each spectrogram is the name of the shuffled axis

	clean time	shuffled time
clean frequencies	$89.0 \pm 1.6\%$	$80.7 \pm 0.7\%$
shuffled frequencies	$21.7 \pm 2,7\%$	$24.4 \pm 3.8\%$

Table 3.5: The validation F1-score of a network trained on clean data and evaluated on spectrograms whose axis were shuffled (see fig. 3.13). We repeated the evaluation with five random initializations of the network.

Table 3.5 gives the result. We can see that the decrease in performance is much higher when shuffling the frequency axis (there is a difference of about sixty points between each result of the top line and their bottom counterpart) than the difference when shuffling the time axis (left versus right). The fact that there is (relatively) little difference between the spectrograms and their time-shuffled counterparts means that the time axis : the interesting patterns seem to be located in majority along the frequency axis. Figure 3.12 does not happen in our case: the network saw the input signals wee stationary and does not seem to search for temporal variations in the signals' spectra. In other words, the fact that the temporal organization of the spectrogram values are much less important than the organization on the frequency axis mean that we can safely sum the gradients of spectrograms.

Understanding the sum of gradients

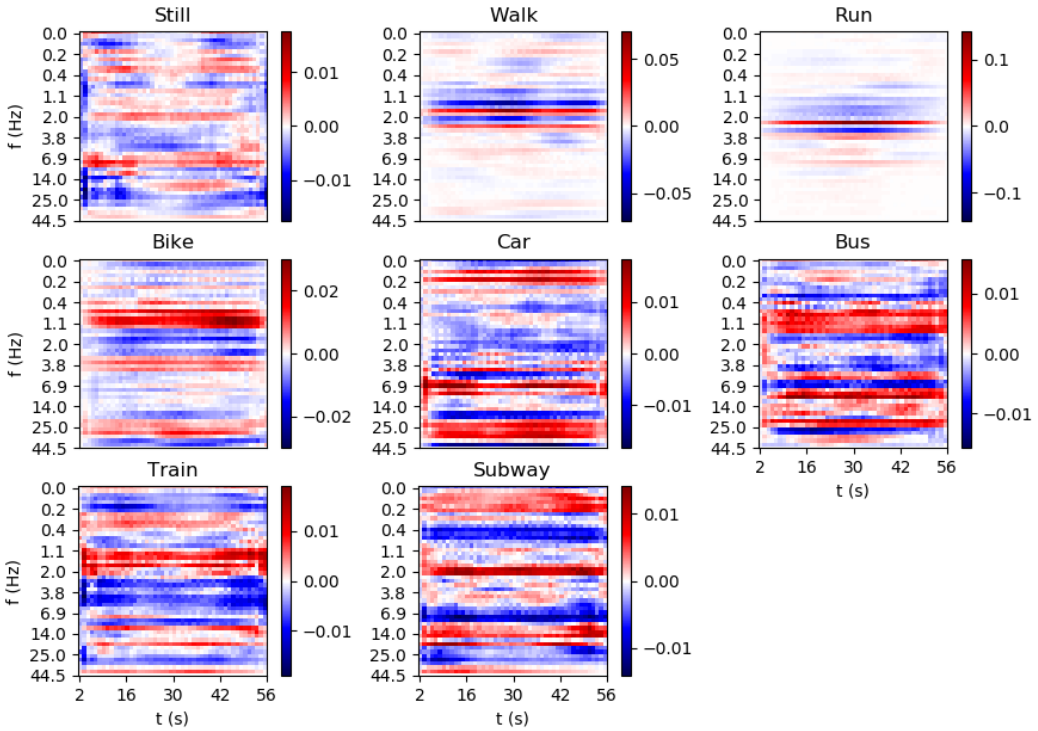


Figure 3.14: The gradient of each class.

Now that we know we will destroy little information by summing gradients, we can proceed to compute the average of gradients: for each of the eight classes, we gather the *training* samples belonging to the class, compute the saliency map using the gradient of the log-probabilities, and average the gradient for each class. The result is in figure 3.14 (a zoom on the most interesting frequencies is provided in fig. 3.15), and displays several interesting patterns. Firstly, to predict the Walk class, the network is mostly influenced by the $[1.8 - 2.0Hz]$ and the $[3.0 - 3.4Hz]$ bands. On the other hand, a high power in the $[1.4 - 1.6Hz]$ and $[2.3 - 2.6Hz]$ bands will strongly reduce the probability of the network classifying the sample as walking.We assume that the negative bands come from the interactions between classes: if the sample exhibits a sharp component at $2.6Hz$ (We remind the reader that the fundamental frequencies for the Bike, Walk, and Run classes are at 0.9, 1.15, and $1.3Hz$, respectively, but the highest power is at 1.8, 1.15, and $2.6Hz$ for these three modes). This is further shown by the fact that the Run gradient is at its highest for the $[2.6 - 3.0Hz]$ band, one line above the lowest value for the Walk gradient. The same goes for the $[1.1 - 2.6Hz]$ band, which contributes to the Bike class.

Surprisingly, if the Bus class displayed strong components at 21, 25, and $30Hz$ (fig. 3.6), the network seems to only use the $25Hz$ band: the intervals $[19.8 - 22.2Hz]$ and $[28.0 - 35.3Hz]$ seem to contribute negatively to the Bus class. Most importantly, these bands are not the ones to have the highest absolute value in the whole gradient: their contribution does not seem to outshine the other frequencies' influences.

Also, please note how strong the gradients for the Walk and Run classes are (the maximal values of these two classes' gradients are five to ten times higher the value of the other classes'). We could assume that any model looking at the frequencies can at least classify these two classes. We will see that it is the case, at least for the Run class. But before proceeding, we need to make sure that the gradients we just computed are relevant.

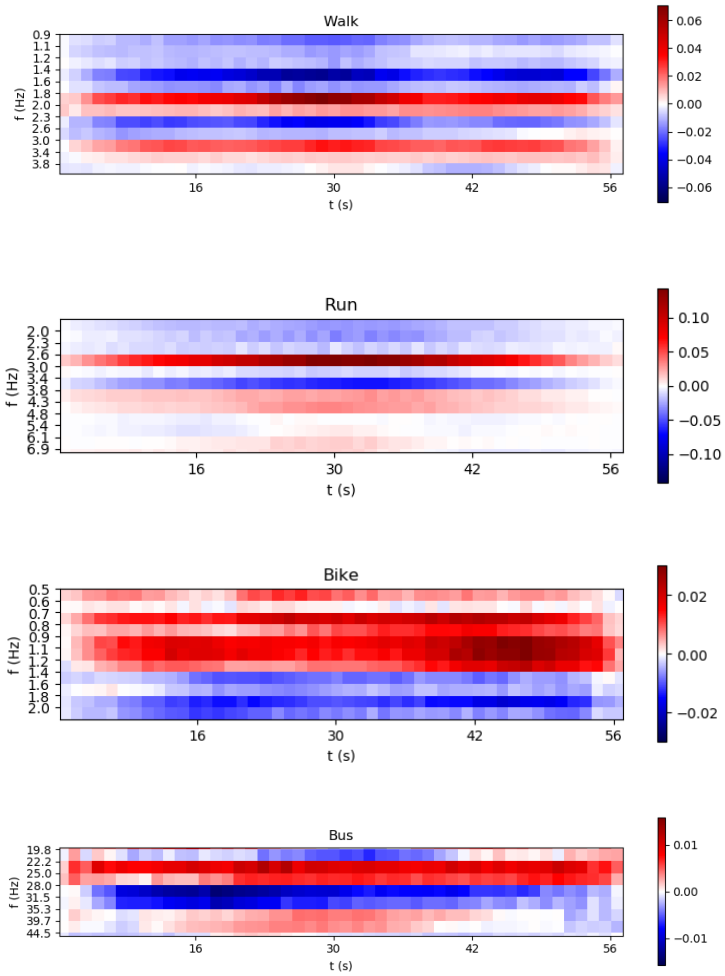


Figure 3.15: A focus on some gradients from figure 3.14.

A sanity check: Going further towards the gradients

For many classes (Still, Car, Train, Subway), the gradients are fairly hard to interpret. To make sure that all our gradients make sense, we will try to evaluate the predictions of the network when we add the gradients of each class to all the samples in the validation dataset. If the gradients are relevant and actually encompass the important frequencies, we will observe a shift in the predictions, such that the networks predict the class the gradient comes from. To be able to compare the speeds of the prediction shifts across classes, we do not consider the gradient themselves $(g_c)_{c\in\{Walk,Bus,etc.\}}$, but the normalized gradients $(\frac{g_c}{||g_c||_2})_c$. To save time, we also add the gradients by batches of 20 (we whose this value to be a compromise between the resolution of the figure and the execution speed of the code). The following pseudo-code algorithm shows the outline of our experiment:

Require: A list of gradients $(g_c)_{c\in\{Walk,Bus,etc.\}}$, a step size (chosen to be 20)
  for every class c **do**
    $predictions\_history_c \leftarrow \emptyset$
    while 90 % of the predictions do not belong in the same class **do**
      for every sample in the dataset **do**
        $sample \leftarrow sample + step\_size * g_c/||g_c||_2$
      end for
      Record the predictions $L_{predictions}$ on this new dataset

$predictions\_history_c \leftarrow predictions\_history_c \bigcup \{L_{predictions}\}$
end while
end for
return the lists $(predictions\_history_c)_c$ which show the prediction shift

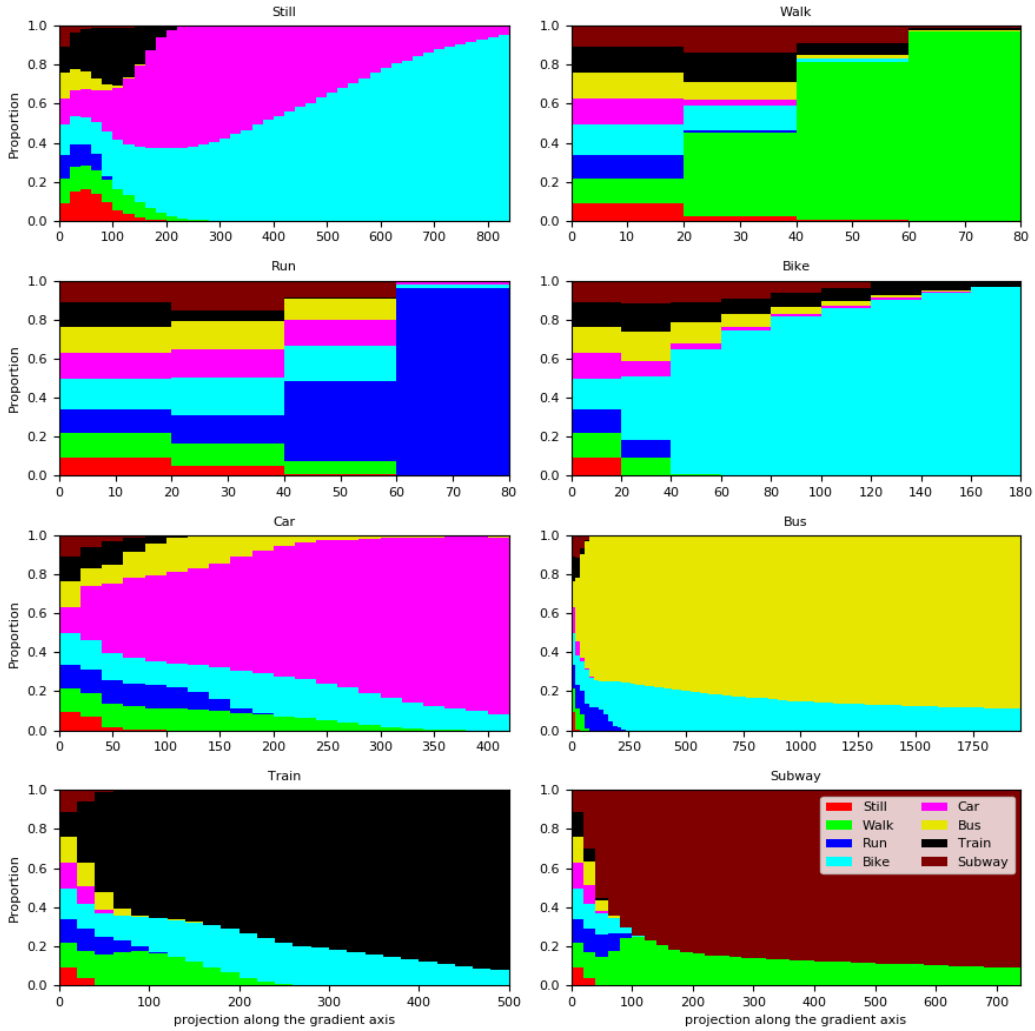


Figure 3.16: The histogram of the predictions on the validation set after we added the gradient for a given class a certain amount of times. To display this graph, we re-weighted the samples so that the bars appear balanced when the dataset is untouched.

Figure 3.16 gives the result: for many classes, adding repeatedly the gradient to the whole set does push the predictions towards the class the gradient was computed with. The only exception is the Still class, for which adding the gradient more than forty times actually decreases the probability for a sample to be classified as Still. We assume this is due to the fact that once we add the gradient a sufficient amount of times, the power of some of the frequency bands becomes overwhelmingly positive, which the networks interprets as seeing a high-energy segment: in short, the network sees the phone move.

On the other hand, the Walk and Run gradients' addition push the predictions towards the respective class extremely fast, even though we made sure all gradients had the same norm.

The fact that the gradients are enough to push the direction for seven of the eight classes indicate that they carry some meaning about the network's predictions. In the following section, we will demonstrate that the network behaves linearly to identify the Run class.

Measuring the linearity of the network

Given how fast adding the gradients for the Run and Walk classes caused a prediction shift, and given that these two modes are easy to distinguish using the signal's frequencies, we might wonder if a neural network really learns complex features learn to classify these easy classes. In particular, we will show that the network behaves like a linear classifier for the Run class. To show it, we will use three experiments, which fig. 3.17 illustrates:

- **Experiment 1:** We create a linear classifier which class prototypes (the vectors used to compute the class logits) are the classes' gradients. This is intended to show *if* the classification problem admits a linear solution.
- **Experiment 2:** We project the validation dataset *onto* the subspace spanned by the eight gradients (that is, we force the samples to be linear combinations of these eight gradients), and ask the (nonlinear) neural network to classify this altered set. If the network is linear, the predictions will not change after the projection.
- **Experiment 3:** We project the validation dataset *along* the subspace spanned by the eight gradients (that is, we 'remove' the gradients from the samples), and ask the network to classify this altered set. If the network is linear, the predictions will change drastically after the projection.

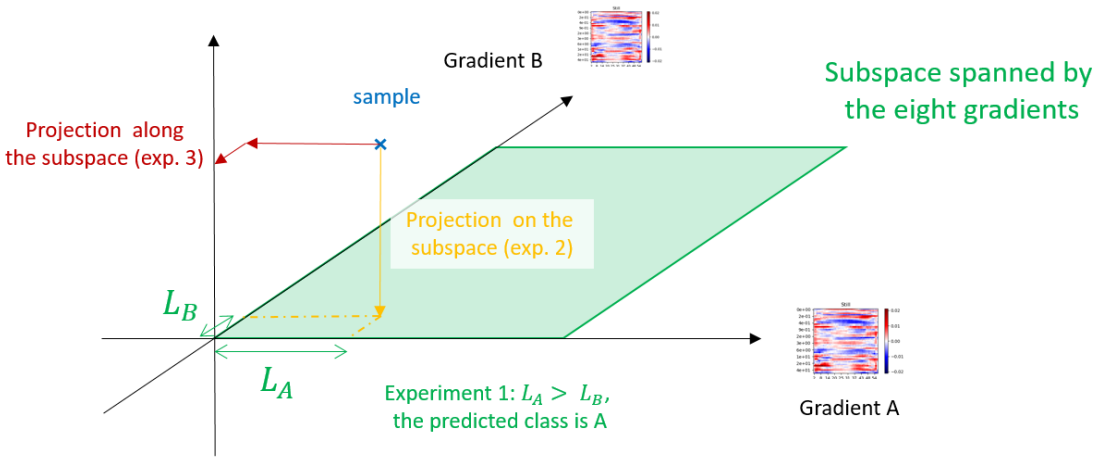
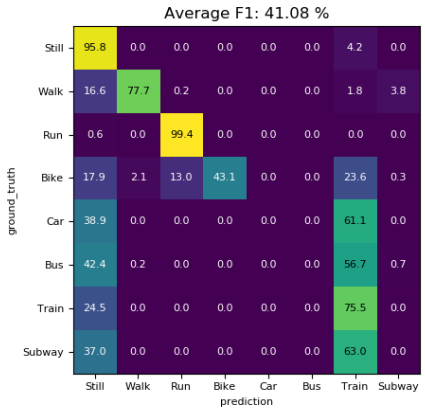
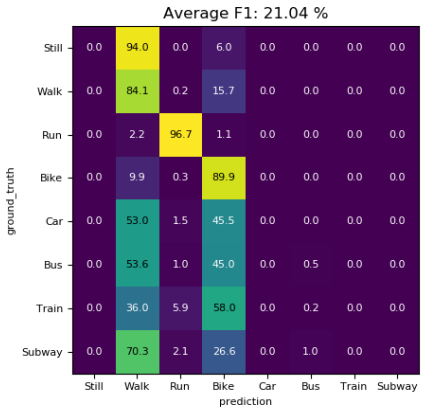


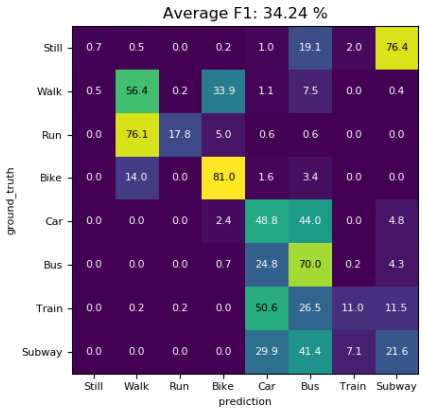
Figure 3.17: the three experiments we will lead to show the network classifies some classes linearly.



(a) linear classifier



(b) projection onto the subspace



(c) projection along the subspace

Figure 3.18: The results of the three experiments described in fig. 3.17. The first experiment (a) shows the Running segments can be classified linearly with little error; while the two others demonstrate that the network actually behaves linearly.

Figure 3.18 displays the results of the three experiments. Each time, we look at the resulting confusion matrix. For experiment 1, we should look at both lines and columns to assess the performance of a linear classifier. But when looking at the fate of the predictions (experiments 2 and 3), we should look at the lines of the matrix (the samples actually belonging in the class), to know whether the alterations we introduced disturbed the network. The results are the following:

- Experiment 1 (fig. 3.18a) shows that a linear classifier discriminates the Run class extremely well, apart from a slight confusion with the Bike segments. This is an indication that the problem is linearly separable using the network's gradients.
- Experiment 2 (fig. 3.18b) shows that if we were to project the samples on the gradients, the prediction of the network on the Run segments would be unchanged, a strong indication that the problem of classifying the Run segments versus the rest is solved by the network using a linear decision boundary.
- Experiment 3 (fig. 3.18c) shows that the Run segments are misclassified as Walk when we remove the Run class' gradient. On the other hand, the Walk segments are well classified even after removing the gradient, an indication that the network is does not behave linearly to discriminate this class.

Experiments 2 and 3 demonstrate that the network using the norm of the accelerometer behaves like a

linear classifier for the Run class. This shows one of the interest of the spectrogram representation: compared to a high-dimensional temporal representation, spectrograms helps make the problem easier to solve.

To sum up, this section just showed that the spectrograms seem to obey the behaviour described in figure 3.3: they make the problem more simple, thereby improving the performance when the number of samples is low.

In particular, we said that the spectrograms made the classification problem linear for the Run class, but this is not exactly what we wanted: we wanted to know what did the spectrograms change *with respect to* the temporal representations. If the problem of classifying Running spectrograms was almost linear, but the problem of classifying raw, temporal representations of Running segments was *exactly* linear, then the spectrogram would not have made the problem easier. Sadly, we cannot directly compute an average of gradients in the temporal domain, because the phase will prevent the most important frequencies from aligning with each other.

There are other ways we could have pursued our work: we could choose to do an architecture search for each representation. Or, we could have decided to study the hypothesis that came from the study of the literature, which is that spectrograms are better suited for smaller datasets than temporal representations (for instance, we could have reduced the number of samples to see if the performance difference between representations increases).

3.5 Conclusion

After a small introduction aiming to improve the padding used in several works of the literature, we focused on the difference between raw representations and spectrograms. We concluded our overview of the literature by saying that that spectrograms seem to be most useful when the number of samples is low, before implementing a comparison ourselves with the SHL dataset. We tried understanding what did the spectrograms bring, and we saw spectrograms made the classification problem more simple for the network. If the conclusions of the literature study apply to any temporal signal, the fact that a network behaves linearly for one class seems proper to TMD: in the case where the signals are not stationary, we would expect the decision boundary to be much more complex than a mere linear function.

Figure 3.3 seems to be verified by both the bibliography and the experiments we led. Our original question was: "should we use spectrograms ?". Our bibliographic study and experiments indicate that, unless we work with a million-samples dataset, the answer is 'yes'. In other words, without a huge database, we need to help the networks by computing a representation that simplifies the problem.

Chapter 4

Global Pooling

In its most basic form, a Convolutional Neural Network is usually made of two types of layers: the convolutional layers, which process 1-dimensional signals, 2D images, or even 3D tensors [241]; and the Fully-Connected layers, using fixed-size vectors. We use the term 'fixed size' because a convolution can use inputs or outputs of any size, provided their number of dimensions match the type of convolution. However, the one or two-dimensional signals all have an extra 'channel' axis that does have a fixed size, which means that the inputs and outputs of a 1D convolutional layer are actually a two-dimensional tensor. The 'channel' axis is different from the others because it encodes the type of information present at a given location (while the other axes encode the location on the 1D or 2D tensor). This is why we will think of the vectors of the fully connected layers are zero-dimensional representations: they encode information about the whole segment.

To obtain this global representation from local (1D or 2D) features, one needs to use an operation that effectively removes one (or two) of the axes. This section is devoted to the study of the pooling operations, the replacements for the flatten step. In this short chapter, section 4.1 will present the types of pooling available, section 4.2 will develop the choice of metrics we focused on to assess the efficiency of the network a given pooling method produces, while section 4.3 presents the results and compares our network to the state of the art. We will see that we obtain a particularly efficient network: we reach performance levels that compare to the state of the art with only 11,000 parameters. If the alternatives we present here are well known in the Computer Vision community [142], our contribution is to bring them to the domain of Transport More Detection.

4.1 The different types of global pooling

Historically, the flatten layer appearing with Convolutional Neural Networks is the *flatten operation* [1], which simply aligns all the values of a tensor into a vector. But in 2015, the Resnet architecture introduced the use of an average along the two dimensions of the image [150]. In our case, if $X_{t,c}$ is the two-dimensional matrix at the end of the last convolution layer (t being the index along the 'time' dimension, while c is the index along the 'channel' dimension), the average of the features would be equal to $Y_c = \frac{1}{T}\sum_{t=0}^{T-1} X_{t,c}$. Alternatively, we could use a maximum ($Y_c = max_{t\in 0..T-1}\,(X_{t,c})$), but a more general option would be to use the generalized mean [242]:

$Y_c = (\frac{1}{T}\sum_{t=0}^{T-1}(X_{t,c})^{\alpha_c})^{1/\alpha_c}$.

This expression uses one parameter $\alpha_c > 0$ for each channel c, which are learnt by gradient descent like any of the other weights. When $\alpha_c = 1$, the expression is equal to the arithmetic average. When $\alpha_c \to +\infty$, the generalized mean converges towards the maximum of the $(X_{t,c})_{t\in[1..T]}$. In order to avoid numerical instability, a small term[1] was added to the input tensor X. In practice, the values of α_c are initialized following $\mathcal{N}(5,1)$, a normal distribution with an average of 5 and a unit standard deviation. The choice of a distribution is arbitrary, the only constraint being that we do not want α_c to be negative before the learning even begins (if $\alpha_c < 0$, the expression is equivalent to using a generalization of the harmonic mean, which is

[1] the lowest value we could use was 5.10^{-5}, which seems quite high

close to 0 when one of the features $X_{c,t}$ is close to zero, which makes the associated channel useless). The value of all the α_c seems to converge between 0 and 5 during the training process.

Now, these pooling methods are said to be *global*, because they use an entire feature map and return a single, fixed-size vector. However, *local* variants of these pooling methods exist: a local pooling uses a feature map as an input, and returns another feature map, where every pixel is the maximum (or the mean, etc.) of a certain number of input pixels at the corresponding position in the input feature map. As an example, many neural networks do include several local maxpooling layers to reduce the dimensionality of the intermediate feature maps. We do not include these methods in our experiments, but we assume their efficiency (in terms of either classification performance or computational requirements) in between the global pooling and the flatten step (which we could understand as performing absolutely no pooling). However, a local pooling layer would lack a key perk that global pooling methods have: the ability they provide to process inputs of arbitrary sizes.

As Wang *et al.* [142] mention, the choice of a global pooling allows to use segments of any length as inputs of a network. As we said, the convolutional layers can use inputs of any size (provided the number of channels match), while the number of features in the FC layers is fixed. However, contrary to the flatten step, the global pooling operations allow obtaining the same number of features no matter how many time steps T the input segment had. This means that no matter how long is the input segment, the fully connected layers will receive a segment with a fixed size (see fig. 4.1). This is why the baseline architecture for the GeoLife database, which we presented in chapter 2 is able to process segments of different sizes.

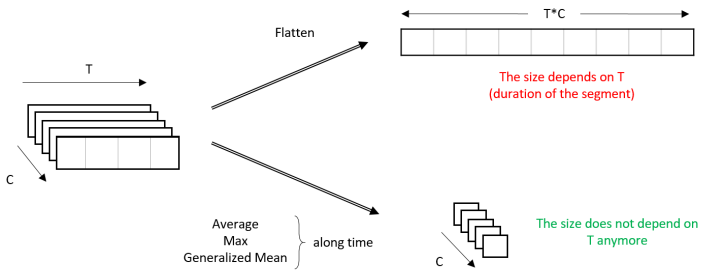


Figure 4.1: Why a global pooling method allows a network to process inputs with different shapes. This idea originally comes from Computer Vision ([142]) , but was unknown in TMD.

4.2 Evaluation metrics

To evaluate the diverse measures we will employ, we will look at three types of values: classification performance (validation F1), but also at the number of weights and operations of a network.

One might wonder why we focus on these measures and not on running times, for instance. To understand the reasoning that led us to choose these values, we will take an example with a practical use case. Let us consider the example of the application which made us focus on Transport Mode Detection in the first place: an automatic carbon footprint estimator on a smartphone. When dealing with embedded devices, most neural networks are trained offline (on a computer) on prerecorded data, and sent to embedded devices for inference. This method relies on the fact that the training is the most computationally intensive step. In a real use-case scenario, the end-user would download an application containing the trained model, and record the beginning and end of their trips. The application would automatically estimate the transport mode of the user to compute an estimation of the greenhouse gas the trip emitted.

Compared to cloud-based applications which off-load computational steps to remote servers, this embedded classifier design keeps the user's privacy while still operating regardless of network coverage (the GPS coverage is different from the phone network or broadband such as 3G, 4G, *etc.*). However, if this embedded classifier is not resource-efficient, the application will end up draining the battery of the end-users, who might choose not to use it. There are publications that manage to embed neural networks efficiently, (see [243], for instance), however, a heavy network with many parameters is always a hurdle to efficient embedding. Many real-time applications also consider another parameter: the inference time. For instance, a network

detecting objects in videos should be able to process more than one image per second to make sure to avoid misdetection. Similarly, a network that works in a client-server architecture (as in [109]) needs to be quite fast because it will process the trajectories from all users at once. However, in our case, the transport mode of a user does not change too often (the average duration of a tripleg is 28 and 24 minutes in the GeoLife and SHL 2018 datasets, respectively), and a model is to process the trips of a single user. This is why an application does not need to run more than once per minute to get accurate results.

This is why our main concern is the number of weights of a network (which influences the memory the network will require) and the number of operations (which condition the battery consumption of the network), in addition to the performance. We do include training and evaluation times as an illustration, even though these results depend heavily on the device. For instance, GPUs are more optimized for parallel processing, such as convolutions. The number of parameters and operations, however, is independent of the implementation, and provides an objective measure for comparison.

Hardware and practical setup

We trained the models on a server with an Nvidia tesla V100 GPU (32 Gb of memory), Cuda version was 10.2, and a 40-core Intel Xeon Gold 6230 CPU @ 2.10GHz with 190 Gb of RAM. The evaluation times were measured on a CPU, a 4-core Intel i7-7820 @ 2.90 GHz with 32 Gb of RAM. However, those running times are not absolute measures: some devices are better optimized for different types of operations. The only objective measurements are the number of parameters and operations, which do not depend on the device. For each result (F1-score, training times), we repeat the training and evaluation process 5 times, changing the seed each time. We display the average and standard deviation for each result. We computed the number of operations using the code from [244]. When the input shape may vary, we used the median length of a segment in the dataset, which is 500 points for the interpolated dataset (in our case) and 200 points for the original GeoLife dataset ([108, 109]).

4.3 Results

4.3.1 Comparison of the alternatives to the flatten step

We compared three alternatives to the Flatten, namely Maximum, Mean, and Generalized Mean. As table 4.1 shows, on the GeoLife dataset, only the flatten step is worse than the rest, in terms of performance, computational requirements, and training and testing time. We assume that the worse performance of the flatten step is due to the fact that this pooling has to process shorter segments. When a segment is long, the networks that use a flatten step see a smaller fraction of the segments than networks using a global pooling, which means the classification is less likely to be precise.

As for the running times, table 4.1 shows us that the use of alternatives to the flatten step make the complete training process *longer*, as the convergence is slower with these architectures. Given that most applications rely on training the model offline (on a computer), the training time of a model is not a major concern. More interesting is the inference time: we can see that the operations do not change the duration, notwithstanding the global pooling operations requiring much fewer operations for a single inference. We hypothesize this equality in running times is due to the fact that the flatten operation relies on a matrix multiplication, an operation that can be parallelized with any modern library. Alternatively, our implementation of these operations may be insufficiently optimized. Either way, the fact that the global pooling operations are as fast as the flatten is not much of a concern to us because Transport Mode Detection does not require real-time inferences.

The number of weights of the network (which determines the memory size required to fit the network in any device) and the number of operations (which we use as a proxy for the energy consumption of the model) are both smaller for the global pooling operations, which is why we will favour these operations for our application.

Pooling	Validation F1-score	number of parameters	operations (FLOPs)	training time (min)	epochs to convergence	inference time (ms)
Flatten (segments of 1,024 points)	$77.0 \pm 1.6\%$	7.6×10^4	9.4×10^4	7.8 ± 0.7	113 ± 17	1.92 ± 0.13
Generalized Mean	$80.9 \pm 1.0\%$	1.1×10^4	3.3×10^4	43.9 ± 7.3	538 ± 114	1.94 ± 0.04
Average	$80.2 \pm 1.3\%$	1.1×10^4	3.3×10^4	78.7 ± 34.6	1161 ± 571	1.81 ± 0.04
Maximum	$80.3 \pm 1.6\%$	1.1×10^4	3.3×10^4	16.8 ± 2.7	262 ± 66	1.85 ± 0.06

Table 4.1: The effectiveness of each kind of pooling, in terms of performance, computational resources, and training and inference time. For each result, we display the average and the standard deviation, over 5 runs

4.3.2 Comparison with the state of the art

When looking at table 4.1, we see that the choice of a pooling operation has a significant impact on the computational requirements of the network (number of parameters and operations), which was why ResNet introduced a global average for initially [150]. We even obtain a network with 11,000 parameters and 33,000 floating-point operations. To the novice, this might seem a lot, and it still represents much more than any other Machine Learning algorithm (save maybe k-nearest neighbours). But it is actually a minuscule number for a deep network: in image processing, models usually have several millions of parameters, and even a model like SqueezeNet, which has been developed to reduce as much as possible the memory footprint, has 400,000 parameters [245]. Here, the model for the GeoLife dataset has 40 times fewer parameters. Even compared to the other works in the TMD literature (table 4.2), our network is four to 100,000 times smaller than the other networks, while still retaining similar performance levels. This is partly because most of the other networks are either classification CNN using the more expensive flatten step ([28, 57, 55]), or LSTMs, which rely on a costly matrix series of multiplications ([108, 109]).

model	reported score	Number of weights	Number of operations (FLOPs)	classes	remarks
LSTM + embedding [108]	94.5 % AUC	1.1×10^6 $(100 \times p)$	4.2×10^8 $(10,000 \times o)$	4	No mention of the splitting No test set
Our GeoLife Baseline	$97.1 \pm 0.3\%$ AUC	1.1×10^4 (p)	3.3×10^4 (o)	4	/
Convolutional LSTM [78]	80.67 % F1	?	?	4	No additional data No test set
Convolutional LSTM [78]	83.97 % F1	?	?	4	AD: weather No test set
Our GeoLife Baseline	$87.1 \pm 1.1\%$ F1	1.1×10^4 (p)	3.3×10^4 (o)	4	/
Convolutional Auto Encoder [57]	76.4 % F1	4.1×10^4 $(4 \times p)$	6.4×10^6 $(100 \times o)$	5	The trajectories are not segmented AD: Unlabeled GeoLife data No test set
Convolutional Auto Encoder with skip-connections [65]	80.4* % F1 67.7 % $\overline{IoU}$	?	?	5	The trajectories are not segmented AD: Unlabeled GeoLife data No mention of the splitting No test set
Fully-connected Autoencoder [81]	93.44 % F1	?	?	5	The trajectories are not segmented AD: Bus stop positions Incorrect splitting No test set
Unsupervised Convolutional Autoencoder [55]	80.5 % Acc.	3.9×10^5 $(40 \times p)$	7.2×10^6 $(100 \times o)$	5	They did not use any labels to compute the clusters No mention of a val. set
CNN ensemble (7 models) [28]	84.0 % F1	$7 \times 2.6 \times 10^6$ $(1,000 \times p)$	$7 \times 1.7 \times 10^7$ $(1,000 \times o)$	5	No test set
semi-supervised LSTM ensemble [127] (4 models)	$91.5 \pm 0.41\%$ Acc.	$4 \times 3.2 \times 10^5$ $(100 \times p)$	$4 \times 5.2 \times 10^8$ $(10,000 \times o)$	5	No test set
LSTM + Wavelet features [109]	91.9* % F1 92.7 % Acc.	8.1×10^6 $(1,000 \times p)$	7.3×10^9 $(100,000 \times o)$	5	No mention of the splitting No test set
Our GeoLife Baseline	$83.9 \pm 1.1\%$ F1	1.1×10^4 (p)	3.3×10^4 (o)	5	/
Random Forests [27]	71 % F1	50 trees	?	6	No test set
Our GeoLife Baseline	$81.8 \pm 0.9\%$ F1	1.1×10^4 (p)	3.3×10^4 (o)	6	/
AE + Logistic Regression [38]	67.9 % Acc.	2.7×10^5 $(10 \times p)$	5.2×10^5 $(10 \times o)$	7	No test set
Our GeoLife Baseline	$74.1 \pm 0.7\%$ F1	1.1×10^4 (p)	3.3×10^4 (o)	7	/

Table 4.2: The comparison of the performance, number of weights, and number of operations required for a single inference (forward pass) of the networks in the literature. This is a reproduction of table 2.4, which we presented in chapter 2, except that we added the number of weights and parameters. The question marks (?) denotes the publication which did not leave enough details to obtain a precise estimation of the number of weights and operations. AD denotes the presence of additional data.

4.4 Conclusion

This short chapter presented the global pooling operations, the replacements to the flatten step. We looked at the pooling methods in the Computer Vision literature and transferred it to a TMD problem. Not only can we use them to increase the performance of our model, but we can also reduce its size to an extreme degree, reaching as low as 11,000 parameters. For future work, it might be interesting to try to use architecture compression to obtain an even smaller network.

Chapter 5

Data fusion

When we presented the publication that serves as our SHL baseline [50], we mentioned that the way they merged data from two different sensors is surprising: they mention simply putting two spectrograms side by side to form a single image. Merging the information from different signals, a problem called *Data Fusion*, is the problem we will try to tackle in this chapter. If most of the works in Multimodal Deep Learning focus on videos (with both RGB images and sound) or images with text, this problem touches any Machine Learning dealing with multiple sensors. In particular, the inertial sensors of the SHL dataset we selected (accelerometer, magnetometer, gyrometer and orientation vector) for Transport Mode Detection. This chapter is devoted to the evaluation of diverse Data Fusion architectures on the SHL dataset. Our main contribution is to establish a benchmark that proves that, on Transport Mode Detection, no data fusion method is significantly better than the others.

In the first section, we provide a brief overview of the fusion methods used in multimodal deep learning. Then, section 5.2 presents a list of data fusion architectures we selected for evaluation. Section 5.3 displays the results of the comparison, and mainly conclude that no fusion method strictly outperforms the others. In section 5.4, we experiment with a novel fusion method that tries the sensor-specific layers to generate features that are complementary between sensors. If the fusion method in itself does not increase the performance, we use it to show that the networks are able to learn the right amount of redundancy by themselves. Finally, section 5.5 concludes by an evaluation of the best fusion method we found on the SHL *test* set, which we did not use until now.

5.1 Data Fusion modes in deep learning

As we said earlier, all research works processing different sensors merge the data one way or another. Most approaches rely on simple fusion modes: Early fusion (concatenation of input signals [50, 246, 223, 247]), intermediate fusion (concatenation of representation coming from different sensors [248, 206]), or late fusion (average of predictions [249, 25])

Some approaches are sensor-specific, either because they work on textual data [250, 251] and rely on the specific structure of the medium; or because they create an explicit alignment between sensor data with different ranges (*eg* an RGB camera looking forward and a LIDAR sensor gathering information from all directions) [248, 252], which is not applicable in our case as the signals from different sensors are already synchronized.

Others train an autoencoder to reproduce one sensor from the other, and use these autoencoders to generate features that will be processed by a classifier [253]. This approach relies on the fact that minimizing the variation of information between the different sensors' features helps to build efficient features [254]. If this method helps to train a classifier that is robust to missing data, the resulting classifier is worse when all the sensors are available. In our case, as we assume the sensors function properly most of the time, we will not consider this method.

But some methods are still relevant: for instance, Li *et al.* [255] design a network that merges the infrared and RGB information for depth estimation. Their network, baptized IVFuseNet, is halfway between an early and intermediate fusion, for it gathers two sensor-specific convolution modules (one for RGB images and

one for depth map), and one series of convolution layers that uses the internal representation of the first two modules to return a prediction.

Wang *et al.* [256] work with audiovisual videos, and noticed that in some cases, adding the audio to the RGB video only makes the model overfit more. They start from an optimization problem (when each network returns a prediction, find the optimal weights to minimize the overfitting), and derive a formula to produce weights so that the overfitting of the different sensor-specific networks is reduced.

Chen *et al.* [257] designed a network that uses a combination of features from RGB videos and Inertial Measuring Units (accelerometers). The model starts by computing features from sensor-specific channels. Then a dedicated module produces coefficients between 0 and 1 that will be multiplied by these features. The channels are then multiplied by their respective coefficients, depending on the usefulness of each sensor for a given sample.

Liu *et al.* [258] tried to conceive a network that does not require every sensor to be good every time, considering that some of them may have blind spots. Their approach is a modification of a late fusion (they have several sensor-specific models that can return a prediction), with a specific loss that leaves untouched the weights of a model if it is not confident in its prediction.

These methods are always better on the dataset each publication considered, but most works provide little comparisons with other methods if any. In general, most of the publications which deal with multiple sensors compare their architecture to a small subset of baselines, and those subsets do not always overlap between publications. Reviews exist to identify the different fusion modes [259, 260], but they only report the performance of each method on its dataset. We provide a clear comparison of the different fusion modes, including the most basic ones.

However, we should mention that our work are not exhaustive: multimodal data fusion is an extremely broad topic, and there are methods that we did not consider. As an example, Paul Liang, a PhD student at Carnegie Mellon University, recently published a reading list on multimodal deep learning [261], that includes as many as 500 publications. As a comparison, we only thirteen algorithms, amore thorough investigation would likely require a complete thesis in itself.

5.2 An inventory of fusion modes

We selected several data fusion methods that are relevant in our setting for comparison. The names may vary, some methods might even not be named. One important note: most of these modes are equivalent to our baseline architecture when a single modality is used (if they leave the possibility to use a single sensor). The exceptions are the bottleneck filters (section 5.2.1) and attention (section 5.2.2).

5.2.1 Early fusion

Early fusion modes consist in giving all signals to a single neural network. The only difference between them is how the input signals are put together before they are processed by the neural network.

- *Time concatenation*: the input spectrograms are concatenated along their temporal axis.
- *Frequency concatenation*: the input spectrograms are concatenated along their frequency axis (after log interpolation).
- *Depth concatenation*: the signals are put together like the channels of a RGB image: each convolution filter of the input layer has access to the same portion of all signals at the same time.

Figure 5.1 illustrates these three methods.

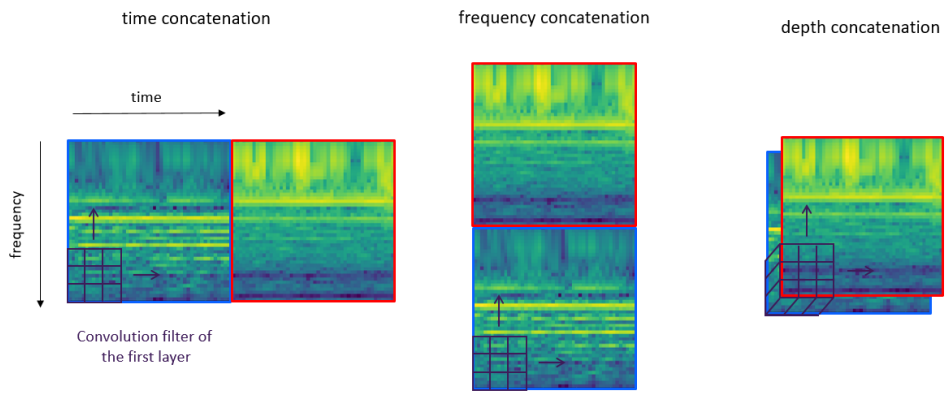


Figure 5.1: the three early concatenations

Note that early fusion is not always possible for every problem. As Moya-Rueda *et al.* notice [214], using these modes require having the sensors synchronized (which implies having the same frequency) in order for the fusion to be relevant, because temporal measures of different sensors are put on the same level.

At first, concatenating signals along their time or frequency axis seems surprising, as the resulting 'spectrogram' is not homogeneous. A convolutional network using such inputs would have no way to know from which sensor comes a given pattern. However, these fusions are similar to a more relevant setting: features concatenation. In order to understand this idea, one should look at a characteristic of convolutional neural networks: even if the receptive field of a given neuron is large, in practice, the features obtained after each convolutional layer retain the spatial information. In other words, the features at a given location of a feature map will be mainly influenced by the patterns at the corresponding location in the input spectrogram (or image), to the extent that one can use a classification dataset to train a segmentation network, and obtain acceptable results with minimal modifications to the classification architecture [262, 263]. As we use a flatten layer to obtain a single feature vector, the scalars of the vector each have a distinct spatial origin. Given that the operation that follows is a matrix multiplication, the network can adapt the weights of the Fully-Connected Layer to make the distinction between features coming from each location of the spectrogram. In particular, with time or frequency concatenations, the network can still distinguish the information coming from each sensor, even if its convolution filters browse the whole spectrogram. An illustration is available in fig. 5.2.

Bottleneck filters

Bottleneck filters (as can be seen in *eg* [246, 264]) are a special type of filters aiming to replace a convolution layer. We begin by concatenating the spectrogram depth-wise (similarly to depth concatenation). We then replace the first convolutional layer (containing 16 3×3 filters that could see all sensors) with a succession of two particular layers: the first one contains only a 1×1 filter, that can see all the sensors and returns a feature map with a depth of 1. This layer will have to learn a good combination of the input modalities. The second layer contains 16 filters with size 3×3 which will see the combination learnt by the first layer. The feature computation happens here. No activation function is added between these two layers. The idea is to separate the fusion of sensors (first layer) from the feature computation, as those steps happen at the same time (in the first convolutional layer) with depth concatenations.

5.2.2 Intermediate fusion

Intermediate fusion consists in merging together the features produced by different sensor-specific networks, so that a single classification network can process them. As it requires the classifier to have some kind of internal representation, it is most adapted for deep architectures.

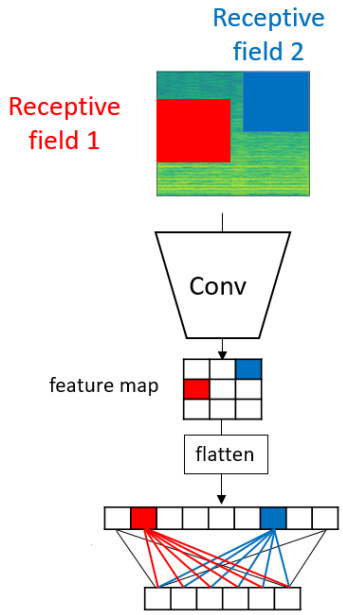
Feature concatenation

The idea behind feature fusion is to let the convolutional layers compute features, before giving both features to the next layer. The rationale behind this method is to allow each convolution module to generate its own relevant features. For RGB-D images, concatenating features from the RGB image and features from the depth map is better than a depth concatenation [248].

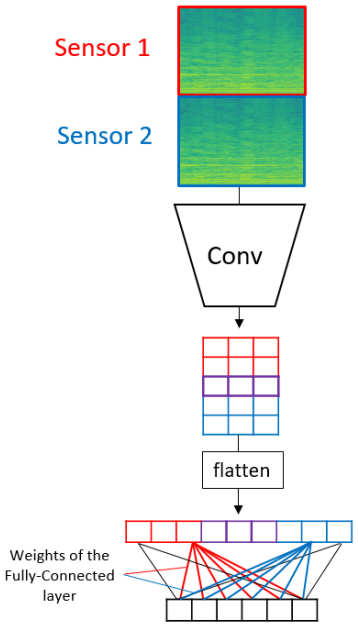
Given the remark made in the previous section, this method seems similar to the time and frequency concatenations. There are two differences between the architectures (fig. 5.2):

- the Feature concatenation allows the network to have a series of convolution filters dedicated to each modality, whereas the input concatenations impose the network to have the same convolutional filters.
- With frequency and time concatenations, the convolution filters can go close to the border of the signals, which means some of the features will come from both sensors. This is impossible with feature concatenation, where the different sensor's features are not merged until after the last convolutional layer.

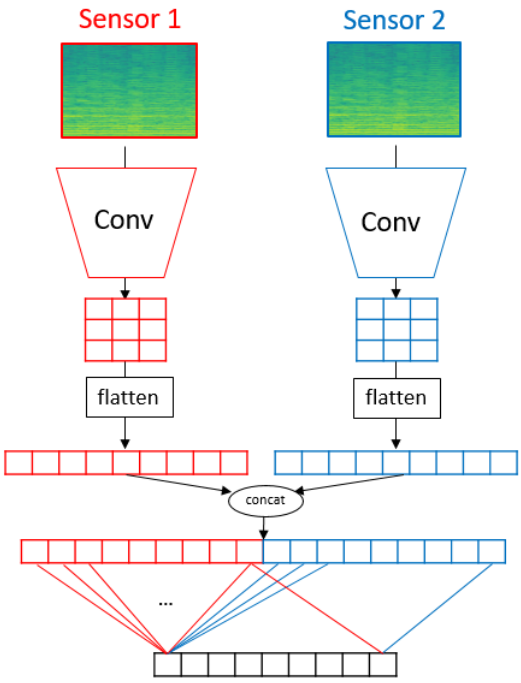
These reasons explain why the feature concatenation is expected to be better than the frequency or time concatenations.



(a) How a neural network learns local features



(b) Frequency concatenation



(c) Feature concatenation

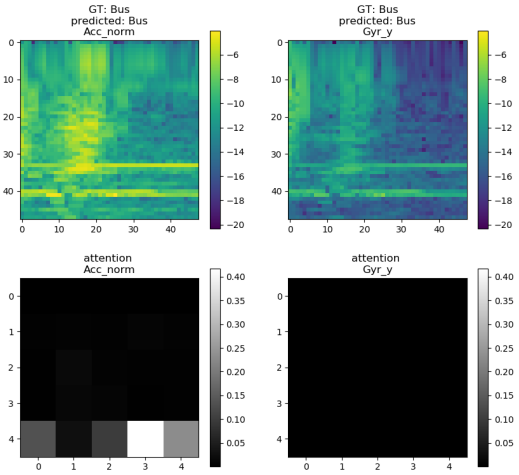
Figure 5.2: An overview of the difference between frequency and time concatenations, and feature concatenation. As the features learnt in the final feature map still retain some spatial consistency (a), a network using frequency and time concatenations (b) can still distinguish between the features from each sensor. The main difference with feature concatenation (c) is that the network can now learn sensor-specific convolution filters, which was impossible with the two early fusion methods.

Attention

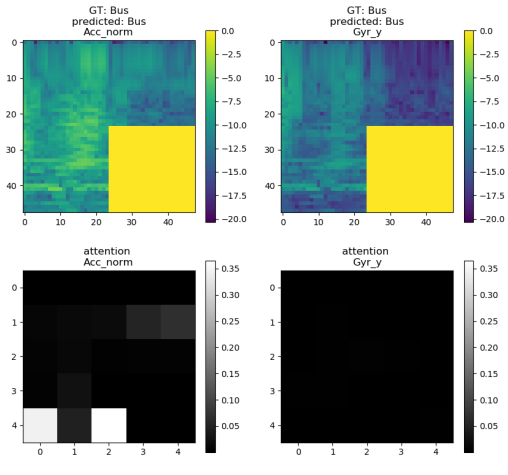
The key idea behind attention is to allow the network to allocate a different importance to every feature, depending on the sample. The implementation is similar to one the from [119] and [121]: for each pixel of the final feature map of each sensor, the network produces a scalar between 0 and 1, which encodes the importance to assign to the feature. The scalar is multiplied to the feature, the results are aggregated (summed) over the spatial dimensions and sensors, and fed into the next fully connected layer (see fig. 5.4a). This allows the network to modulate the importance it gives to each sensor, depending on the sample it has to classify.

We can illustrate the behaviour of our attention mechanism by looking at the attention when the input samples are flawed. During the training time, we hide the bottom-left hand corner of the spectrogram: a quarter of the pixels are set to zero. Then, at test time, we submit clean spectrograms, along with their obstructed version, and compare the attention maps with these two samples.

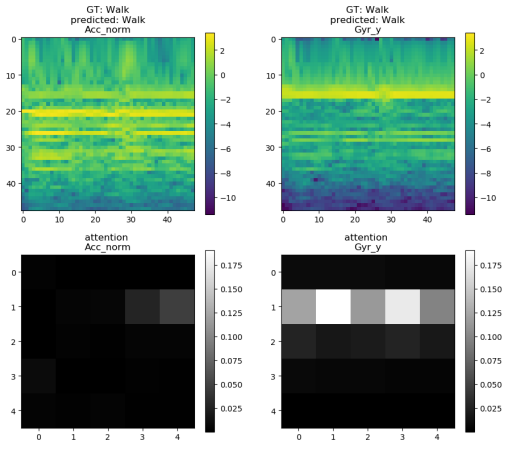
When we compare a clean sample with its obstructed counterpart, the network learns to distinguish between clean and obstructed spectrograms: we can see that, when the network dedicates some attention to the bottom right weights on a clean sample (fig. 5.3a), setting the values to zero removes all the attention: our network learnt to ignore the noise in a representation (fig. 5.3b). Obviously, when the network does not dedicate any attention to the bottom right-hand corner, obstructing this part of the image doe not change the attention map.



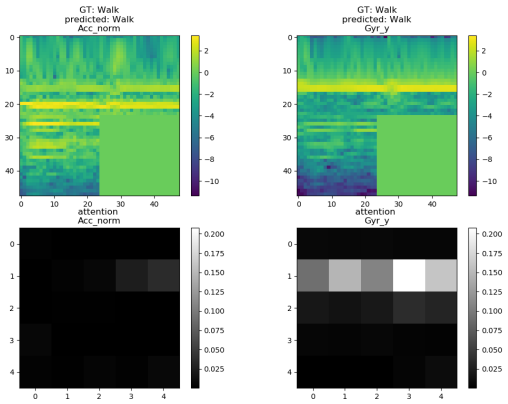
(a) A clean sample of Bus



(b) the same sample, with one corner obstructed



(c) A clean walk sample



(d) The obstructed Walk sample

Figure 5.3: A sanity check of our attention mechanism. We can see that, when a network was trained with obstructed samples, it learns to ignore constant portions of the spectrogram by assigning them no attention

Note that for this experiment to work, submitting obstructed samples during the training is essential: if all the training samples were clean, the network would not know that a uniform, zero-square corresponds to noise, and it would pay attention to these regions.

Selective fusion

Chen *et al.* [257] conceived a neural network that is based on attention (called 'soft attention' network), with a notable variant: the network still produces a scalar between 0 and 1 for each pixel of each sensor-specific feature map; these scalars are still multiplied to their corresponding features, but features are not aggregated. The rescaled features are instead flattened, and given to the next FC layers. Figure 5.4b illustrate this architecture

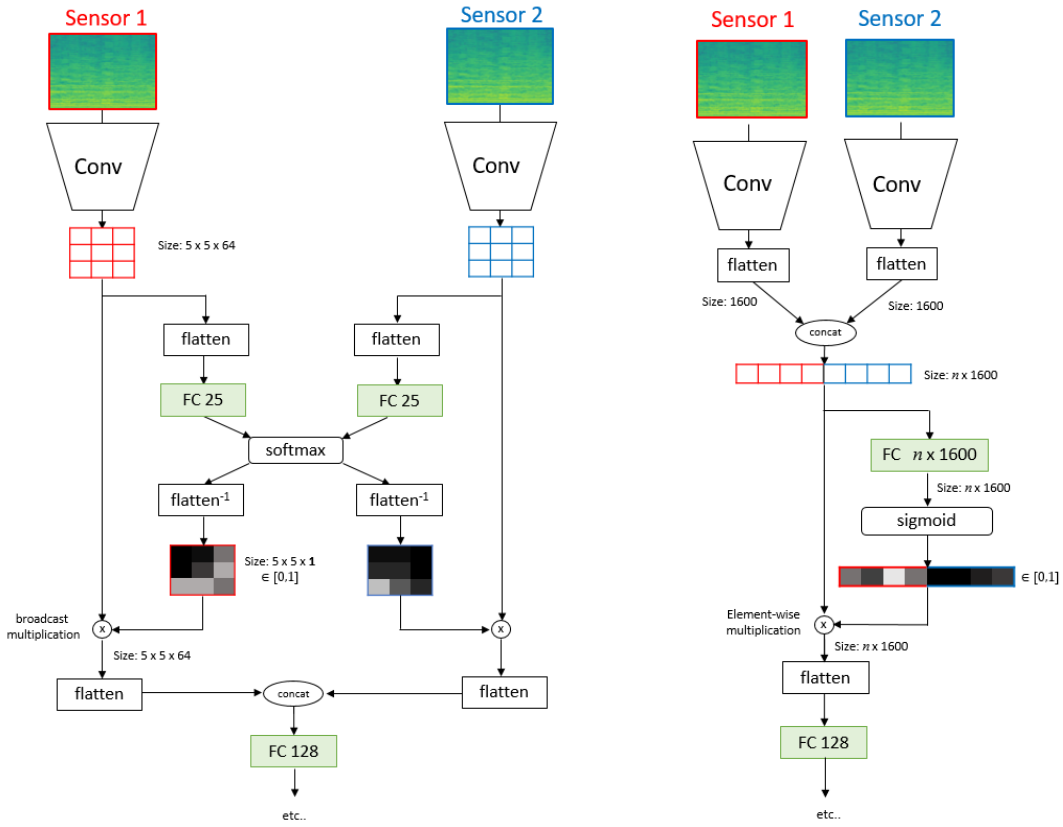


Figure 5.4: An illustration of the architectures of the baseline attention (5.4a) and the selective fusion (5.4b).

5.2.3 Late fusion

Late fusion methods rely on using only the predictions (logits or probability vector) of different sensor-specific models. As they allow to use any type of model (both Machine Learning and Deep Learning), these methods are the most flexible category.

- *Probability fusion*: Each network produces a probability vector (after the softmax), indicating the prediction of the network. With the probability fusion, we simply compute an average of probabilities. If p_c^i is the probability network i assigns to class c, the final probability vector of the ensemble of the n models is given by
  $\forall c, p_c = \frac{1}{n} \sum_{i=1}^{n} p_c^i$.

- *Logits fusion*: To merge the logits, we extract the vector before the softmax. This vector of logits indicates whether the sample is likely to belong to each class (the higher the logit of one class, the more likely the sample is to belong to this class). We compute the average, for each class, of the logit each network assigns to each class, before using a softmax to obtain a single output probability.

- *Weighted fusions*: With the two previous modes (probabilities and logits fusion), the average was unweighted, which means a 'bad' sensor is given as much importance as a relevant one. To avoid it, we try letting the network learn the weight to assign to each sensor. With both methods, we compute a weighted sum of the predictions of the sensor-specific models. For instance, the weighted probabilities average is:
  $\forall c, p_c = \sum_{i=1}^{n} \alpha_i p_c^i$, where $\sum_{i=1}^{n} \alpha_i = 1$.

The implementation of this mechanism is not straightforward, as the α_i have to remain within $[0, 1]$ and sum to one. In order to learn the coefficients of a weighted sum, each network coefficient is obtained using a softmax of real-valued weights:

$\forall i, \alpha_i = \frac{exp(w_i)}{\sum_{j=1}^{n} exp(w_j)}$

Thus, the parameters w_i can be learnt on $\mathbb{R}$ by gradient descent, and the α_i can be between 0 and 1, and sum to one.

Learn to combine modalities in multimodal deep learning

Liu *et al.* [258] noticed that sensors might not be relevant everytime, and instead carry only a partial information. However, in most settings, the sensor-specific networks are trained to predict right every time, leading to possible overfitting.

They design a loss function that aims to reduce the penalty for sensors that are not confident in their prediction. They start by considering a model that simply computes the class-wise product of the probabilities each model returns: $p_c = \prod_i p_c^i$. If gt is the index of the ground-truth class, the cross-entropy loss is given by:

$\mathcal{L} = -log(p_{gt}) = -\sum_i^n \log(p_{gt}^i)$

They introduce a coefficient q_c^i (for all classes c) as follows:

$q_c^i = [\prod_{j \neq i}(1 - p_c^j)]^{\beta/(n-1)}$

where n is the number of sensors and β is a hyperparameter that controls the intensity of the correction. Its value must be between 0 and 1 and was set to 0.5 in our experiments.

This coefficient is used to help compute a new loss:

$\mathcal{L} = -\sum_i q_{gt}^i \log(p_{gt}^i)$

The general idea is that when a sensor i is sure that the sample belongs to the class c, the probability p_c^i will be close to 1, which means the coefficients $(q_c^j)_{j \neq i}$ will be close to zero. This means the loss will leave the models j alone and change the weights of model i.

Note that this approach is equivalent to replacing the probabilities p_c^i by $(p_c^i)^{q_c^i}$ with a model that computes a product of the sensor-specific probabilities and a classic cross-entropy loss.

Gradient Blend

When studying networks using audiovisual content (videos), Wang *et al.* [256] noticed a particular phenomenon: in most cases, adding audio data to the image model does not help. In fact, the image-specific model is so good that adding the audio only make the whole model overfit. They start from a model where the final probability p_c is a linear combination of the probabilities p_c^i, and solve an optimization problem to find optimal weights from expressions of the train and validation losses at $t = 0$ (before the training) and at the end of the training

For each sensor i, define:

$O = (\mathcal{L}_{t=0}^{train} - \mathcal{L}_{t=0}^{val}) - (\mathcal{L}_{t=end}^{train} - \mathcal{L}_{t=end}^{val})$

$G = \mathcal{L}_{t=0}^{val} - \mathcal{L}_{t=end}^{val}$

The relative weights of each sensor are given by: $w_i = \frac{O_i}{G_i^2}$

Their method, named *Gradient Blend* is likely to succeed in our case, because one signal (the accelerometer) is significantly better than the others, especially the magnetometer.

However, Gradient Blend requires using a validation loss to compute each weight. To avoid contamination and to allow for a fair comparison, we will not use our validation set to compute the validation loss. Instead, we estimate it by further splitting our train set into two subsets: the first one gathers 80% of the training set and is used to train the weights of the sensor-specific networks, while the other 20% are used to compute the 'validation' losses that will give the weights of the models. When the global network is training (that is, when all the weights w_i are defined), the two parts of the training set are merged back together, and used to tune the weights by gradient descent. As a result, we can evaluate this global model on our validation set safely.

Late fusion modes: is it better to train the models separately or jointly ?

The methods belonging to the 'late fusion' category rely on generating one model per sensor, and merging the predictions of each complete model. When using Deep Learning models, a new option arises: the different models can be trained either separately (each model tries to do its best using the sensor it has access to), or jointly (all models share a common loss function). However, this choice is not binary: one can decide to begin with a separate training, before finetuning the models jointly. We did not consider the opposite, that is, to begin with a joint training, and end with a separate one. As the late fusion models share a common loss at test time, separating the models mid-training seems unlikely to yield good results. To know which proportion of joint training is optimal, we used the following protocol: we start with the four sensors, and for each fusion method that allows doing so (the late fusions), we train the network separately during a proportion p of 50 the epochs, before merging the models, and training them together for the remaining $1 - p$ of the epochs. Table 5.1 shows the result: for the logits and weighted logits methods, no proportion is statistically better than the others. The same applies for Gradient Blend, although the performance is significantly lower. For this mode, we had to do an exception: as we need to train the sensor-specific models separately at least once (so that the parameters O_i, G_i make sense), $p = 0$ corresponds to 1 epoch where the models are trained separately. For all the other modes, $p = 0$ means that the models are trained jointly.

For the methods that deal with probabilities, training the individual models together impacts the performance negatively, even if the collective training is as short as 5 epochs (corresponding to $p = 0.9$). For the weighted probabilities, the explanation is the following: a neural network always tries to maximize the output probability of the correct class, on average, on the whole dataset. As probabilities are bounded (within [0, 1]), and because the α_i stay the same for all the samples, the only way a network can increase the average correct probability on the training set is by increasing the weight α_i of the best sensor-specific network available (the accelerometer network): the networks overfit. At the end of the training, the global network only assigns a weight of 1 to this sensor, and a weight of zero to all the other sensors, which is why its performance is equal to the performance of a single network using the accelerometer signal. We confirmed this by explicitly looking at the weights the network assigned to each model (results not shown).

This is not the case with weighted logits, for similar reasons: in order to maximize the output probability on the training set, the network can tune the logits, which are easier to work with since they are not bounded. As the logits change from a sample to the other, this prevents the network to listen to only a single modality.

For the next experiments, the proportion will be set to $p = 0.5$ for the logits, weighted logits, and gradient blend fusion, and to $p = 1$ for the probabilities fusion. Choosing $p = 1$ for the weighted probabilities fusion is irrelevant, for this method is equivalent to a mere probabilities fusion when $p = 1$ (all models are trained separately, the weight are not updated and left equal to their starting value of $\frac{1}{n}$). This is why we also choose to set $p = 0.5$ for the weighted probabilities.

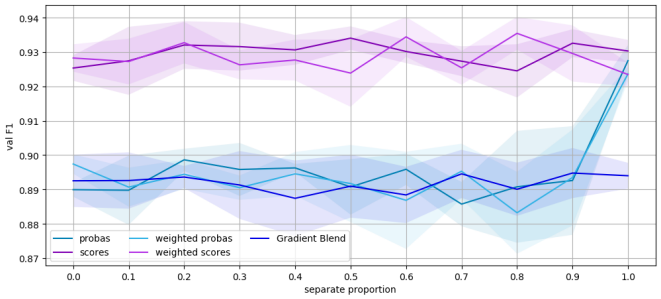


Figure 5.5: The mean and standard deviation of different late fusion methods, as a function of the proportion of epochs where the sensor-specific models are learnt separately.

5.3 A benchmark of fusion modes

Table 5.1 gives the results of each fusion method, applied to four sensor combinations. We notice that, given the standard deviation of the experiments, most fusion methods have statistically similar performances. Two

of them, however, seem particularly worse than the rest: bottleneck filters and attention. Without being exceptionally bad, the methods Gradient Blend, Learn to Combine, and weighted probabilities also yield consistently worse results than the others. This is all the more surprising that all the methods from the literature had rationales that apply to our case, and that these methods succeeded in their respective tasks. If the works that compare two basic fusion methods sometimes report that these methods have similar performance (about one percentage point in [256, 255, 265]), some also report differences in the fusion methods ([25, 258, 119]). In all cases, the more complex methods we took from the literature should have differed from the rest.

Maybe our models are not deep enough: for instance, given that there are only two fully-connected layers, only two layers separate the intermediate fusion from the late fusion methods. The same goes for the fact that there are only three convolutional layers. Also, the decision boundary might be "too simple": if the network classifies the Run class using a linear boundary (section 3.4), chances are that the other classes are detected using a "simple" expression of the input data. In this case, all fusion methods would be equivalent because they would all equate to merging the input features.

Finally, a third hurdle to an efficient comparison is the size of the training dataset: one could argue that complex methods require more training samples to be efficient, but achieve better results on larger datasets (an opinion which we somewhat illustrated in fig. 3.3)

If one still wanted to follow our conclusions, we could give them the following advice: as most fusion methods have the same performance, we recommend using the most simple ones: time, frequency, or feature concatenations; feature concatenation; probabilities average (where all models are trained separately), logits average.

method	early fusion				intermediate fusion			late fusion					
	time concat.	freq concat.	depth concat.	Bottleneck filters	features	Selective Fusion	attention	probas	logits	Weighted probas	Weighted logits	Gradient Blend	Learn to combine
$\|Acc\|, Gyr_y$	90.89 ±0.57	90.46 ±1.08	90.57 ±0.94	88.70 ±1.30	90.83 ±1.54	90.01 ±0.41	84.11 ±1.30	90.26 ±0.56	90.95 ±0.37	88.85 ±0.54	90.89 ±1.13	89.18 ±1.10	90.10 ±0.66
$\|Acc\|, \|Mag\|$	90.78 ±0.66	91.37 ±0.49	91.83 ±0.35	85.83 ±4.45	91.74 ±0.46	90.62 ±1.03	86.67 ±1.01	90.75 ±0.77	91.66 ±1.35	88.04 ±0.93	92.17 ±0.59	89.53 ±0.64	87.37 ±0.65
$\|Acc\|, Gyr_y,$ $\|Mag\|$	91.36 ±0.74	92.13 ±0.90	91.91 ±0.88	87.01 ±2.16	91.87 ±0.64	92.39 ±0.87	87.85 ±1.05	92.33 ±0.61	92.55 ±1.08	89.40 ±0.55	92.98 ±0.37	89.47 ±0.92	89.56 ±0.98
$\|Acc\|, Gyr_y,$ $\|Mag\|, Ori_w$	92.32 ±1.18	92.30 ±0.54	91.23 ±1.25	84.59 ±5.52	92.51 ±1.01	92.93 ±0.60	87.56 ±0.62	92.43 ±0.40	92.83 ±0.19	89.43 ±0.49	93.01 ±0.36	89.36 ±1.24	91.41 ±1.11

Table 5.1: The mean and standard deviation of the validation F1-score of each method, for different fusion modes

5.4 Decorrelated networks

5.4.1 Principle

When observing the results (see table 5.1), we can notice an interesting pattern: with many of the "simple" fusion methods (frequency and depth concatenation, features fusion, logits and weighted logits) the combination of the norm of the accelerometer and the norm of the magnetometer seem better than accelerometer and gyrometer, even though the gyrometer is better individually than the magnetometer (see the results in table 2.5 in chapter 2). One obvious explanation is that the norm of the magnetometer carries a different kind of information from the accelerometer: the latter is mainly affected by the dynamics of the sensor; while the former changes mostly when the user is inside a metallic cabin. This can be confirmed by noting that the average power spectra of accelerometer and gyrometer signals are much more similar to each other than the accelerometer and magnetometer (fig. 3.6). If choosing different input sensors leads to better results, we might increase the performance of a network by forcing it to learn decorrelated features. We will develop this idea in the rest of section 5.4.

The first idea that comes to mind is to use a simple L1 loss to prevent the features of both networks from being too close to each other: if X_1 and X_2 are feature matrices extracted from the first and second network, respectively (see fig. 5.6), the weights of the network would be trained with an additional constraint: $\mathcal{L} = -|X_1 - X_2|$, in addition to the cross-entropy (classification) loss.

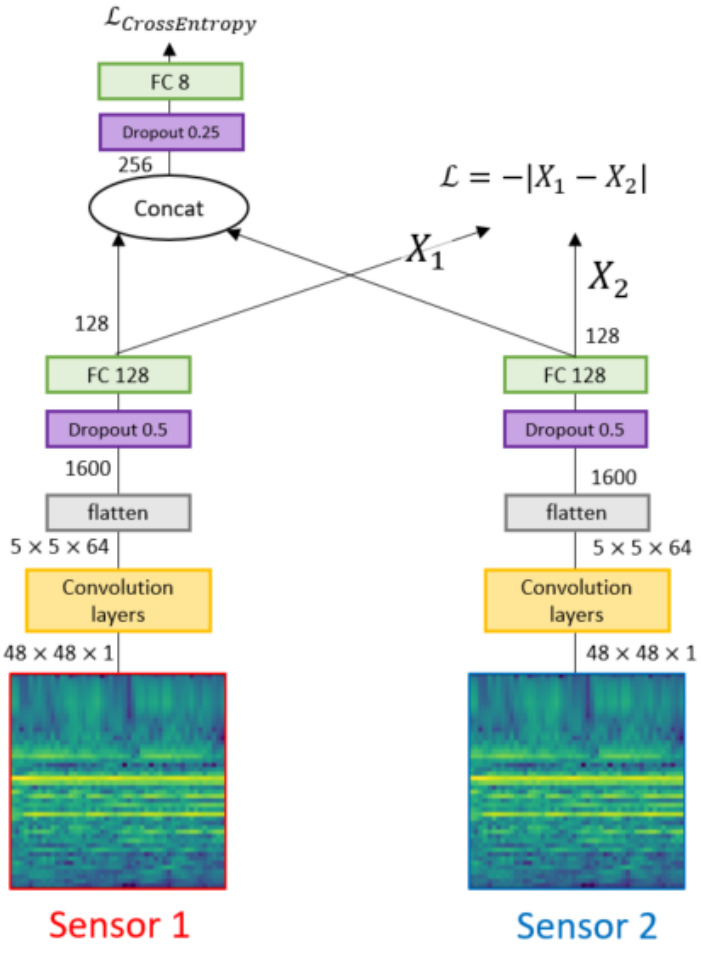


Figure 5.6: A naive way to force the features to be complementary.

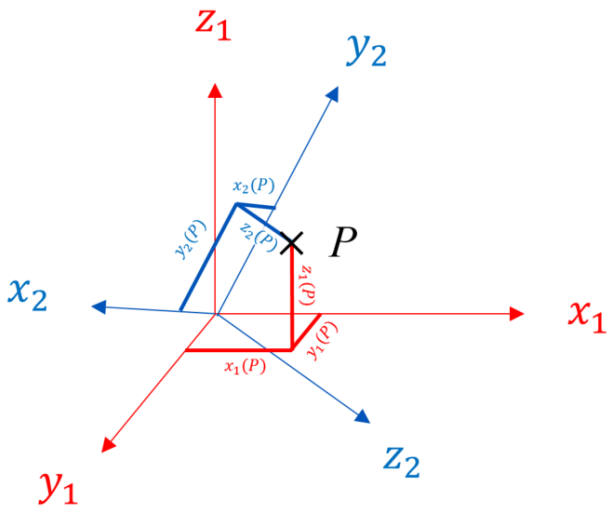


Figure 5.7: An illustration of why we cannot use a simple L1 norm to separate the features of two bases. The same point P has two coordinates $X_1 = (x_1, y_1, z_1)$ and $X_2 = (x_2, y_2, z_2)$ in the red and blue coordinate systems (respectively), and the corresponding features do not match (*e.g.* $x_1 \neq x_2$). Note that in this illustration, the two bases correspond exactly, that is, it is possible to recompute exactly the features (x_2, y_2, z_2) using x_1, y_1, z_1; and inversely. In the general case, a complete alignment of the bases might not be possible, but the CCA offers mathematical guarantees that we obtain the best alignment possible.

However, comparing two features as if they existed in the same space make no sense: for instance, if the scalars of the first feature are a mere permutation of the features of the second one, the vectors will look different while representing the same information (see fig. 5.7). In order to compare them, we need an operation that finds a base change that aligns the coordinates as much as possible. We use Canonical Correlation Analysis (CCA) [266], an operation which finds two base changes (B_1, B_2, one for each set of features), such that the *correlation* between features is maximized. This operation has already been used for data fusion [249, 267], but in a way that makes it almost useless (the next chapter explains why). In our case, to force the features to be different, the loss now tries to separate the features *after* the base change: as the new features are expressed in the same base, we can separate them using a L1 loss, which will ensure the information they carry is different. The weights of the sensor-specific convolution layer are trained with a sum of classification loss and decorrelation loss: if X_1, and X_2 are the feature matrices after the two sensor-specific convolutional layers, and $X_1' = B_1 X_1$ and $X_2' = B_2 X_2$ are the result of the base change after CCA, the complete loss is:

$$\mathcal{L} = CELoss + \mu \mathcal{L}_{decorr}$$

It is the addition of the classic cross entropy loss and a novel term, we name *decorrelation loss*:

$$\mathcal{L}_{decorr} = -|X_1' - X_2'| = -|B_1 X_1 - B_2 X_2|,$$

As for μ, it is a hyperparameter we will try to optimize.

At test time, the decorrelation loss is not used, and the network is similar to the feature concatenation network. The original CCA operation is only defined for two sets of features. If expansions of the operation exist for more than two sensors [268], we will first use only the two-sensor CCA. We focus on the accelerometer with the magnetometer, even if we will consider the accelerometer with gyrometer for the comparison with other fusion methods.

The next section presents the many additional changes we needed to apply before obtaining significant results and the two ways we found to compute the Canonical Correlation Analysis.

5.4.2 Experimental protocol

Even if we introduced a computation of new variables with the same meaning in figure 5.6, to decorrelate the feature efficiently we would need to change the place where the features are extracted: in fig. 5.6, the

features are "too close" to the classification layer. For reasons we can only develop in the next chapter, if we tried applying a L1 loss to make the features more distant from each other, we would directly go against the classification loss. This is why we extract the features one layer before. Now, in our implementation, the features are only extracted after the flatten step (see fig. 5.8). The reason why the features are extracted before the dropout is not insignificant: if the features X_1, X_2 were extracted after the dropout layer, we would risk having the phenomenon described in fig. 5.9. This is why the features are computed after the flatten step, and before the first dropout and fully connected layer. We considered applying the decorrelation loss to features extracted even sooner (between convolution layers), but the sheer number of features prevented the CCA from converging.

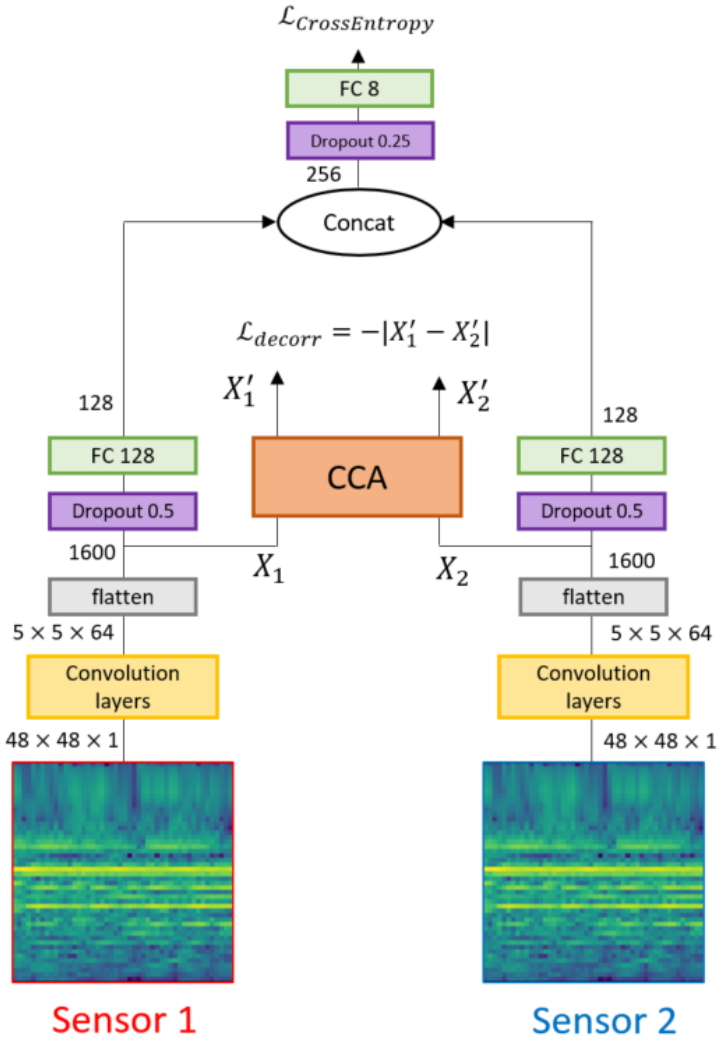


Figure 5.8: The complete architecture of the networks we forced to produce decorrelated features.

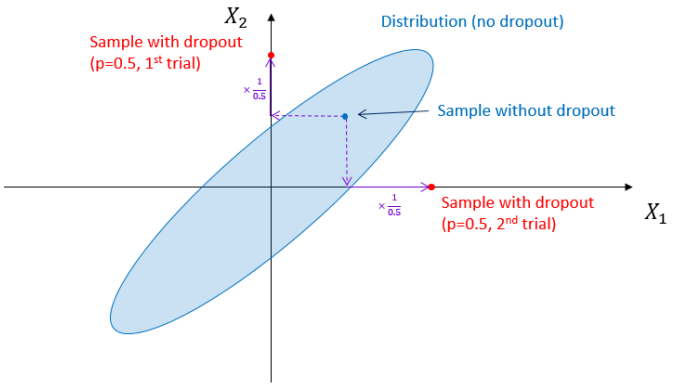


Figure 5.9: The reason why the features that are used to compute a decorrelation loss are extracted before any dropout is applied. Dropout, by removing some of the features at random, destroys the correlation between realizations.

Now that we know where to extract the features, we can compute the aligned representations. We found two ways to do so:

- *classic CCA*: Canonical correlation analysis offers an explicit solution, relying on inverses of covariance matrices. We use the implementation from [269] the following way: at every batch, after the features are computed, we use the whole training set to evaluate the correlation matrices on the complete dataset, and deduce the feature matrices X_1', X_2' for the current batch. Then, the decorrelation loss uses these feature matrices on this batch to train the weights of the network. At every batch, we compute the explicit CCA between the two sets of features on the training set. We then use the base changes B_1, B_2 to evaluate the loss on the features from the current batch. To compute the CCA, we use the code from Raghu *et al.* [269].

- *deep CCA*: the previous method has two drawbacks: first, it requires to use the whole training set to compute the result of the CCA operation. Secondly, the CCA objective is computed using a script outputting the base changes. There is a risk that between two batches, the result of the CCA (and thus, the expression of the loss) could change dramatically. To avoid this, we replace the explicit CCA computation with an iterative solution. We use *deep CCA* ([270]), a neural network that tries to maximize the correlation between two feature matrices. Deep CCA consists of two Fully Connected neural networks, both of them using a vector as an input (in our case, a vector of 1,600 features for each network) and outputs a vector of smaller dimension (chosen arbitrarily to be equal to 128). In between, we tried using one, two, or three fully-connected layers with a hidden feature size of 256. The weights of these deep CCA networks are trained to *maximize* the correlation of the features, while the decorrelation loss tries to *minimize* the correlation between them. However, the decorrelation loss only changes the weights that participated in the computation of the features (*i.e.,* the weights of the convolutional layers). To enforce some continuity of the CCA between the batches, we alternate one batch of deep CCA with one batch of the classification network.

  Note that we used the code from the original publication ([270]). In practice, the computation of the loss requires the correlation matrix to be full rank, which is only possible if there are more samples than features in the feature matrices $X_{1,2}$. As these matrices have 128 features each, we set the batch size to 512 for deep CCA networks. We made sure that this hyperparameter does not change the performance of the baseline networks (the networks that each use one sensor) before implementing it for deep CCA models.

We presented the list of caveats we need to consider before implementing a decorrelation loss in practice. Now that we can run the networks, let us see if the decorrelation loss actually improves the performance of the network.

5.4.3 Results

As we said in the previous section, we train two networks using a loss equal to $CELoss + \mu\mathcal{L}_{decorr}$, where μ is a hyperparameter we will change. For all our results, we will consider the following values for μ: $[-10^{-1}, -10^{0}, -10^{-1}, -10^{-2}, 0, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}]$: we use a logarithmic scale with a factor of 10 (the borders of the interval were chosen so that the loss goes under 80 %). Notice also the fact that we consider $\mu = 0$, which corresponds to a situation where there is no decorrelation loss. we also add the negatives values for μ, which correspond to cases where we force the networks to produce *correlated* (*i.e.,* redundant) features. We will look at the difference between $\mu = 0$ and $\mu > 0$ to know if a decorrelation loss helps the network. Similarly, if the performances are higher for $\mu<0$, than for $\mu = 0$, this means that we help the network y forcing its features to be more correlated.

As we mentioned in the previous section, in the case of the deep CCA networks, we consider having one, two, and three fully-connected layers to compute the correlated components.

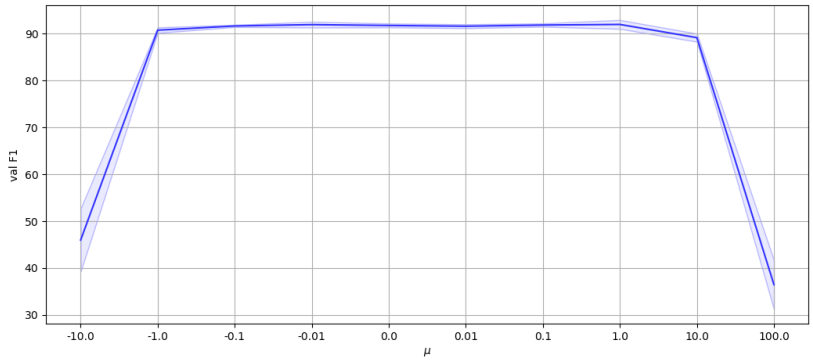


Figure 5.10: The performance of a neural network using a decorrelation loss computed using the theoretical expression of the CCA.

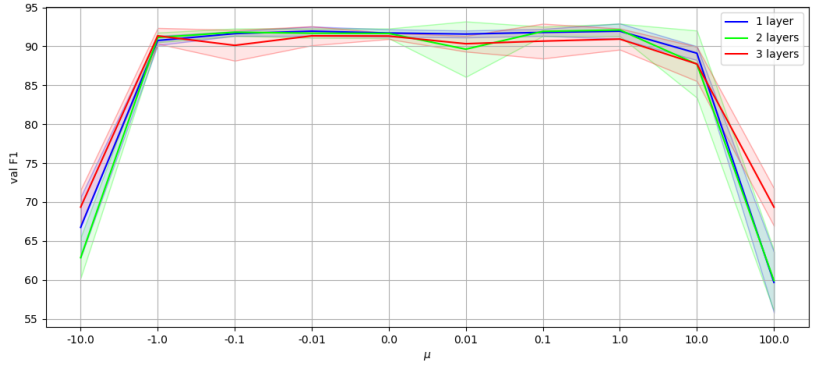


Figure 5.11: The performance of a neural network using a decorrelation loss computed using the a DeepCCA network.

As figures 5.10 and 5.11 indicate, .When the values of μ are too high in absolute values, the classification performance decreases dramatically, because the decorrelation term is so high that the objective starts to become so important the network neglects the classification task. More interesting is the allure of the curves; they look to be constant in the interval $\mu \in [-1, 1]$, and decrease when μ is out of these borders. This implies that a network is able to know automatically know how much it should correlate its features. We will demonstrate this fact in the next section.

5.4.4 Why did the decorrelation loss not help

In this section, we will try to know whether the networks are helped or impaired by a decorrelation loss. To do so, we will measure the *scalar product* between the gradients of the two losses. The weight of a neural network, put side by side, form an immense vector in a space of extremely high dimension (typically thousands to billions). To train the network, we compute the gradient of the cross-entropy loss, and move the weights of the network in this direction on an infinitesimal distance. We repeat this process at every batch until the network eventually stops improving its performance.

In our case, we have two losses: the cross-entropy loss, for classification, and the decorrelation loss. We can quantify their agreement by looking at the scalar product of the losses' gradients. If the two gradients pull the weights in the same direction (resp. opposite directions), the scalar product between them will be positive (resp. negative). In short, the scalar products between the gradients quantify the agreement between their respective losses.

We compute the scalar product between the gradients at each batch of the whole training process and display it to know whether our intuition was true. We train two models with $\mu = 0$ to know if decorrelating the features helps or impairs the networks. As an illustration, we display the scalar products of the gradients of the losses for each of the sensor-specific layers, and for the complete network (the union of the sensor-specific layers and the last fully-connected layer in fig. 5.8)

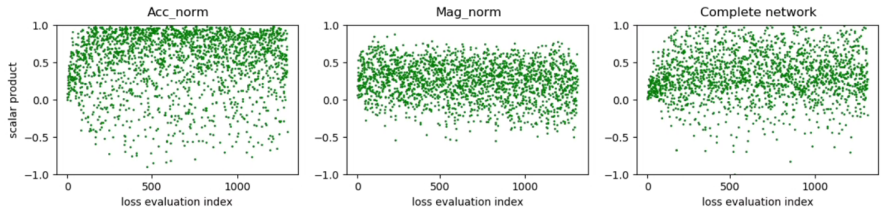


Figure 5.12: The scalar products between the classification and decorrelation losses at every batch, on the classic CCA network

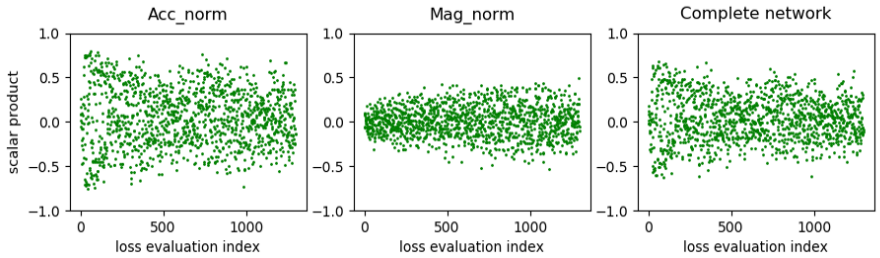


Figure 5.13: The scalar products between the classification and decorrelation losses at every batch, on the two-layer, deep CCA network

The results are presented in figures 5.12 and 5.13. They differ for the classic and deep CCA models. For the classic model, we see that the scalar product between the loss is mostly positive and close to 1 (its theoretical maximum). This means that the decorrelation and classification losses pull the network in the same direction. For the deep CCA model, however, the sign of the scalar product changes a bit more between positive and negative values. However, given that the deep CCA networks are learning at the same time as the classification model, we cannot be sure that they reach the (the whole point of using neural networks was to have a slow-moving model to compute the CCA), we cannot conclude from the deep CCA networks because we cannot be sure that their decorrelation loss is actually relevant.

Starting from the observation that a couple of different sensors perform better than a couple of similar sensors, we tried to force the networks to produce features that are complementary across sensors. We notice that forcing the networks to produce decorrelated features does not help it to improve their performance, because the networks are perfectly able to optimize the good amount of redundancy themselves.

5.5 Evaluations on the test set

If the data fusion method does not seem to change the results much, we might wonder if adding two sensors (magnetometer and orientation) to the two sensors of the baseline (accelerometer and gyrometer) helps the network. To compare our approach against the state of the art, we select the best fusion method and sensor combination in table 5.1 (that is, a weighted logit fusion of the four sensors), and train it 5 times on the union of train and validation sets, before evaluating the F1 score on the held-out test set of the challenge. to choose the "best" data fusion method, we simply choose the method that has the best average performance. Even if there is a likely risk of this choice relying on insignificant measurements, we need to select a single method and see no other reasons to choose than maximizing the average validation performance.

Approach	Models	fusion method	sensors	F1-score
Four-sensor fusion	CNN	weighted logits	Acc, Gyr Mag, Ori	$89.96 \pm 0.07\%$
Baseline [50]	CNN	frequency concatenation	Acc, Gyr	88.83%
Best SHL 2018 submission [31]	ML + DL ensemble	HMM on predictions	All	93.86%
Posterior improvement [84]	ML + DL ensemble	HMM on predictions	All	94.9%

Table 5.2: The results on the held-out test set. We repeated our training process five times and display the average and standard deviation

As Table 5.2 shows, the results are surprising: not only the F1-score is still significantly lower than it was on the validation set ($89.96 \pm 0.07\%$ and 93.01 ± 0.36, respectively), but it is barely superior to the baseline we chose (88.83%). In practice, the use of additional sensors (and the energy consumption that comes with it) might not justify the performance gain (about one percentage point). These two Deep Learning approaches are still inferior to the methods in [31, 84], which use an ensemble of Convolutional Neural Networks and Machine Learning models with handcrafted features, and merge the predictions of the models by giving them to a meta-classifier (a Hidden Markov Model). In practice, halving the error might be worth the extra complexity and computational costs of implementing handcrafted feature computation and using all sensors.

5.6 Conclusion

In this chapter, we focused on the choice of a data fusion architecture to take into account the information from all four sensors we chose. The most important contribution of this chapter is the benchmark we realised: we selected thirteen data fusion algorithms in the literature to evaluate them and found that no one was particularly better than the others. Afterwards, we tried designing a novel algorithm that forces the networks to learn complementary information, which did not yield results that differ significantly from the other architectures. However, we demonstrated that the networks learn to correlate the features by themselves, and trying to intervene in one way or another is only detrimental. Finally, we wanted to see if adding two sensors to the original publication, as well as changing the fusion method, could increase the test performance. If we noticed an increase compared to the publication we based our baseline on, the increase in performance was too low to justify the additional complexity.

Chapter 6

A study on Canonical Correlation Analysis

In the previous chapter, we mentioned a data fusion method relying on CCA we found in the literature. Canonical Correlation Analysis operation has had many uses with deep learning, the most important of which is to quantify the similarity between representations of deep networks. However, this chapter will focus on the properties of CCA when applied to data fusion of features from deep networks. To use this fusion method with deep networks, Imran *et al.* and Ahmad *et al.* [249, 267] proceed as follows:

1. Train one neural network per sensor (each training occurs separately) to solve the classification problem.
2. Extract the feature matrices from the last layer of the networks, which we call X_1, X_2.
3. Use CCA to compute the aligned representations X_1', X_2'.
4. Use a Machine Learning classifier (*e.g.* SVM) to classify the sum $X_1' + X_2'$

However, the results they obtain with this method are not much better than the other data fusion methods, and we could wonder whether this method is really relevant. The main conclusion is that the fusion method relying on CCA is equivalent to a simple sum of the logits at the end of the network.

Section 6.1 will serve as a detailed introduction to the properties of CCA. Then, we will present an overview of the use of this operation in the Deep Learning literature in section 6.2, before applying CCA to our networks to draw some interesting conclusions on our problem in section 6.3. The rest of the chapter is devoted to a more complex reasoning. To understand it, we have three affirmations:

- **(A)**: The classes are well separated in the feature space.
- **(B)**: The first canonical components are close to a linear combination of the class components.
- **(C)**: The fusion method relying on CCA is equivalent to a simple sum of logits.

If proposition (C) is the one that matters the most to us, we want to understand where it comes from. There are three ways to show C: we can demonstrate it directly using experiments; we can also show that B is true and $B \implies C$, or we show that A is true and $A \implies B$ (knowing that we already have $B \implies C$).

A is fairly easy to demonstrate: given that the classification layer of a network is linear, the classes are well separated if and only if the classification performance is high. Contrary to A, showing proposition B experimentally is not trivial, which is why we will present it first: section 6.4 will prove it experimentally. Once we have a solid set of experiments to know whether B is true, we will show how $A \implies B$ in section 6.5. After this, we will finally come to the proposition that motivates this chapter, the fact that a CCA fusion is approximately the same as a sum of the class logits (C). Section 6.6 will explain why $B \implies C$, while section 6.7 shows C experimentally.

The last section (section 6.8) will try to see what happens when we consider extracting features from earlier layers and conclude that the reasoning this chapter is about is likely to be specific to the the features from the last layer.

6.1 Introduction

This chapter uses many specific notations and definitions, and the present section explains them: section 6.1.1 gives the first definitions, section 6.1.2 introduces the process of feature extraction from deep neural networks, and section 6.1.3 presents the CCA operation itself with some of its properties.

6.1.1 Notations and definitions

variable	set	definition
X_i	$\mathbb{R}^{n \times s}$	features matrix of network i s samples, n dimensions
W_i	$\mathbb{R}^{n_c \times n}$	class weights of network i n_c classes, n input features each row j is the class component associated to the j^{th} class
b_i	$\mathbb{R}^{n_c}$	class bias
y_i	$\mathbb{R}^{n_c}$	class logits, $\forall x \in \mathbb{R}^n, y = W.x + b$
Y_i	$\mathbb{R}^{s \times n_c}$	logits matrix
Σ_{X_i,X_j}	$\mathbb{R}^{n_i \times n_j}$	covariance matrix of the feature matrices X_i and X_j
B_i	$\mathbb{R}^{n_i \times n}$	basis change
P	$\mathbb{R}^{n \times n}$	projection ($P.P = P$)
U, V	$\mathbb{R}^{n \times m}$	orthonormal families of vectors m vectors in dimension n
E	$\mathbb{R}^{n \times n}$	eigenvectors of a $n \times n$ matrix
$I_n^{n_s}$	$\mathbb{R}^{n \times n}$	$\forall n_s \in [1..n], I_n^{n_s} = \begin{bmatrix} I_{n_s \times n_s} & 0_{n_s \times n - n_s} \\ 0_{n - n_s \times n_s} & 0_{n - n_s \times n - n_s} \end{bmatrix}$

Table 6.1: A summary of the notations we will use in this chapter

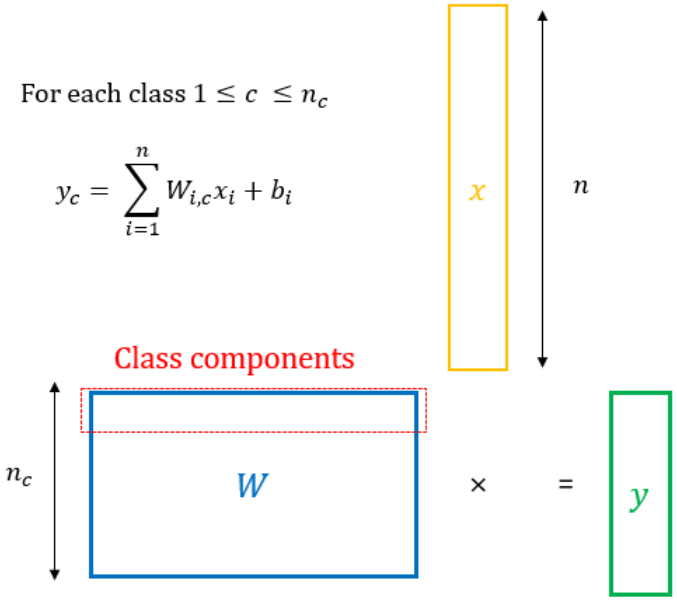


Figure 6.1: The class components are the n_c vectors formed by the lines of the weight matrix W of the last FC layer.

CCA works with samples from a vector space $\mathbb{R}^n$ and involves many concepts from linear algebra. Table 6.1 explains some of the notations we will use later.

A *component* from a feature space is both a vector of this feature space and the linear application that consists in projecting a vector onto the line spanned by that vector. Hence, a series of m components in $\mathbb{R}^n$ can be expressed as a matrix in $\mathbb{R}^{n \times m}$. For instance, in the following sections, we will consider the n_c *class components*, which form the weight matrix $W \in \mathbb{R}^{n_c \times n}$ of the last classification layer (see fig 6.1).

6.1.2 Deep features

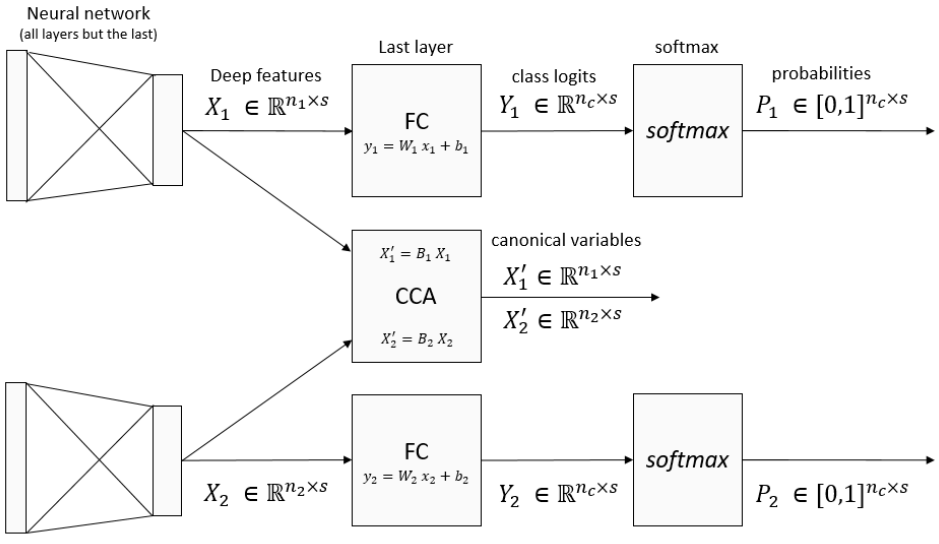


Figure 6.2: The extraction of deep features

To compute deep features from any input, we select the features used by the last layer of the network. These features are classified by a fully connected layer, before the softmax which gives the final probability. The fully-connected layer is linear ($y_1 = W_1.x_1 + b_1$). In other words, for each class c, computing the logit corresponding to this class is equivalent to computing a scalar product with the c^{th} row of W. We call each of these vectors the *class components*. See figure 6.1 for an illustration.

6.1.3 A general presentation of Canonical Correlation Analysis

The Canonical Correlation Analysis (CCA, [266]) aims to find linear combinations of coordinates with maximal correlation between two datasets. Let X_1 and X_2 be two feature matrices with s samples and n_1 and n_2 features, respectively ($X_1 \in \mathbb{R}^{n_1 \times s}$ and $X_2 \in \mathbb{R}^{n_2 \times s}$)[1]. Let us set $n = min(n_1, n_2)$. CCA finds two basis change matrices $B_1, B_2 \in \mathbb{R}^{n \times n_i}$ which produce two new sets of features $X'_i = B_i X_i \in \mathbb{R}^{n \times s}$, such that:

- The covariance matrices of X'_1 and X'_2 are diagonal
- The diagonal coefficients of the correlation matrices of X'_1 and X'_2 are maximal. In other words, the correlation between the i^{th} row of X'_1 and the i^{th} row of X'_2 is maximal.

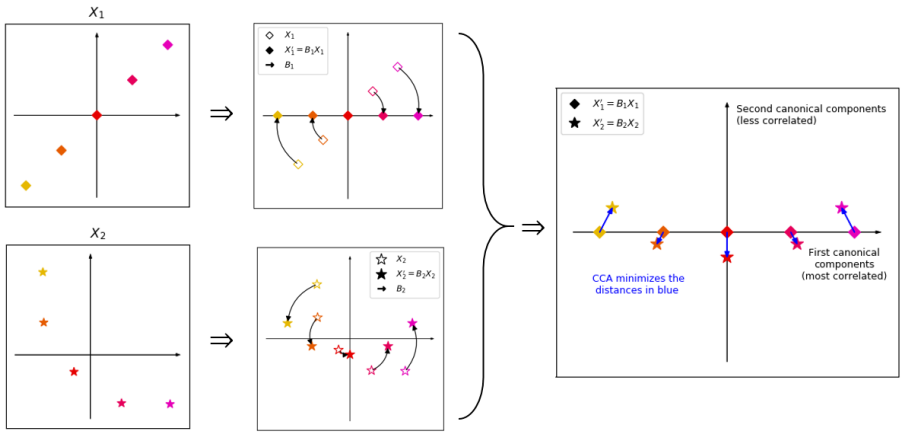


Figure 6.3: An illustration of the CCA operation. X_1 and X_2 are two 2-dimensional feature spaces representing the same five samples (two points sharing the same colour correspond to the same sample represented with different features). Note that if B_1, B_2 are rotations in our example, they can be any base change in the general case.

Note that computing the solution assumes that the covariance matrices of both X_i are invertible so that we can have n decorrelated components in X'_i. In practice, to have 'enough' components, we use PCA to keep 99.99% of the variance and remove zero-variance components, similarly to [271]. This is somewhat similar to SVCCA [269], except that we willingly keep the low variance components, which were considered noisy (and hence, removed) by SVCCA.

The new features X'_i are called the *canonical variables*, while the basis changes B'_i are called *canonical components*. Figure 6.3 illustrates this operation. The components are ordered: the first row of X'_1 and the first row of X'_2 represents the most correlated linear combination of features one can find. The second row of these matrices represents the second most correlated linear combinations, subject to the second row being decorrelated with the first row, and so on. We will focus on the most correlated components that is, the first n_s features in X'_1 for some $n_s \in [1..n]$. The canonical components can be expressed the following way [272]:

[1]The notation X_i is sometimes used to denote the i-th row or column of matrix X. If we except figure 6.1, we will not use this notation in this chapter, and X_1 and X_2 will always denote the first and second feature matrices (with their respective networks and sensors).

$$B_1 = \Sigma_{X_1X_1}^{-1/2} E_1 \tag{6.1}$$

Where E_1 is the set of eigenvectors of the matrix $\Sigma_{X_1X_1}^{-1/2}\Sigma_{X_1X_2}\Sigma_{X_2X_2}^{-1}\Sigma_{X_2X_1}\Sigma_{X_1X_1}^{-1/2}$, ordered by descending eigenvalue. The same goes for B_2.

The canonical variables have an interesting property: they are invariant to invertible linear transformations.

Theorem 1. *Invariance to linear transformations [272]:*
Let $X_1 \in \mathbb{R}^{n_1 \times s}, X_2 \in \mathbb{R}^{n_2 \times s}$ be feature matrices, $M \in \mathbb{R}^{n_1 \times n_1}$ an invertible transformation, $b \in \mathbb{R}^{n_1 \times 1}$, and $\hat{X_1} = M_1X_1 + b_1$. Let us compute CCA a first time on the couple (X_1, X_2) to obtain $X_1' = B_1X_1$, then compute CCA a second time on the couple $(\hat{X_1}, X_2)$ to obtain $\hat{X_1}' = \hat{B_1}.\hat{X_1}$. Then:

- $\hat{X_1}' = X_1'$
- $B_1 = \hat{B_1}M$

It should be noted that only the canonical *variables* (the X_i') are invariant to linear transformations. The canonical *components* (the B_i) are not, because they end up cancelling the transformation applied to X_i. This distinction will prove useful at the very end of this chapter. However, CCA only covers the *linear* relationships between sets of variables: if, for instance, each element of X_2 is the square of the corresponding element in X_1, CCA will not see the relationship between the feature matrices. Some improvements have been suggested in the literature to cover a broader range of relationships between sets of features, the most famous of them being kernel CCA [273]. However, this chapter will focus mostly on classical, linear, Canonical Correlation analysis.

To conclude this section, let us sum up the hypotheses we work with:

- The features are extracted from *classification* networks. In future work, we might consider generalizing to features extracted from unsupervised networks (autoencoders) or self-supervised networks, but we do not think our experiments generalize to handcrafted features.
- We extract features from the *last* layer of a network. One experiment in section 6.8 briefly brushes the subject of features from other layers, but without drawing any complete conclusion.
- We work with only two feature matrices at once, that is, we will not use generalizations of CCA to more than two feature matrices [268].
- We limit our analysis to the baseline, linear CCA, and we do not use recent variations such as PWCCA [274] nor kernel CCA [273].

6.2 Related works

The idea of applying CCA to any two feature matrices existed before the emergence of neural networks [275, 276, 271, 277, 278, 279]. The idea is the same as with deep features: extract the feature matrices X_1, X_2 using handcrafted features, compute the canonical variables X_1', X_2', and use them for the application (classification [276, 277], source separation [271], etc.).

Remark: for classification, there are two ways to use CCA on deep features. When the canonical components are given to the Machine Learning model, the model receives either the sum of components $X_1' + X_2'$ or the concatenation of the two feature matrices $\begin{pmatrix} X_1' & X_2' \end{pmatrix}$. We focus on the sum in our explanations because this is what the researchers did with deep features [249, 267], but the reasoning is the same when the sum is replaced with a concatenation.

With the recent advent of deep learning, it was only a matter of time before researchers tried to use CCA fusion on deep features. The most important field of application is to use CCA to quantify the similarity between representations of deep networks. Multiple publications already studied network similarity to find constants in the behaviour of all deep models. A complete inventory is out of scope for this work, because we mostly pursue the works [280] and [281]. We will just mention that many of these works compared the

predictions of two networks [282], or their correctness [283]. The works we are interested in mostly study the deep features that precede the logit layer.

In 2017, Raghu *et al.* popularized Canonical Correlation Analysis (CCA) in the Deep Learning research community. This operation had several applications in the last few year, the most important of which is to demonstrate that networks learn from the bottom first [269] (the earlier layers stabilize before the layers close to the output). In addition, they published their code, which we reused partly to compute the CCA numerically.

On Linear Identifiability of Learned Representations

Roeder *et al.* [280] studied the representations learnt by unsupervised or self-supervised models. They showed that the representations learnt by two neural networks are equal up to a linear transformation if the networks solve the problem perfectly. By using canonical correlation analysis, they show that the equality up to a linear transformation is close to being achieved in several unsupervised learning problems: they compute the correlation of the canonical variables X'_1 and X'_2 and find that this correlation is relatively close to 1. This means that recomputing the features from one network using the features from the other is feasible: if

$B_1X_1 = X'_1 \approx X'_2 = B_2X_2$,

then $B_2^{-1}B_1X_1$ should be close to X_2.

Their claim is valid for broader tasks than mere supervised classification (they work on tasks like self-supervised learning and word embedding), but we focus on classification because we can compare the canonical components to class components, and/or canonical variables to class logits. However, they provide a mathematical formulation for tasks such as classification, semantic segmentation, word embedding, *etc.* (section 2 of [280]); which could allow to generalize the concept of "class components" beyond classification. This formulation could be used to adapt the reasoning to other problems.

Exploring the Interchangeability of CNN Embedding Spaces

McNeely-White *et al.* [281] generated additional results on ImageNet classification. Mainly, they select a broad diversity of architectures, and compute the accuracy when they replace the features X_2 with an approximation using the features from the first network: instead of measuring the accuracy when the logits use clean data ($Y_2 = W_2X_2+b_2$), they replace the features X_2 with $W_2^+W_1X_1$ (where W_2^+ denotes the Moore-Penrose pseudo-inverse of W_2) and classify these recomputed features using the fully-connected classification layer of network 2 (with weights and biases W_2 and b_2, respectively). They obtain that the loss of accuracy is relatively small (a few points on average), indicating that the reconstructed features are close to the original.

The formula $X_2 \approx W_2^+W_1X_1$ looks extremely similar to the previous one ($X_2 \approx B_2^{-1}B_1X_1$). In fact, if we had, $X_2 = W_2^+W_1X_1$, we would have, $Y_2 = W_2X_2 = W_1X_1 = Y_1$. This would mean that the correlation between the class logits is perfect, which would imply that CCA puts these components first. Obviously, the claim $Y_1 = Y_2$ does not hold in practice, but the correlation between logits happens to be high enough so that the CCA picks up the logits first.

When McNeely-White *et al.* state that the two results are linked, they might not have realized that the fact that the proximity they measured is $X_2 \approx W_2^+W_1X_1$ is the main reason why the canonical variables X'_1, X'_2 are close to each other in the experiments from Roeder *et al.* This is a direction we will pursue in section 6.4.3 after we present how CCA can help us make interesting conclusions on Transport Mode Detection.

6.3 An application of canonical correlation analysis to deep features

This section will illustrate how we can use CCA to understand better the features from the neural network. We will use the CCA to show that the accelerometer and gyrometer are more similar to each other than the accelerometer and magnetometer (a fact that was not demonstrated until now) in section 6.3.1. We will also demonstrate that the architecture of the network allows the information about the power of the input signal to appear in the network's features in section 6.3.2, and, finally, we will see that the initialization of a network does not impact the features after the training (section 6.3.3).

6.3.1 Measuring the canonical correlations to quantify the similarity between sensors

In chapter 2, we said that the accelerometer and gyrometer were close because they both encode the dynamics of the phone. We also added that the accelerometer was quite different from the norm of the magnetometer. Canonical Correlation Analysis gives us a way to quantify this difference. We consider two networks and compare the correlations of the couples of canonical variables.

The most extreme case is when the two feature matrices are equal up to a linear relationship. In this case, the canonical variables are exactly equal, and their correlation coefficient is equal to 1. In general, the closer the correlation is to 1, the closer the features are to each other.

However, when considering two independent multidimensional distributions, computation of CCA will often yield a positive (nonzero) correlation, because the number of samples is finite: there is always a way to find a direction with at least *some* correlation between the feature matrices. To take into account the finiteness of data, we compare correlation between features to the correlation between two random (independently-drawn) normal feature matrices: after we computed the PCA (after we kept 99.99 % of the variance), we generate a couple of feature matrices filled with independent, normal random variables (zero mean, unit variance). If we had an infinite number of samples, it would be impossible to find any correlated components within the feature matrices, and the correlation between canonical variables would be zero. But as the number of samples is finite, it is possible to find canonical components even if feature matrices were generated independently. We will compare the correlation obtained with these simulated curves to the correlation obtained with deep features in order to quantify the correlation which is simply due to the finiteness of the number of samples.

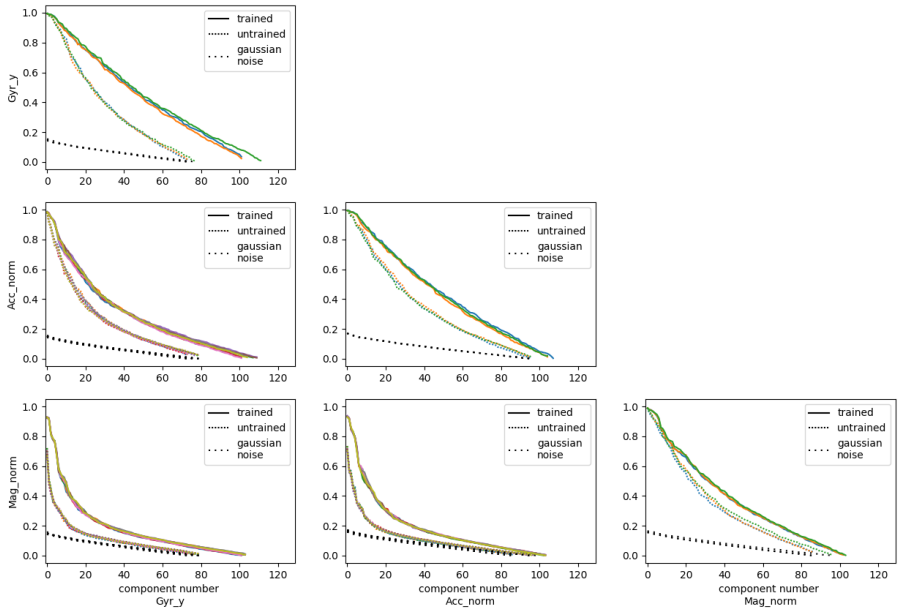


Figure 6.4: Each figure displays the correlation coefficient between each of the couple of canonical variables, generated with the SHL dataset. Two curves (whether dotted or plain) sharing the same colour were computed using the same pair of networks, before (dotted) and after the training (plain). The different colors represent different couples of networks (we do not compute an average because the number of components we kept to get to a full-rank feature matrix changes between initializations).

Figure 6.4 displays the results.In this figure, we represent the average correlation between features from *trained* networks (networks after the training process), but also between *untrained* networks, that is, between two networks that have just been initialized (we use the Glorot initialization [284]). To emphasize, as the weights of an untrained network were not modified by any training procedure, the untrained networks' predictions are entirely random.

As we can see in fig. 6.4, the correlations after training are higher than before training, which means the two networks' features got closer during the training process, as the dashed curves are consistently lower than the plain curves. In addition, the fact that the correlations from the last components obtained using untrained networks (coloured dotted curve) are similar to the last correlations of random Gaussian variables (black dotted curves) indicates that there is only a little difference between the latest components and two independent Gaussian variables.

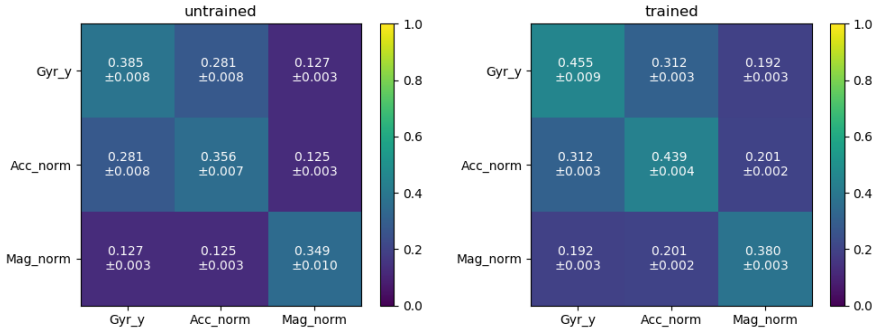


Figure 6.5: Each figure displays the average correlation coefficient between each of the couple of canonical variables, generated with the SHL dataset.

These curves are interesting, but they are quite hard to compare to each other. To be able to compare sensors, we simply compute the average of all the values in every curve and display the averages in fig. 6.5. We made the distinction between trained (left) and untrained (right) cases. In particular, we can see that the average correlation between the accelerometer and magnetometer (0.245) is lower than the correlation between the accelerometer and gyrometer canonical variables (0.335). As a side note, we should mention the fact that the correlation of the *untrained* accelerometer and magnetometer (0.194) is lower than the untrained accelerometer and gyrometer (0.315). As the weight of an untrained network does not depend on the sensor the network will use, the only reason for this mismatch in correlations to exist is that these correlations translate similarities with the input data. These remarks finally confirm our intuition: the accelerometer is closer to the gyrometer than it is close to the magnetometer.

6.3.2 Demonstrate that the network keeps the power

The previous section measured the similarity of features but did little to understand what does this similarity rely on. In particular, we could see that the features from two untrained networks still kept some correlated components, despite being processed by completely random weights. In this section, we will illustrate one property of the input signal we can find in the network's features: the input power can be found in the networks' features. From a statistical point of view, the power of a signal is equal to its standard deviation if its mean is zero (which we assume in the following): $P = \frac{1}{T}\sum_{t=1}^{T} X_t^2$. To show the input power can be found in the network's features, we simply compute the standard deviation from the raw, temporal measurements of each sample and measure the correlation with every feature from the neural network. We also display the correlation between the feature matrices and the power of the original, one-dimensional time signal (seen as a feature matrix with one column). The plain curves in fig. 6.6 denote the set of correlations between features of the canonical basis and the power of the signal. As we can see, most of the values are quite far from +1 or -1, which mean that the power is not present in *all* the features of the network. However, when computing the component which is the most correlated to the power with CCA, we find that this component is quite close to one, which means that a linear combination of features is often enough to reconstruct the power.

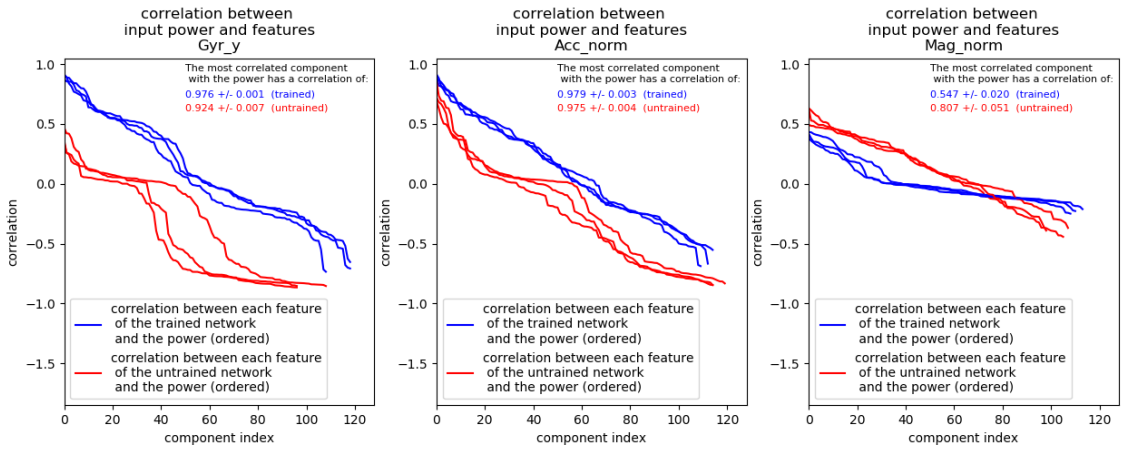


Figure 6.6: the correlation between each of the features and the power of the input signal

To understand how this data is available in the features, let us list the operations that occur between the input and the features:

- Convolution (convolution layers)
- Adding a bias (convolution and fully-connected layers)
- ReLU nonlinearity (convolution and fully-connected layers)
- MaxPooling
- Multiplication by a fixed matrix (fully-connected layers)
- Dropout

In particular, all but one of them (the addition of a bias) have he following property:
$f(\alpha x) = \alpha f(x), \forall \alpha > 0$

Despite looking similar to the linearity, this property does not mean the network is linear. However, its validity for most of the network operations is interesting. Let us forget (temporarily) that the addition of a bias is an operation in our network, and assume that all the operations verify the property. As this property is stable by composition (if f and g verify it, $f \circ g$ will verify it), the whole network does verify it. This means that if we multiply all the values of one signal by a positive constant α, the features of this signal will all be multiplied by this same value. Similarly, the signal's power would also be multiplied by α. In other words, the amplitude of the input signal influences both the power and the features produced by the network. What this means is that the signals' power is correlated to the amplitude of the network's features. The presence of bias addition does disturb the reasoning, but the disturbance is not strong enough for the power to be completely erased from the feature embeddings. For instance, figure 6.6 shows that there exist an adequate linear combination of features that creates a value that is well correlated with the power.

There is one important point to make: the explanation we just gave is true whether the network is trained or not. This means that the power of the signal is available even before the network started training. Figure 6.6 confirms it experimentally: there exist a linear combination of the 128 features that allows computing the power with reasonable precision from features of *untrained* networks. Note that this is true only because the operations we used in our network let the information about to the signal's power go through them, and we might reach a different conclusion with other properties (fundamental frequency, entropy, *etc.*).

As the power of the signal is available from the beginning of the learning process, chances are that the network uses this easily accessible information as soon as the training starts, especially given how relevant it is to the problem of transport mode detection. It might be interesting to know if this stands for other deep learning problems. However, one must be aware of the fact that not all recent "building blocks" of neural networks obey this rule: attention (whether classical attention or multi-head attention), or squeeze-and-excitation blocks [8] are notable counter-examples.

For our problem, the power of an input signal is an extremely important feature. The fact that it is available from the start may explain why the networks reach a good performance level with relatively little epochs (for instance, a network using the norm of the accelerometer reaches a F1 score of 80% in the first five epochs, before eventually reaching values of about 89%). However, with different networks (such as the transformers relying on attention), the power might not be available from the start. The network may still be able to learn to recompute the power eventually, or any other feature that is useful for classification, but the convergence may be slower to reach. In other words, we think the simplicity of our network did help it to reach a good performance efficiently.

6.3.3 Influence of the network initialization

In the literature, multiple lines of work can attest that the seed used for the random initialization of network weights has little influence on the final performance (we are not talking about the random law the weights are initialize from, but about the difference between two realizations of the same law). The fact that many of our own experiments have standard deviations of about one percentage point (with the noticeable exception of the networks using the orientation vector in section 3.4, or in appendix D) confirms this fact. To evaluate the effect of network training, we could try to look at fig. 6.4 where the colour of the curves denotes the networks from which the features come. For instance, the dotted and plain blue curves in the bottom-left hand corner of the figure denote the correlations between features from the same couple of networks using the norm of the magnetometer, before and after training (respectively). We could try to see if the colours for which the dotted curve is higher are also the colours for which the plain curves are higher (*i.e.*, if the couples of networks that produce correlated features before the training still produce correlated features after it), but the difference between the curves is not high enough for us to distinguish anything. To see things better, we keep on measuring the feature obtained by different networks before and after they are trained, but we colourize things differently. We realize four comparisons, depending on the networks the features are extracted from:

- Two trained networks coming from different initializations (blue curve)
- The features from the same network, before and after training (green curve)
- The features from one untrained network and a different trained network (red curve)
- Two untrained networks (black curve)

For each comparison, we extract the features, compute the canonical variables X'_1, X'_2, and measure the component-wise correlation coefficient. We obtain a decreasing curve because the canonical variables are ordered. For three SHL sensors (accelerometer, magnetometer, gyrometer), we repeat the experiment for three random network training processes (which means there are three couples of networks to compare for the green and red curves). Figure 6.7 displays the resulting correlation curve. The similarity (measured with CCA) between any trained network and its untrained version is the same as the similarity between a trained network and any untrained network. In other words, we bring one additional experiment supporting the fact that the initialization of the weights has little influence on the network post-training.

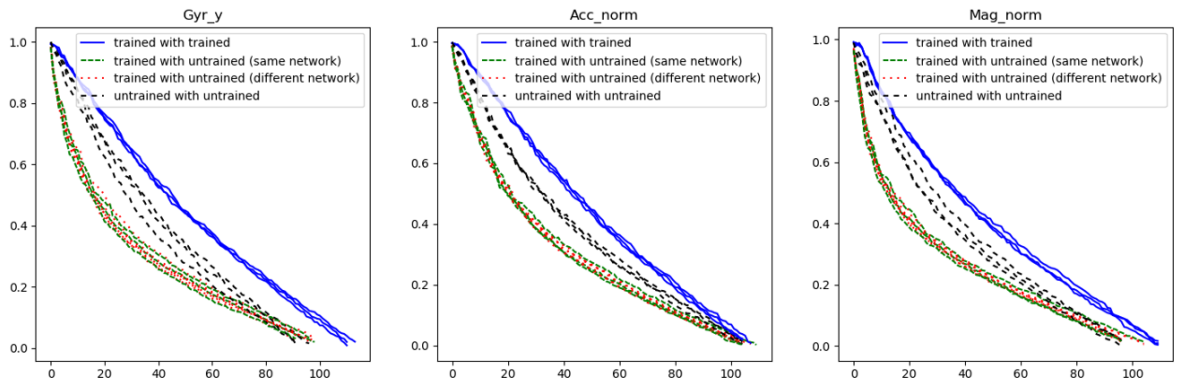


Figure 6.7: The correlation between different versions of trained and untrained networks, with SHL data.

In this section (6.3), we used CCA to prove three things:

- the accelerometer is closer to the gyrometer than the magnetometer
- the features of the network allow it to see the input signal's power, even before the training
- the features of a network do not depend on its initialization

These affirmations represent examples of conclusions we can obtain by applying CCA to deep features. However, we were originally interested in CCA to perform the fusion between features from different sensors. The application of CCA is developed in the next section, to demonstrate that this fusion method is irrelevant.

6.4 The equality between the first canonical components and the class components

This section aims to show that the class components can be found with a linear combination of the first canonical components (proposition B of our introduction, see fig. 6.8 for an illustration). It is fairly long, and organized on thee major parts: subsection 6.4.1 displays a first plot of the first couples of canonical variables, which allows us to get an insight of what proposition B actually means. Subsection 6.4.2 displays some experiments we call *projection experiments*, that can be found in the literature and used to demonstrate B. Finally, subsection 6.4.3 will serve both to introduce better experiments to demonstrate B, and use it to actually establish the proposition.

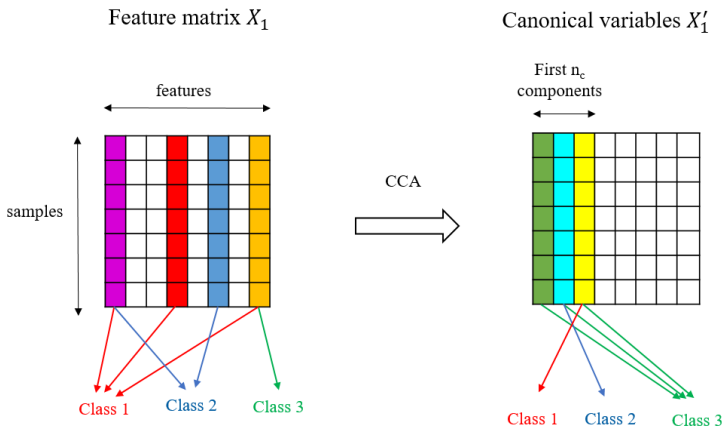
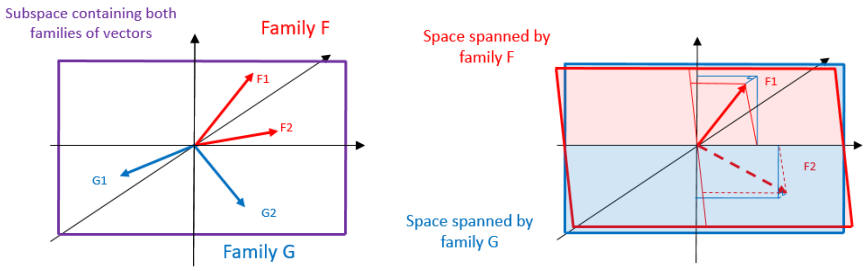


Figure 6.8: The principle of the equality between class components and the first canonical components on a three-class problem. The colours in the different feature matrices denote the different information about the three classes. The feature vectors will undergo a matrix multiplication (denoted by the arrows under the left matrix); and the rows of the matrix the features are multiplied by are the class components. Similarly, the arrows under the matrix of the canonical variables represent matrix M in equation 6.2.

Proposition B is "The first canonical components are close to a linear combination of the class components", and it is equivalent to:

$$\exists M \in \mathbb{R}^{n_c \times n}, W_1 = M I_n^{n_c} B_1 \tag{6.2}$$

In equation 6.2, matrix M denotes the linear transformation that are referred to in the words "up to a linear transformation". In our experiments, we are not interested in computing matrix M explicitly, but we want to know how close are the first two terms of the equation. Figure 6.9 illustrates how we will measure the similarities of components up to a linear transformation: we compare the subspaces spanned by the two families of components.



(a) (b)

Figure 6.9: An illustration of the problem of the comparison of two families of vectors (components), F and G. (6.9a) shows we cannot compare the elements one-to-one, and (6.9b) shows the need for a robust measure (for instance, the dimension of the subspace spanned by family $F \bigcup G$ would not be a good measure).

As we mentioned earlier, the experiments we describe in this section are purely empirical, *i.e.*, we do not give a theoretical guarantee that the linear layer keeps all correlated components. However, even if our experiments are empirical, we will draw conclusions that might apply to any classification network. In order to show how general our claims are, we will use the CIFAR-10 Dataset, a Computer Vision dataset where the network has to recognize the object among 10 possible classes in 60,000 low-resolution RGB images (with size $32 \times 32 \times 3$). We used the code from [285], which implemented the same hyperparameters and architecture as ResNet-50 [150].

6.4.1 Introduction: A glance at canonical variables

To illustrate our intuition, we start by plotting the values of the 16 most correlated components between each couple of features. For each component i, we plot the i^{th} component of X_1' versus the i^{th} component of X_2'. We repeat the process for CIFAR (figure 6.10), and for SHL (figures 6.11, 6.12, and 6.13). Firstly, we can notice that the first components from each sensor are more correlated than the later components, as expected. Secondly, the correlations are higher in the case of features coming from two identical sensors, and that the correlations between accelerometer and gyrometer are higher than the correlations between accelerometer and magnetometer: this comes directly from the results we showed in section 6.3.1.

Finally, for the problem that matters to us, it looks like we can easily separate several classes from the others with these variables, which is an indication in favour of the proximity between canonical and class components. However, if these graphs give clues indicating that the first canonical variables are equal to the logits, they do not prove anything: we will present the proof in the next sections.

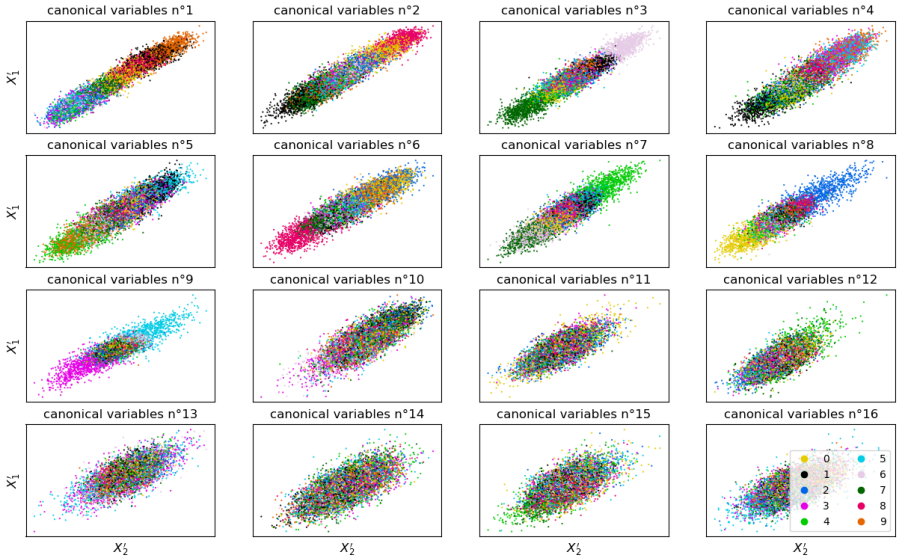


Figure 6.10: The first 16 pairs of canonical variables between two initializations of Resnet-50 models trained on CIFAR-10. The colour indicates the class.

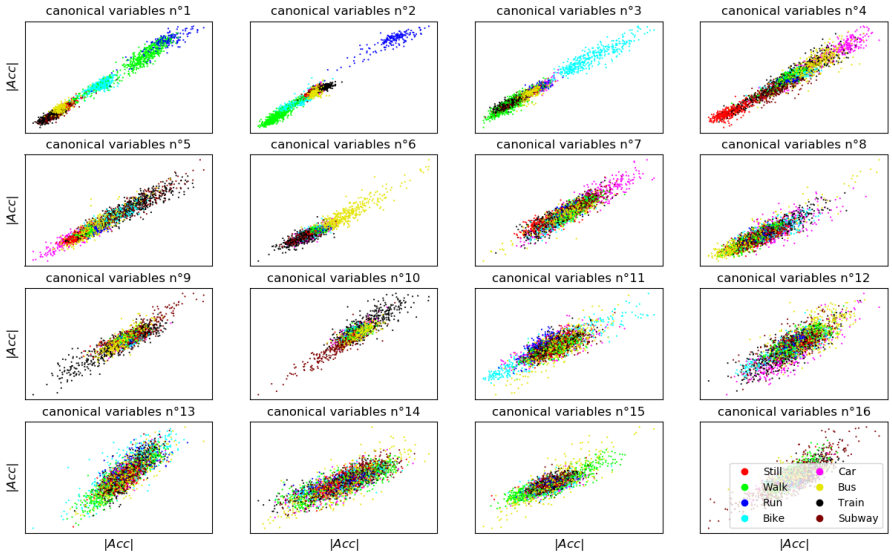


Figure 6.11: The first 16 pairs of canonical variables between two initializations of the accelerometer model. The colour indicates the class.

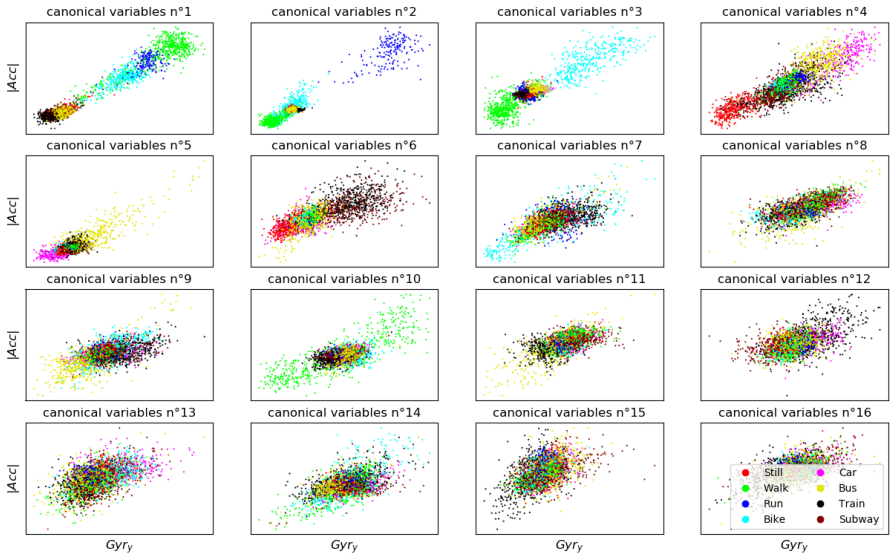


Figure 6.12: The first 16 pairs of canonical variables between an accelerometer and a gyrometer models. The colour indicates the class.

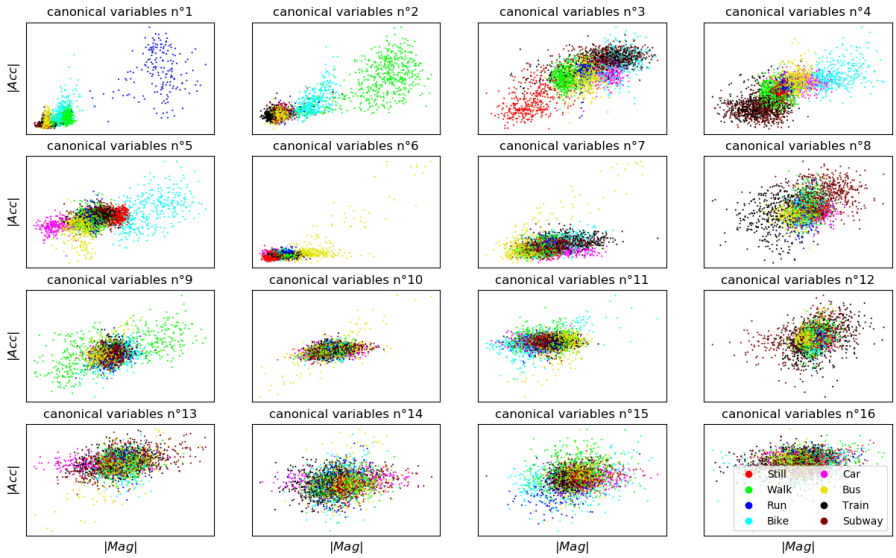


Figure 6.13: The first 16 pairs of canonical variables between an accelerometer and a magnetometer models. The colour indicates the class.

6.4.2 Projection experiments

A more rigorous (but not perfect) experiment is to project the feature matrices on the most correlated components to see if this affects the classification performance. We will reproduce and extend the experiments from [269] (Figure 2 from this work). We start from two trained networks, we extract the hidden features from the last layer of the first one, then we project on a subspace of inferior dimension n_s, before re-injecting the features in the network to measure the performance.

The rationale we adopt is that if the performance is intact, it means that the n_c class components are unaffected by the projection. In other words, it means that the n_c class components belong in the subspace spanned by the n_s most correlated components. Fig.6.14 illustrates this experiment in the case of the projection on the n_s most correlated components. Note that when we use all features, we project on the original space, *i.e.*, we leave the data unchanged. The difference between the end of the curves (performance on pristine data) and the rest (altered data) will indicate the proximity between the considered subspace and class components. We will see in section 6.4.3 and later that this rationale is imperfect in the case of the CCA experiments, but we nevertheless present it for three practical reasons:

- The experiments are going to be relevant later on (section 6.8).
- It is always useful to make sure results from the literature are reproducible (here, from [269], fig. 2).
- These experiments seem to contradict with what we will say in later sections (section 6.4.3), and one needs to understand fully an idea before attempting to criticize it.

In addition, the flaws of these experiments are theoretical, *i.e.,* we did not observe a difference in conclusion between the present projection experiments and the more rigorous alternatives we will present later on. This is why we will anticipate slightly and accept the conclusions of the projection experiments: we will demonstrate in section 6.4.3 that the conclusions are true.

The dimension of the subspace and the way of choosing the subspace will vary. To choose the components, we will use seven methods (three of them coming from [269]):

1. *The most correlated components found with CCA* (`CCA_highest`), as in [269]. This subspace should be equal to the subspace spanned by the class components when $n_s = n_c$.
2. *The least correlated components* (`CCA_lowest`): if the most correlated components are the class components, the components with the lowest correlation should not include any relevant information for the problem.

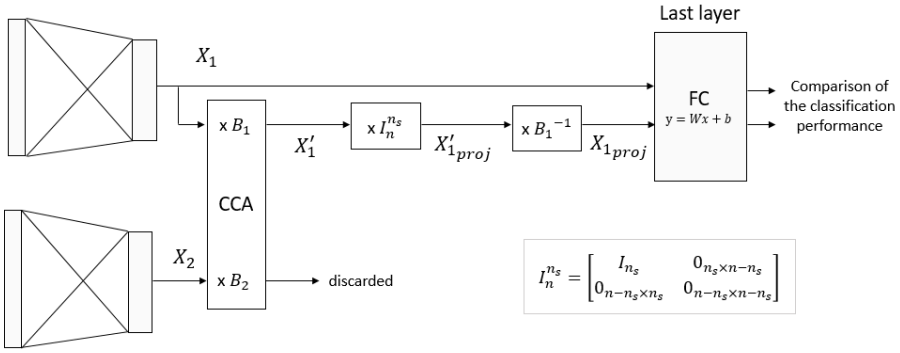


Figure 6.14: The principle of the subspace projection experiment, illustrated with the projection on the most correlated components (red curve in fig. 6.15): $P_1 = B_1^{-1}.I_n^{n_s}.B_1$ projects X_1 onto a linear space with dimension n_s. The value of n_s is a parameter we will modify in our experiments.

3. *CCA with random components* (`CCA_random`): one may argue that the CCA curve is above the others in [269] because CCA allows creating decorrelated components, which would mean that its components are less redundant than random directions. If this was the case, selecting random CCA components would be better than selecting components with a random projection.

4. *PCA*: Kamoi *et al.* [286] showed that the n_c components with the most variance are the components that will be used for classification. We project the features on the components with the most variance to validate their findings.

5. *A set of n_s features chosen randomly* (`random_keep`), as in [269], for comparisons.

6. *A random orthogonal projection of features* (`random_projection`). Comparing the random selection of n components versus the projection on n components shows that the standard basis of $\mathbb{R}^n$ (the set of $x_1 = [1\ 0\ 0\ ...], x_2 = [0\ 1\ 0\ ...], ...$)[2] does not play a particular role (*i.e.* selecting the values of n_s features is not particularly meaningful).

7. *The n_s features with the highest activation in absolute value* (`max_activation`), as in [269]. These features are sometimes said to be more efficient at capturing the information than random features [269].

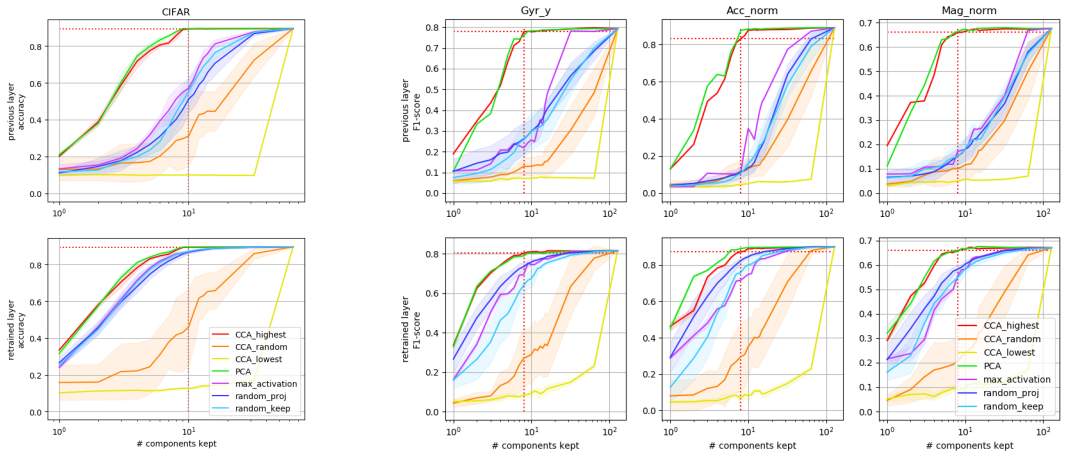
To save time, we do not consider all the possible number of components: because we want a high resolution around n_c, we only considered the $2 * n_c$ first components (where n_c is the number of classes, 8 for SHL and 10 for CIFAR), and, after that, the number of components which are powers of 2 $(16, 32, ...)$, up to the maximal number of components (128 for SHL, 64 for CIFAR)

In addition to this, after measuring the performance of the layer when using projected features, we also try retraining the classification layer on a projected version of the validation set, with the same hyperparameters as the initial training of the network. The goal of this retraining is to illustrate the difference between the components a network actually uses for classification and the components that carry a piece of information the problem. If the performance of the retrained layer is low, this means we can be sure that the projection removed *all* useful information. If only the performance of the original layer is low, this only means that we got rid of the information that was used by the network.

Note that the CCA operation requires two databases. In fact, when we use CCA, we use a second matrix of features X_2, but only to compute X'_1 (we discard X'_2). In the first experiment (fig. 6.15) this second network is another initialization of a network working with the same sensor, while the second experiment (fig. 6.16) shows results obtained with two networks using different sensors.

[2] this basis is most often called *canonical basis*, a name we will not use for obvious reasons.

Similarity between identical sensors



(a) (b)

Figure 6.15: The performance of the networks after projecting their features on subspaces with varying dimensions, on the CIFAR (a) and SHL (b) validation sets. The top row indicates the validation performance of the network as-is, while the bottom row indicates the performance when retraining the classification layer on a projected training set. For each curve, the experiment was repeated 5 times, and the standard deviation is given by the width of the curve (which is sometimes too small to see). The dotted line highlights the performance with the n_c most correlated components. Best view in color.

Figure 6.15 displays what happens when we project the features from different subspaces. Here, we compute the canonical components (red, yellow, and orange curves) using features from a different initialization of the same sensor. We can draw several conclusions from it:

- The performance of the projection on the n_c highest variance components (`pca`, green curve): this verifies the findings of Kamoi *et al.* [286], the n_c components with the highest variance are the class components.
- Similarly to Figure 2 from [269], the most correlated components (`cca_highest`, red curve) are more useful for the classification problem than a random choice of components from the standard basis.
- The red curve (performance of the components with the highest correlation) is almost at its maximum for n_c components even before retraining, there is almost nothing to gain after n_c components. This validates the claim from [281]: these components correspond to the subspace used by the classification layer.
- The yellow curve (the components with lowest correlations), is under all the others. Before retraining, the performance of a projection on the n_s components with the lowest correlation is minimal, even when we select half the components. After retraining, the performance of these components is still well under the performance of random components: selecting the least correlated components effectively removed most of the classification information.
- The orange curve (random CCA components) is lower than the random choice of features (blue curves). This means that the performance of the components with the highest correlation is not due to an encoding of the information which is 'more compact' than random directions. If one of the methods were redundant, it would mean that adding directions would not change: we would need to add more dimensions to effectively cover the same space, and the curve of the redundant representation would appear

to the right of the curve of an efficient encoding. In our case, the curve of random CCA components appears to the right (or to the bottom, as the curves increase), which means that reason why the red curve is higher than the random directions is not due to the fact that any two CCA components are non-redundant, but to the fact that the *first* components are particular. Additionally, the standard deviation of this curve is unusually high: as the CCA operation isolates the class components from the rest, selecting some of its components at random creates extremes situations: either a classification component is selected, or it is not. The standard deviations of the other random methods are not as high because the random choices allow spanning partly the class components.

- Before retraining, the two blue curves are equivalent, this indicates that the standard basis of $\mathbb{R}^n$ do not play any specific role in regards to classification.

Similarity between two different sensors

We now lead experiments to verify the most important affirmation: the fact that the most correlated components are equal to the class components, even when the correlation is computed across sensors. This time, when computing CCA, we use features from a network using different sensors. In this section, we do not include any of the other dimensionality reduction methods (PCA, random projection and selection of components, maximal activation components) because those methods work with only one database: the results would be the same as the curves presented in fig. 6.15.

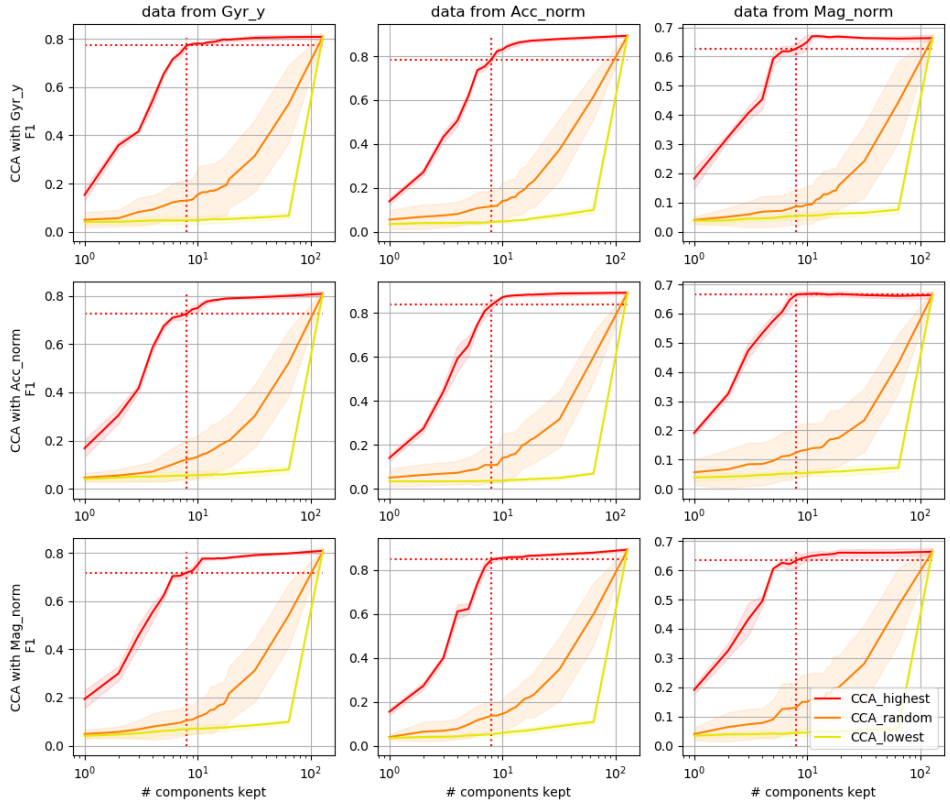


Figure 6.16: The classification performance of classification layer using features projected on a subspace with varying dimension, when the CCA is computed thanks to data from another sensor. As in fig. 6.15, one can see that the performance with the n_c most correlated components is close to the performance with all components. The graphs in the diagonal were generated using the same protocol as the first row of graphs in fig. 6.15b. The dotted line highlights the performance with the n_c most correlated components

Figure 6.16 shows that the performance is maximal when the number of components is slightly higher

than 10. However, contrary to fig. 6.15b, the performance is off by a few points when the number of selected components is equal to 8, the number of classes. This means that the equality between most correlated components and classification vectors is less strong than in the previous case when the CCA was computed from the same sensor. Still, the performance with only 8 components is high enough for us to conclude that the components computed with CCA overlap significantly with the class components.

Could these experiments be useful in Architecture Compression ?

The action of projecting the intermediate features on a well-chosen subspace has already been explored to reduce the size of a (trained) network while keeping the performances intact (this is called *network pruning*, or sometimes *architecture compression*). For instance, one can choose the subspaces thanks to PCA [287], or maximum activation [288]. Depending on the case, the network may or may not be retrained after the projection. Using CCA is another way to choose which subspaces will be removed. Fig. 6.15 shows that reducing the dimension thanks to CCA (red line) is as efficient as using the PCA (green line). However, CCA is less convenient to use, as it requires training two networks instead of one. As a consequence, we think that using PCA is better suited than CCA to prune a single-sensor network, which means that CCA is not at the same level as the state of the art in this domain. However, one may think of using the CCA to prune two networks using *different* sensors: the CCA might select more relevant information than a PC on each of the two networks separately.

In any way, the experiments we led in the present section may only be used to demonstrate the claim we are interested in: the class components are close to the first canonical components, up to a linear transformation (B).

6.4.3 An explicit measurement of subspace similarity

Why the projection experiments are not enough

The projections experiments seemed persuasive enough, but they had two major drawbacks:

The first one is that these experiments relied on classification performance. They looked at the classification performance of a projected version of the features and concluded that the components the features were projected on were equal to the class components if the classification accuracy is the same as the accuracy of the original features. Yet, to classify a sample, the network simply takes the highest logit among all the class logits. For example, if a projection divides all logits by two (see fig. 6.17), the classification performance will be untouched, even though the image of the projection is quite far from the classification subspace. Granted, in practice, the probability for a vector space to distort all logits equally while being much different from the space spanned by class components is small, but this probability is not zero. The second flaw is that even though we used the training samples to compute the components and we projected the validation features on these components to measure the performance, the SHL dataset is only composed of one user. The experiments risk over-fitting to this very user, and one might have not obtained similar results if the validation samples came from another distribution. In short, the previous experiments indicate a relationship between the n_s most correlated components and classification performance on the SHL validation dataset, but do not prove that these components are equivalent to class components all the time.

This is why we will try to use an experiment that is without blind spots. Instead of comparing the class logits Y_1 and canonical variables X'_1, we will compare the class components W_1 to the canonical components B_1. In other words, instead of comparing the effect each of these transformations has on the data, we will compare the transformations themselves.

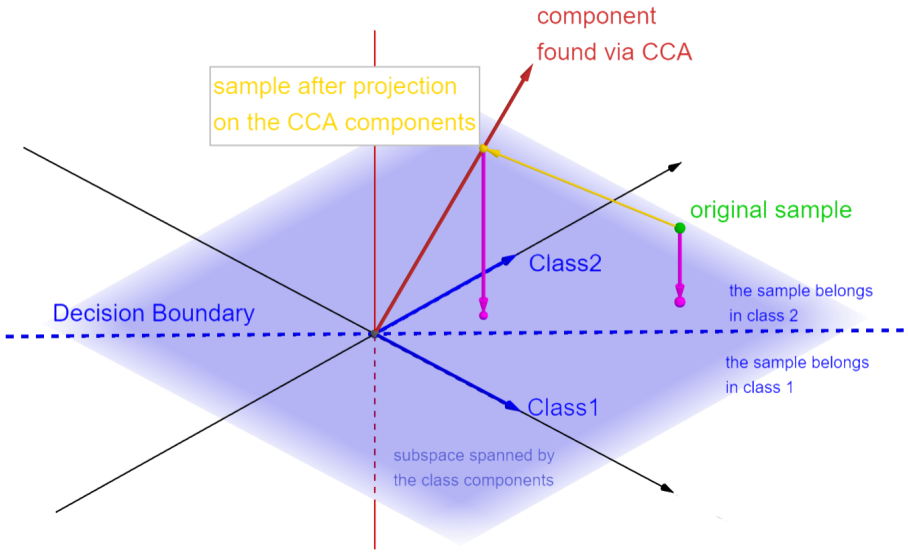


Figure 6.17: Why a simple classification measurement is not enough. In this example, the projection on the canonical components (yellow arrow) does not change the prediction of the network for the green sample, for both predictions (pink arrows) land on the same side of the decision boundary. In this example, the logits (the coordinates of the projections of the points on the blue plane) are changed by the projection on the first canonical components. Despite the canonical components (red) being quite far from the class components (blue plane), the classification accuracy is unchanged.

Why subspace distances do not work as-is with canonical components

If one were to measure the distance between the canonical components and the class components, one would see that the subspaces seem quite far from each other. As we will show later on, at a first glance, the class components seem to be closer to a random subspace than to the canonical components. This result directly contradicts the projection experiments (along with the findings from [280]).

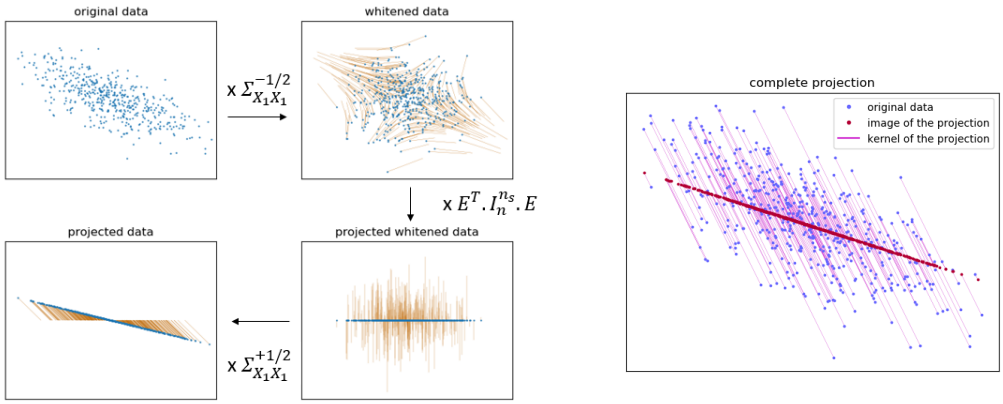
To understand what goes wrong in such reasoning, one must look at what happens when we project a vector on CCA components. If B_1 are the canonical components, the projection on the first n_s canonical components can be expressed by $x \to B_1^{-1} I_n^{n_s} B_1 x$. The image of the projection (that is, the set of all $y \in \mathbb{R}^n$ such that $\exists x, y = B_1^{-1} I_n^{n_s} B_1 x$) is equal to the vector space spanned by the first n_s canonical components, while the kernel of the projection (the set of all x such that $B_1^{-1} I_n^{n_s} B_1 x = 0$) is equal to the vector space spanned by the *last* $n - n_s$ canonical components.

In section 6.1, eq. 6.1, we said that the canonical components can be expressed by: $B_i = E_i \Sigma_{X_i X_i}^{-1/2}$, where the $(E_i)_{i=1,2}$ are eigenvectors of matrices we will not present here. We only need to know the eigenvectors are orthonormal: $E_i.E_i^\top = I$. This means the projection of a vector on the canonical components is not orthogonal: as $B_1 = E_1 \Sigma_{X_1 X_1}^{-1/2}$, the projection $X \to P_1.X$ (where $P_1 = B_1^{-1}.I_n^{n_s}.B_1$) is a projection ($P_1 = P_1.P_1$), but it is not orthogonal ($P_1.P_1^\top \neq I$). As Fig. 6.18 and 6.19 illustrate, this means that the kernel of the projection is not orthogonal to its image.

Yet, all the distances between subspaces rely on an implicit assumption: for every subspace U, the farthest subspace from U we can find is its *orthogonal complement*. The orthogonal complement of U, which we note $U^\perp$, is a vector space of dimension $n - m_U$ which contains all the points in $\mathbb{R}^n$ that are orthogonal to vectors of U. What matters to us is that the distances we use verify: $max_V d(U, V) = d(U, U^\perp)$. But with the projection on canonical components, this is not the case: when we think about a projection, we would expect the farthest point to the image of the projection to be its kernel. In the case of an orthogonal projection, this consideration would not be a problem: the kernel of the projection is the orthogonal complement of its image. But when considering the canonical components, the kernel and the image are no longer orthogonal to each other, which is why the distances we used are not relevant. To summarize grossly, it is almost as if keeping the first n_c components was not equivalent to removing the last $n - n_c$ components.

There are two ways to show the phenomenon is strong enough to take place numerically. The first one is simply to compute the distance to the kernel of the projection instead of its image. In other words, we measure the distance between the *last* $n - n_c$ components of B_1 and the n_c components of W_1. Figure 6.26 will show that this measure indicates that the kernel of the projection P_1 is much farther to the subspace spanned by the vectors of W_1 than any random vector space.

The second method is consists of simply solving the problem, *i.e.*, to get back to equivalent orthogonal components. One could try straightforwardly to design a distance that takes this anisotropy into account in the expression of the distances (similarly to a Mahalanobis distance), but a more simple approach is to make sure that the basis change matrices B_i are orthogonal. To do so, we simply whiten the data: before computing CCA, we replace X_1 by $\hat{X_1} = \Sigma_{X_1X_1}^{-1/2} X$. Whitening has been used in the Machine Learning community when using features that have significantly different variances, but we use it here on features from deep neural networks. This way, the covariance matrix of $\hat{X_1}$ is the identity, and $\hat{B_1} = B_1 \Sigma_{\hat{X_1},\hat{X_1}}^{-1/2} = E$ is orthogonal. Now, comparing the whitened data to the class components make no sense: the class components were trained to process the original data. This is why we compare the whitened data to a modified version of the class components: we replace W_1 with $\hat{W_1} = W_1 \Sigma_{X_1,X_1}^{1/2}$. In our experiments, we made sure that this change does not alter the predictions numerically.



(a) (b)

Figure 6.18: An illustration of the projection on CCA components with toy data. Figure 6.18a illustrates three steps equivalent to the projection on the n_s canonical components (whitening, orthogonal projection, and inverse whitening), while fig. 6.18b illustrates how the kernel is not orthogonal to the image of the projection

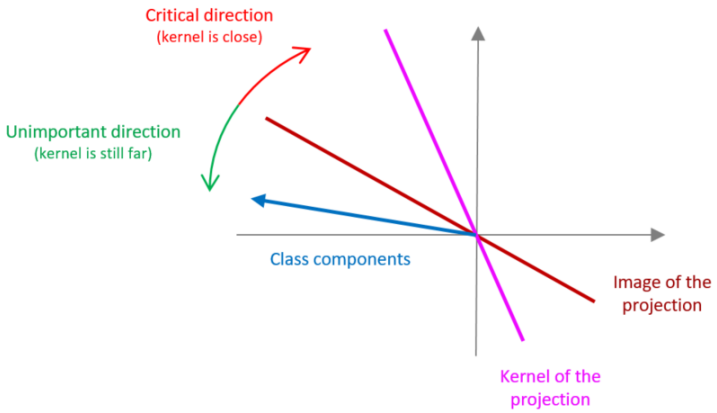


Figure 6.19: Why the class components are relatively unaffected by the projection on the first canonical components, despite appearing quite far from its image: the components are closer to the image than to the kernel of the projection

To sum up, as $B_1 = \Sigma_{X_1X_1}^{-1/2} E_1$, the projection on the first CCA components ($x \to B_i^{-1} I_n^{n_s} B_i x$) can be thought of as a sequence of three operations (fig. 6.18a):

1. whitening ($x \to \Sigma_{X_iX_i}^{-1/2} x$)
2. *orthogonal* projection on the n_s first eigenvalues of E_i ($x \to E_i^\top I_n^{n_s} E_i x$)
3. inverse whitening ($x \to \Sigma_{X_iX_i}^{1/2} x$)

By whitening the data, we end up cancelling the first and third operation, such that the complete projection $x \to \hat{B}_i^\top \hat{B}_i x$ is orthogonal. One might argue that computing the canonical components from whitened data is not equivalent to computing the canonical components from the original features. To be true, the first (original) canonical components and the class components are far from each other if "far" is defined using an isotropic measure (such that the farthest space is the orthogonal complement). With a distance that takes into account the anisotropy of the projection (fig. 6.19), the first canonical components are close to the class components.

Another way to understand the whitening operation is to say that replacing X_1 by $\hat{X}_1 = \Sigma_{X_1X_1}^{-1/2} X$ and W_1 by $\hat{W}_1 = W_1 \Sigma_{X_1,X_1}^{1/2}$ is equivalent to distorting the feature space so that the kernel of the projection on CCA components is orthogonal to the image of the projection (the components themselves). This is the same principle as saying the Mahalanobis distance ($\sqrt{(x-y)\Sigma^{-1}(x-y)^\top}$) distorts the feature space so that the covariance of the data is spherical.

Experiments in the remainder of the section demonstrate that CCA on whitened data creates components that are close to the modified class components.

Experimental protocol

The experiments in the remainder of the section 6.4.3 have three objectives:

- Establish that the distance between the first canonical components and the class components is high if the canonical components are computed as-is (which we mentioned at the beginning of section 6.4.3 without proving it).
- Demonstrate that the *last* canonical components (the kernel of the projection on the first components) are disproportionately far from the class components (*i.e.*, show that fig 6.19 happens in practice).
- Show that whitening the features solves the problem.

We compare the canonical components found with CCA on the original data B_1 to the class components W_1, and we repeat this process with the whitened data ($\hat{B}_1$, $\hat{W}_1$). Each time, we use Gram-Schmidt to obtain orthonormal bases, before comparing the bases using different measures of the distance between the class components and the canonical components.

Now, distances computed on these high-dimensional data are hard to interpret. To provide some elements for comparison, we add two *baselines*:

- `random orthogonal projection`: the first n_s components of an orthogonal projection chosen at random.
- `PCA`: Kamoi *et al* [286] showed that the highest variance components are equal to the class components, and used this fact for outlier detection. If the canonical components are as close to the class components as the principal components, we will consider it to be enough for practical applications.

Note: We do not try to use whitened data with these two baseline comparisons, because these projections are already orthogonal: there is no need to distort the space even further. Additionally, computing the PCA of whitened data makes no sense, as the covariance of the whitened data is the identity.

For each of these four comparisons (CCA, whitened CCA, random projection, PCA), we show that the first n_c projection components are equal to a linear combination of the class components. To do so, we will consider the distances between the first n_s projection components and the class components, where n_s gradually increases from 1 to the dimension of the subspace.

An illustration of the anisotropy

The next pages will be the occasion to demonstrate rigorously both the fact that whitening is needed and the fact that the first canonical components are close to the class components, up to a linear relationship (B). However, the previous explanation involved concepts that might be fairly unusual, and this novelty might translate into experimental errors: there are chances that we did something wrong without noticing it. Before tackling the full proof with mathematical distances in section 6.4.3, we will validate the results using methods that are less rigorous, but easier to understand, than the experiments we lead in the section before. If these measures are not real distances, they still give indicate how close can two subspaces be. Like the distances, they have the property that the farthest element to any subspace is its orthogonal complement.

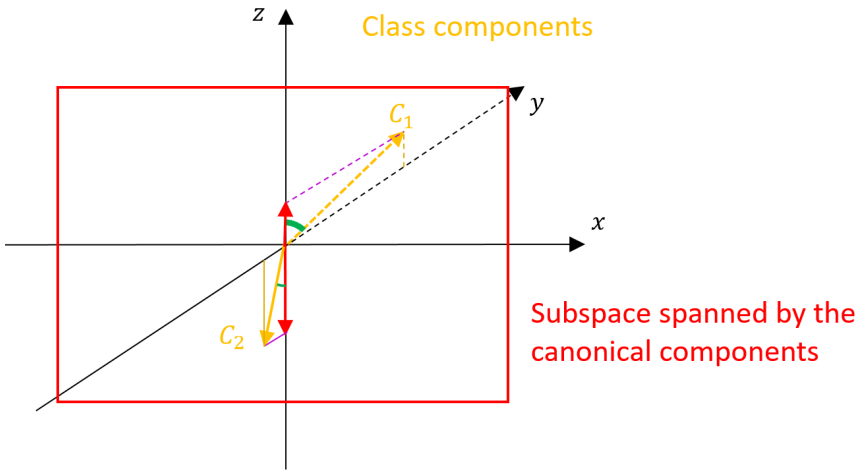
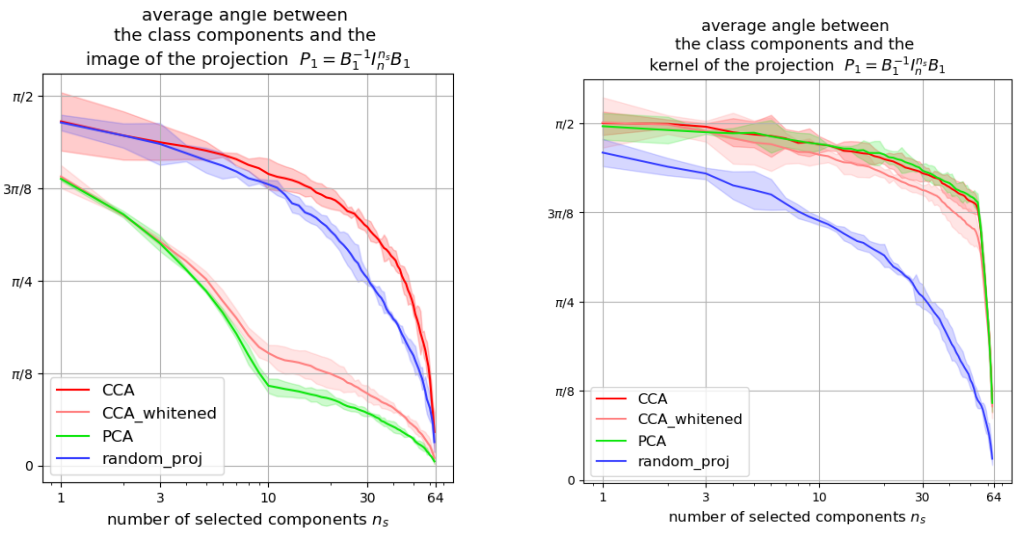


Figure 6.20: An illustration of the measure of the angle. Component C_1 (yellow, behind the red plane) is quite close to the y axis, which means it is almost orthogonal to the subspace spanned by the canonical components (xOz plane, in red). Hence, the angle (in green) between this component and its projection on the plane (in red) will be high. On the other hand, component C_2 is closer to the plane, ant the angle between this component and its projection is smaller. We compute an average for all class components, and use this average to characterize the proximity between the subspace and the class components



(a) (b)

Figure 6.21: The average angle between a class components and its projection on the first (a) or last (b) n_s components.

Angle between components With CIFAR, we plot the average angle between the $n_c = 10$ class components and their projection on the subspaces (see fig. 6.20 for an illustration). As fig 6.21a shows, the angle between the class components and their projection is still quite high[3] with $n = 10$ conserved dimensions. This is because two random directions are likely to be orthogonal in high dimension. We want to draw attention to two behaviours:

- The red curves (non-whitened CCA components) have different behaviours depending if we are looking at the first (fig. 6.21a, left) or last (fig. 6.21b, right). The distance between the image of the projection (the most correlated components) look to be extremely far from the 'PCA' curve in green. On the other hand, the distance between the last components (kernel of the projection) is extremely similar to the 'PCA' curve. This illustrates one of the seeming inconsistencies we observe because the components are not orthogonal.
- The pink curve (n_s most correlated components, after whitening) is similar to (albeit higher than) the curve of the highest variance components, whether we are looking at the image or the kernel of the projection. As we already know that highest variance components are close to class components [286], this experiment tends to conclude to the proximity between CCA and class components.

Projection of random vectors To confirm the previous results, we resorted to another measure of proximity between components. We generate 10,000 random vectors on the unit sphere, and we project these vectors on each subspace we want to compare to the class components. If one subspace is close to the class component subspace, the norms of both projections will vary in the same amounts (if one norm is high, the other will be high). In other words, the correlation between the 10,000 couples of norms will be close to 1. This is illustrated by the green and yellow subspaces in fig. 6.22. On the other hand, if the subspaces are very different from each other, the correlation is low (close to zero or even negative, *cf.* the yellow versus red subspaces in fig. 6.22). Figure 6.23 provides the results: again, the most correlated components (red, left figure) are not different from the random directions, while the canonical components created from whitened features are (pink).

[3] higher than the distance between the class components and random components.

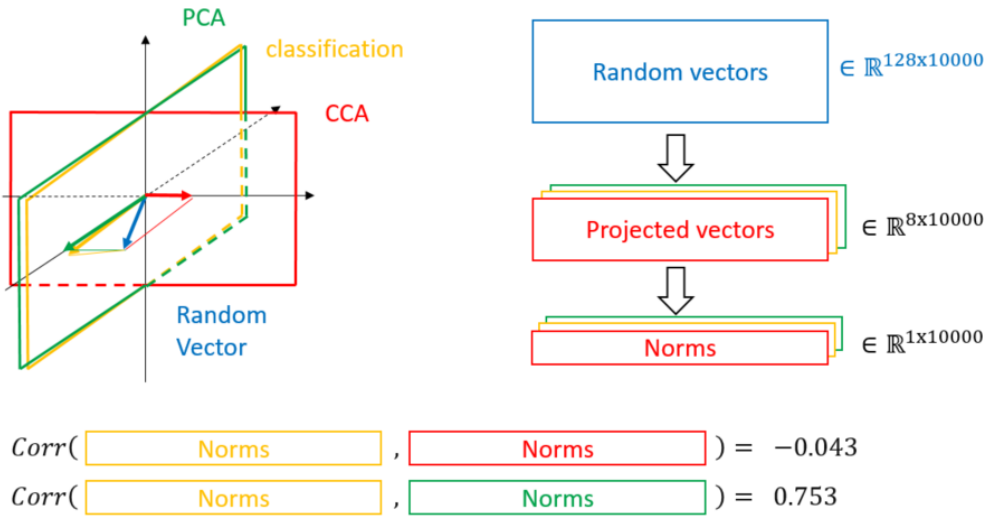


Figure 6.22: An illustration of the principle of the random projections experiment. When two subspaces are close (*e.g.* the yellow and green), the norms of the projections of a random vector will be close to each other. Inversely, when the subspaces are far from each other, the correlation coefficient between the norms will be low.

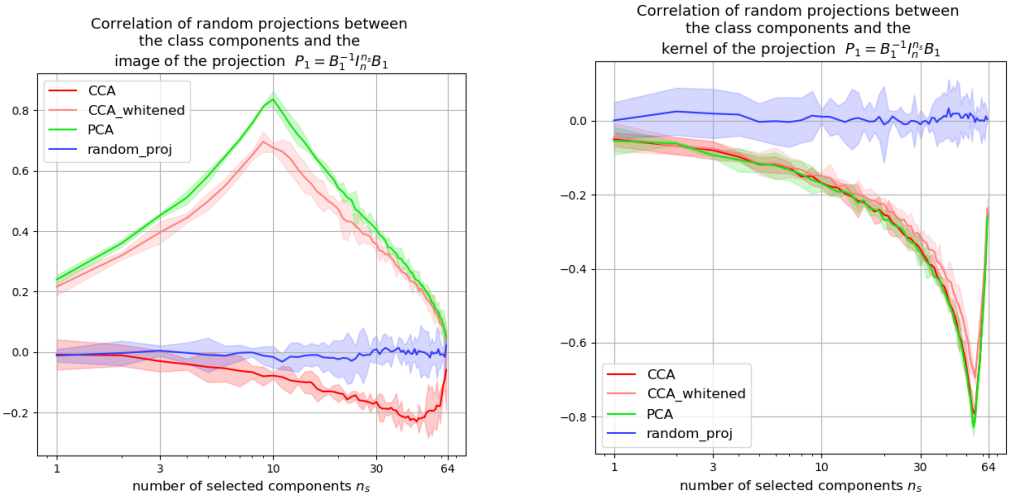


Figure 6.23: The correlation between the norms of random vectors projected on the subspace of class components and their projection on the first (a) or last (b) n_s components.

Distance Experiments

Subspace distances In this section, we will see how to compare the n_c components of W_1 to the first n_c components of B_1 rigorously, and without relying on any dataset. From a more abstract point of view, the problem is the following: we have two families of vectors, F (the first canonical components) and G (the class components), belonging in the same vector space. We want to know whether there is a linear relationship between the vectors of F and the vectors of G. As we mentioned at the beginning of section 6.4 simple one-to-one comparison is not enough to cover all possible linear relationships. To do so, we compare the subspaces spanned by each of these families. If the subspaces are equal, this means that we can compute

the vectors of G (resp. F) using vectors that belong in a basis of the subspace spanned by F (resp. G). In other words, if $F = (f_i)_{1 \geq i \geq n_F}$ and $G = (g_j)_{1 \geq j \geq n_G}$, the equality of the subspaces implies:

$\forall i \in [1..n_f], \exists (m_{i,j})_{1 \geq j \geq n_G}, f_i = \sum_{j=1}^{n_G} m_{i,j} g_j$

The matrix $M = (m_{i,j})_{i,j}$ is the matrix we were looking for in eq. 6.2.

We will use continuous distances: for instance, the comparison cannot rely on the rank of the family $F \bigcup G$ because the rank is not robust to small variations, as fig. 6.9b illustrates. We find orthonormal bases for the subspaces spanned by F and G using Gram-Schmidt (which we name U and V), and we use distances from the literature between subspaces applied on these bases.

If U and V are two subspaces of $\mathbb{R}^n$ with dimensions m_U, m_V, respectively[4], we make use of the following three distances to measure the distance between two subspaces of $\mathbb{R}^n$:

- **Frobenius distance** [289]: $||UU^\top - VV^\top||_F$ where $||.||_F$ is the Frobenius norm
- **nuclear distance** [289]: $||UU^\top - VV^\top||_*$ where $||.||_*$ is the nuclear norm
- **Wang-Wang-Feng Subspace distance** [291]: $\sqrt{max(m_U, m_V) - Tr(UU^\top VV^\top)}$, where Tr is the trace of a matrix

Note that these distances all rely on $UU^\top$, the orthogonal projection on subspace U. In particular, the first two distances derive from the norm of a difference of projection matrices [289], hence their names.

These distances are bounded, but the bound depends on the dimension of the subspaces. For instance, the maximal WWF-SSD distance between two subspaces U, V is inferior or equal to:

$\sqrt{max(dim(U), dim(V))}$.

To be able to compare distances when the number of dimensions varies, we display the *normalized distances*: we divide each distance by the maximal value it could take given the dimensions of the spaces. Table 6.2 displays the maximal value, for each of the distances we use.

Distance	Expression	maximal value
Frobenius	$\|\|UU^\top - VV^\top\|\|_F$	$\sqrt{m_U + m_V / 2}$
nuclear	$\|\|UU^\top - VV^\top\|\|_*$	$m_U + m_V / \sqrt{2}$
WWF-SSD	$\sqrt{max(m_U, m_V) - Tr(UU^\top VV^\top)}$	$\sqrt{max(m_U, m_V)}$

Table 6.2: The distances we use to compute the proximity between class and canonical components. $U \in \mathbb{R}^{n \times m_U}$ and $V \in \mathbb{R}^{n \times m_V}$ are two orthonormal bases for two subspaces of $\mathbb{R}^n$, $||.||_F$ is the Frobenius norm, $||.||_*$ is the nuclear norm, and Tr is the trace of a matrix

[4]Note that we use a matrix $U \in \mathbb{R}^{n \times m}$ to denote both a vector space of dimension m in $\mathbb{R}^n$ and an orthonormal basis for this space ($U^\top U = I_m$). The distances we use do not depend on the choice of a basis for the spaces [289, 290].

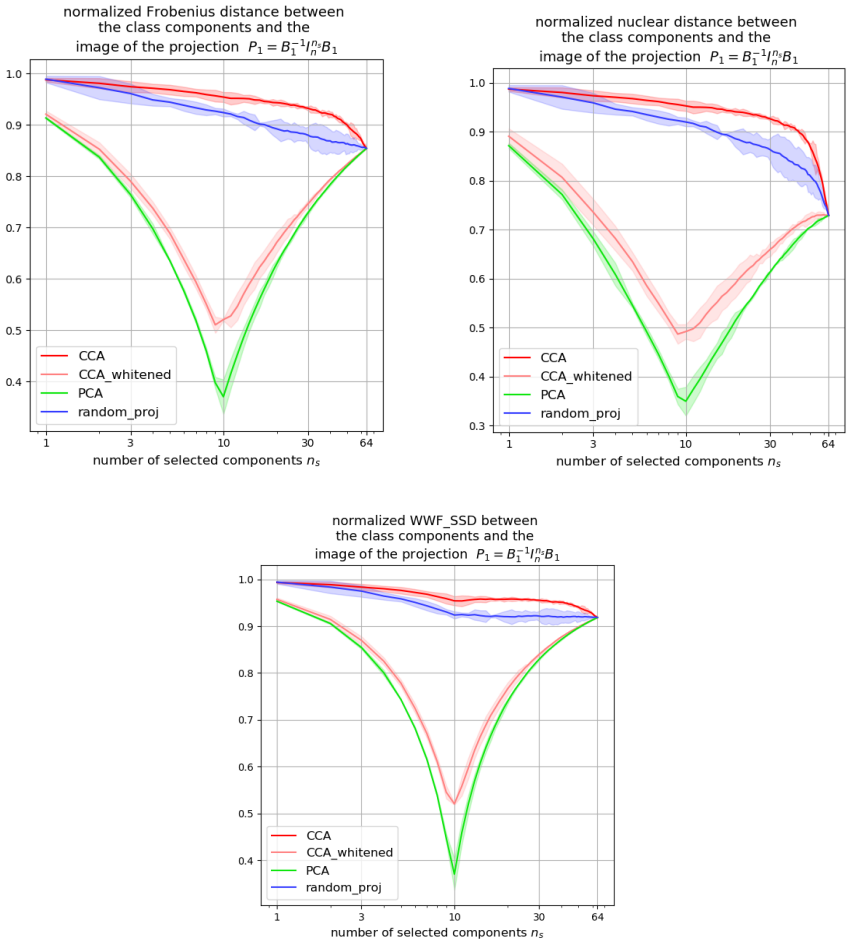


Figure 6.24: The distance between the class components and the first n_s components of the image of diverse projections. We display the average over three runs, the width of the curve denotes three times the standard deviation.

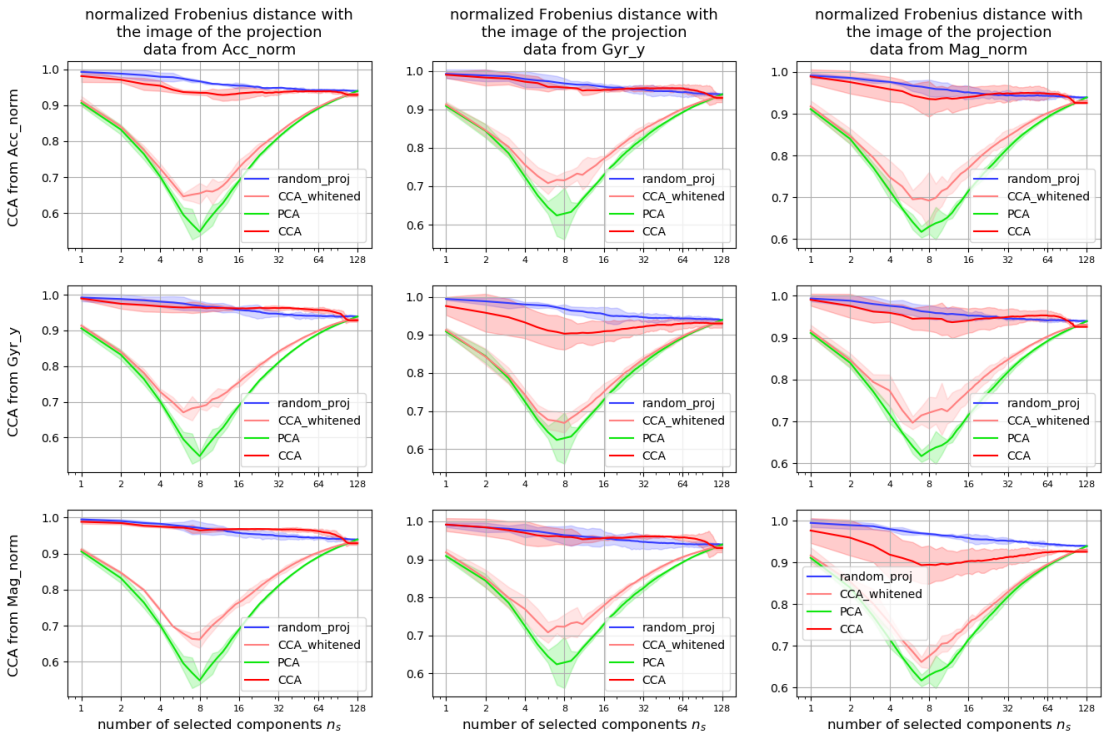


Figure 6.25: The Frobenius distance between the image of the projection and the class components, with SHL data. We display the average over three runs, the width of the curve denotes three times the standard deviation.

Distances between the image of the projection and the class components Figures 6.24 and 6.25 display the distances between subspaces with a varying number of dimensions and the subspace spanned by the class components. Note that for the sake of clarity, we display only the Frobenius distance with the SHL networks, the curves with the other distances have a similar aspect. In each of the experiments, even the minimal distance seems to be quite high: between 0.3 and 0.4 for CIFAR 10, and about 0.6 with SHL. This is a specificity of the high dimensionality of the feature space. Even if the distance is continuous, the sheer number of dimensions tends to pull points away from each other, which makes us unlikely to see small distances. To show it, one can look at how a random projection is consistently far away from the class components. This is why the distances between CCA and class components should not be compared to 0 and 1, but the baselines (PCA components and random components). The display of distances between 0 and 1 is mostly useful to compare distances with a varying number of kept components.

About the baselines, one can notice the first point of our argumentation: the distance between canonical components and class components (red) is similar to the distance between random components and class components (blue). This is why a direct measurement is misleading. By whitening the data, we can reach distance levels (in pink) that are closer to the distance between the highest variance components and class components (green). We should note that there remains a significant difference between the two.

Distances between the kernel of the projection and the class components In this section, we display the distance between the class components and the kernel of the projection $P_1 = B_1^{-1} I_n^{n_s} B_1$, which is no other than the *last* n_s components of the basis B_1 (as a reminder, the image of the projection is the subspace spanned by the first n_s components).

Figure 6.26 displays the distances between the kernel of diverse projections and the subspace spanned by the class components. Compared to the previous section, the meaning of closeness are reversed: if the kernel and the class components are close, the class components are the first to be erased by the projection.

As we mentioned in section 6.4.3, we can see that the kernel of the projection on the last n_s canonical components (red) is as far from the class components as the projection on the lowest variance components

(green). Whitening the data (pink) slightly lowers this distance, but the curve is still significantly higher than the distance to random components. For the three non-random curves (CCA, whitened CCA, and PCA), the distance between the last n_s components only gets lower when n_s becomes greater than $n - n_s = 56$ (whith the CIFAR dataset, with $n = 64$ features and $n_c = 8$ classes) when the last components start including the class components themselves.

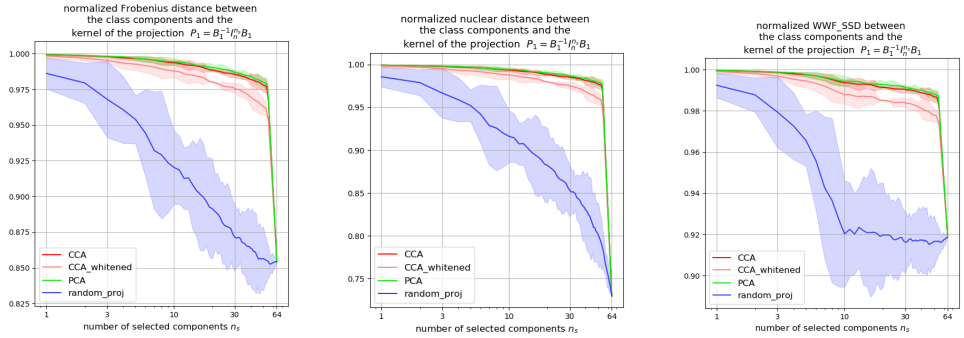


Figure 6.26: The distance between the class components and the first n_s components of the kernel of diverse projections. We display the average over three runs, the width of the curve denotes three times the standard deviation.

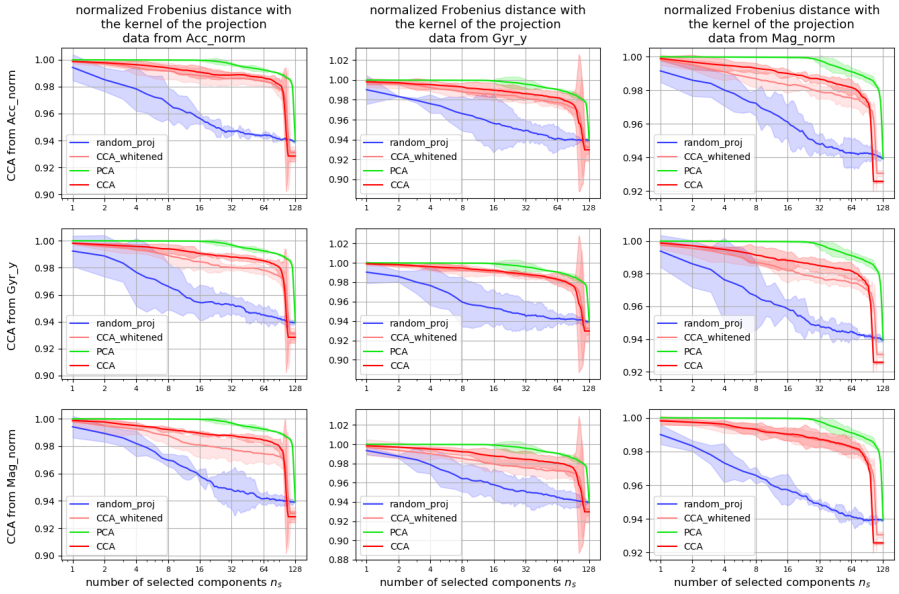


Figure 6.27: The Frobenius distance between the kernel of the projection and the class components, with SHL data. We display the average over three runs, the width of the curve denotes three times the standard deviation.

6.4.4 Partial conclusion: the proximity between class and canonical components

In this section, we demonstrated that the class components are equal to a linear combination of the first n_c canonical components (proposition B of our introduction). This affirmation was not trivial to establish: we first used the projection experiments in the literature (section 6.4.2), before criticizing them because they rely on a classification performance. We introduced an important caveat that one needs to care about before using more rigorous experiments: the fact that the canonical components are not orthogonal implies the

need for us to whiten the features before using distances that assumed the farthest subspace to any subspace is its orthogonal.

This section was the longest and the most complex of the chapter. Now that we know that proposition B takes place in practice, we will study its causes, and demonstrate that the fact that the classes are well separated in the feature space (A) implies the proximity between the class and the first canonical components (B).

6.5 The causes of the equality

This section is devoted to showing that $A \implies B$. To write it explicitly, we will show that the fact that the classes are well separated in the feature space (A) is enough to cause the class components to become close to a linear combination of the first canonical components (B). In short, we will modify anything but A, and see that B still occurs.

To detail the concepts we need to use, one could look at fig. 6.28, which explains the difference between inter-class correlation and intra-class correlation. The results in the previous section (the fact that CCA picks up the class components) show that there is some kind of correlation, which means that the top-left scenario is impossible. Similarly, the fact that the networks have reliable performance (70 to 90 % F1 score) means that the down-left scenario is unlikely: there must be a linear way to split the classes and hence, some kind of inter-class correlation. We are left with two scenarios: either the top-right scenario, in which there is no intra-class correlation between the logits, and CCA only picks up class components because the two networks are good at classifying the samples; or the down-right scenario, which implies that two different networks assign similar logits to a sample, a similarity which goes beyond simply belonging in the right class.

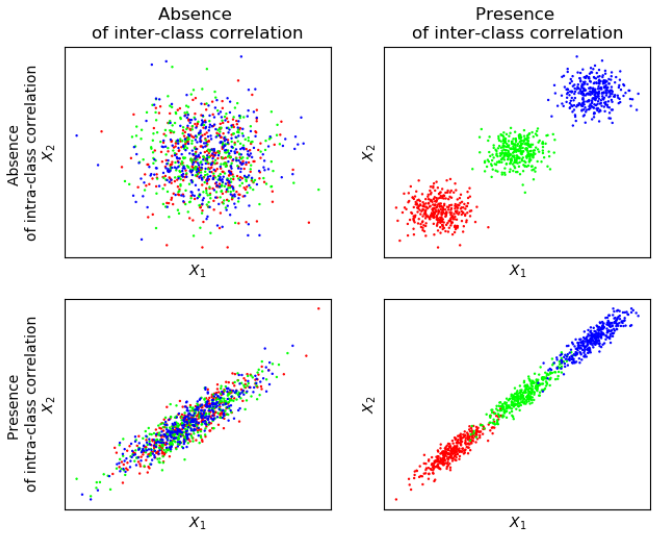


Figure 6.28: the different types of correlation between samples illustrated with synthetic data: inter-class correlation (left versus right) and intra-class correlation (top versus down)

Among the conditions that enable this observation, the easiest to verify empirically is the first one: the class logits of different networks are more correlated than other components, whether across initializations or sensors. After the networks are trained, we look at the logits of each class on the *validation* set. For instance, fig. 6.29b tells us that the correlation of the class logits predicted by the first and second initializations averaged over the 10 CIFAR classes, is 0.90.

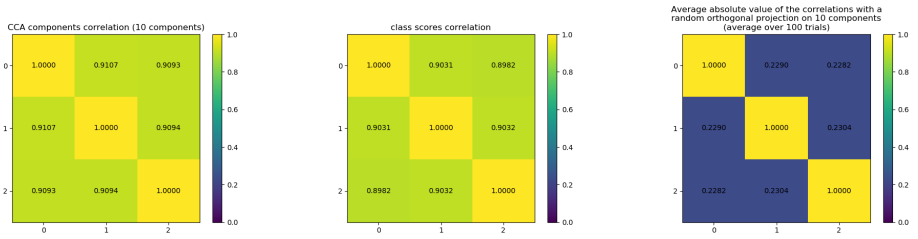


Figure 6.29: The average of the absolute values of the correlations between (a) the first 10 CCA components (b) the 10 class logits (c) 10 directions chosen with a random orthogonal projection, with three networks trained on the CIFAR dataset with a different seed.

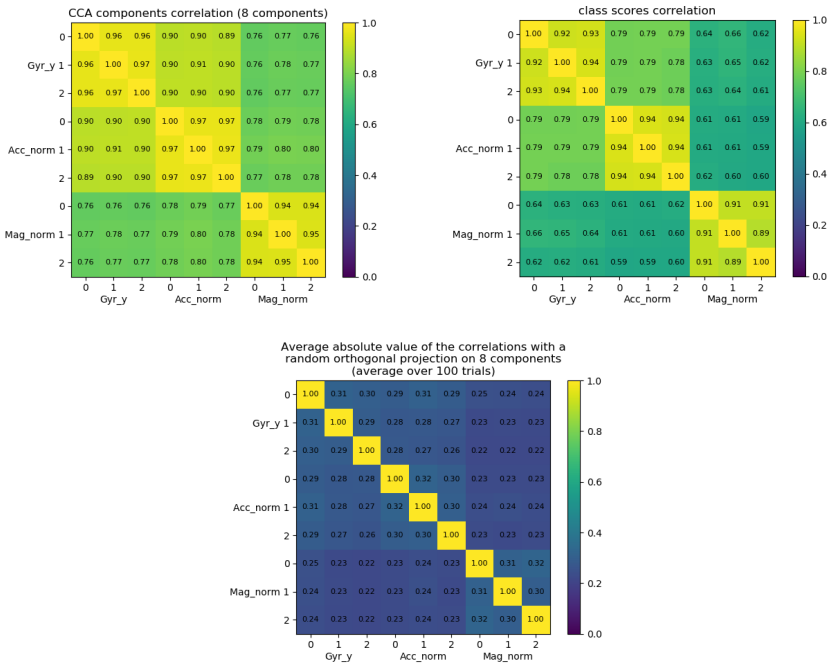


Figure 6.30: The average of the absolute values of the correlations(a) the first 8 CCA components (b) the 8 class logits (c) 8 directions chosen with a random orthogonal projection, with the SHL dataset. For each sensor, we create three different initializations of a network using the sensor. As expected by the intuition, the accelerometer and gyrometer feature and predictions are closer to each other than they are close to the magnetometer.

As CCA find components with maximal correlation, and as the logits can be obtained with a linear combination of the features, the correlation of the n_c first CCA components will be higher than the correlation of the n_c class logits. However, fig. 6.29 and 6.30 show the correlations of the class logits are quite close to the correlation of CCA components, they are much closer to each other than to a random linear combination of features. This seems to indicate that the organization of the features is the one of the bottom scenario in fig. 6.28: there are strong intra-class correlations within the data. However, such a strong correlation between logits is not necessary, and the new experiments are there to prove it.

To verify this, we do not use synthetic data, because there is a risk that the CCA bases its calculation on statistical properties from X_1, X_2 which we are not aware of. To account for it, we use an operation that removes the correlations while keeping the statistical properties of each random variable intact: the

shuffle. We simply take each feature vector (the rows of X_1 and X_2), and assign them a new position at random in the matrix, with uniform probability. This way, any of the statistical properties of X_1 (mean, standard deviation, moments of any order) are kept (they do not depend on the order of the samples in the matrix), but the correlations between the features are destroyed, because the alignment between samples is broken: any sample in X_1 faces a random sample in the database in X_2. Now, shuffling completely at random is not extremely interesting to us, because such a shuffle would create canonical components that are completely devoid of meaning. We wanted to measure the importance of class clustering. To account for class membership, we use a *class shuffle*: when we assign each sample x a new position, we make sure the new position is the position of a sample in the same class as x. This way, any sample from X_1 will only face a sample at random belonging to the same class as x. If class clustering is the *only* reason why canonical components are close to class components (top-right scenario in fig. 6.28), then computing the CCA between class-shuffled feature matrices X_1, X_2 would not change the result. On the other hand, if the phenomenon we observe is due to intra-class correlations, then computing CCA on class-shuffled data would show considerably less proximity between class-shuffled and normal data.

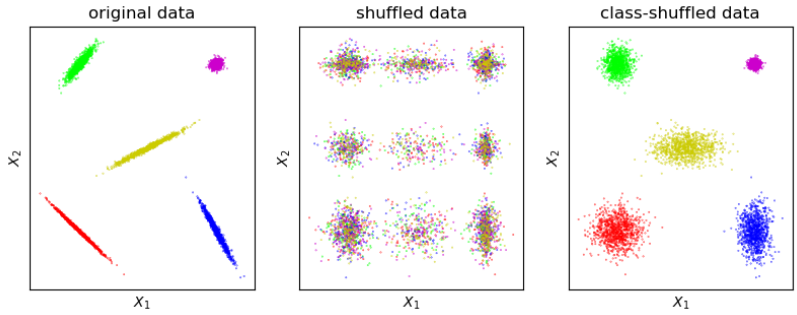


Figure 6.31: An example of class shuffling on synthetic data (the colour represents the class). Random shuffle destroys the correlation, and class-shuffle allows to destroy intra-class correlations while keeping inter-class clustering intact.

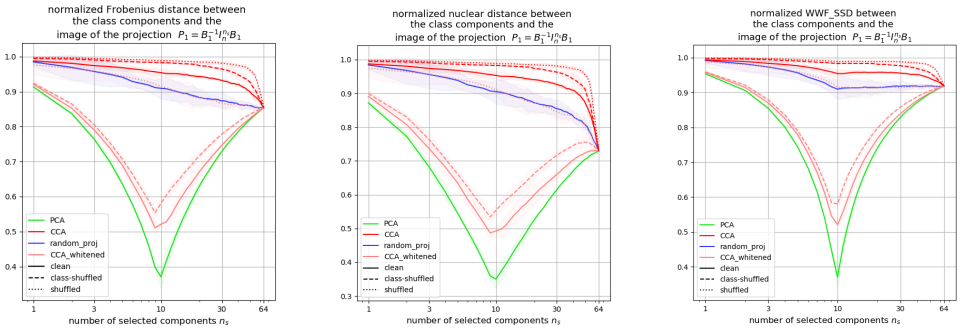


Figure 6.32: The distance between the class components and the first n_s components of the image of diverse projections computed on clean or shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation.

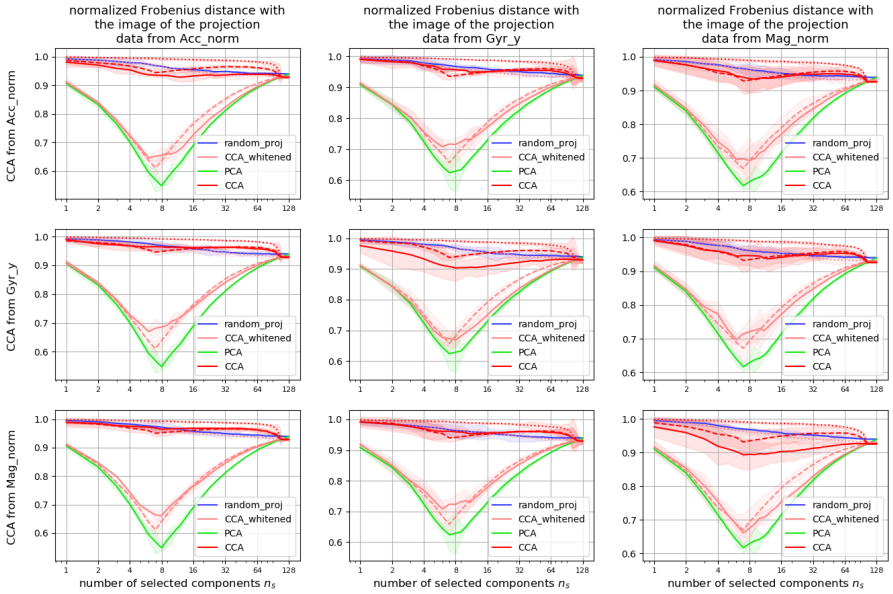


Figure 6.33: The Frobenius distance between the the class components and the image of the projection computed from clean and shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation.

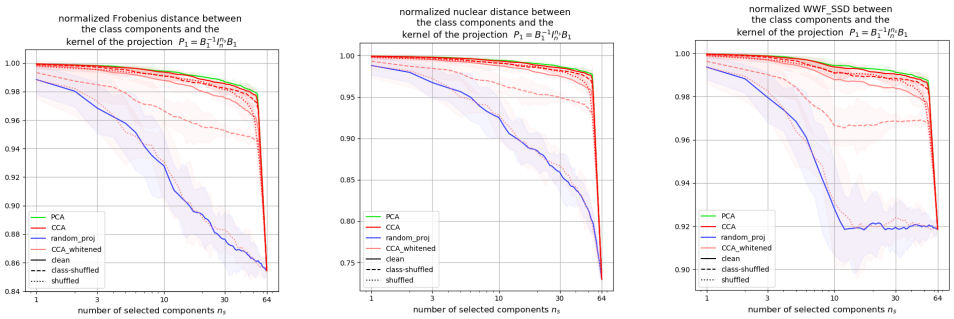


Figure 6.34: The distance between the class components and the first n_s components of the kernel of diverse projections computed on clean or shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation.

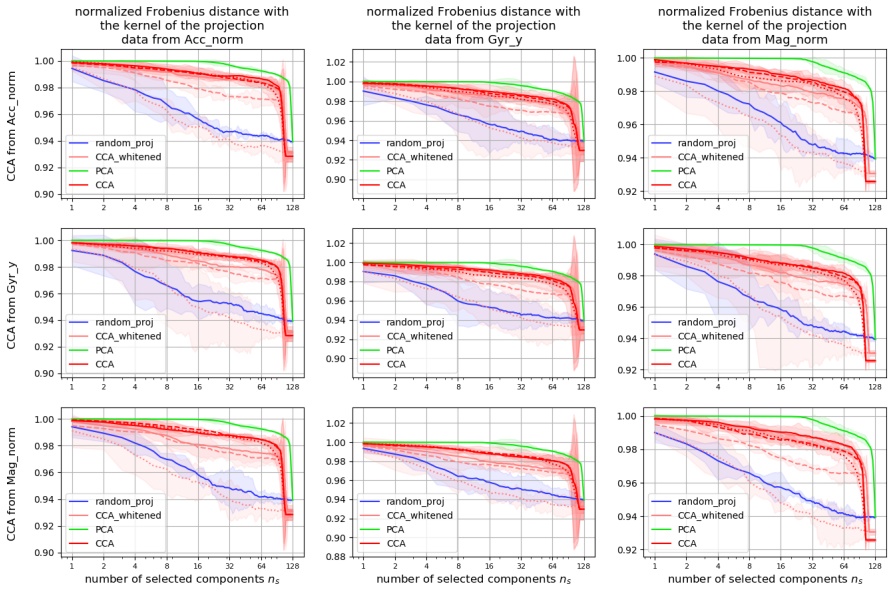


Figure 6.35: The Frobenius distance between the the class components and the kernel of the projection computed from clean and shuffled data. We display the average over three runs, the width of the curve denotes three times the standard deviation.

We repeat the experiments from section 6.4.3, except that we also compute the CCA components from shuffled and class-shuffled data (shuffling the data does not change anything for PCA). When considering shuffled data, the canonical components are equivalent to random components. But when considering the class-shuffled canonical components, the curve is relatively close to the curve with normal data, while still significantly distinct. It looks like the distance between the class-shuffled curve and the normal curve is much smaller than the distance between normal and random curves. This is why we think that class clustering accounts for the majority of the proximity we observe: class clustering does play a major role in the proximity we observe, but it is not the only factor. Intra-class correlations have little influence over the canonical components. In other words, even if the logits were not correlated, the CCA would still put the class components first.

Now, when we say "$A \implies B$", one might wonder how much A needs to be true for B to take place. tha is, if the equality between the class and the first canonical components (B) might only occur in extreme cases when the classes are extremely well separated. For instance, the SHL network with the accelerometer data reaches 90 % validation F1-score, similarly to the CIFAR 10 network. However, figure 6.33 shows that B still happens with the gyrometer and magnetometer features, which have validation performances of 80 % and 66 %, respectively.

Now, one could argue that these sensors are still similar to each other: in chapter 3, figure 3.6 shows that the norm of the magnetometer still carries some information about the dynamics of the phone (we see traces of a $2Hz$ Dirac comb in the norm of the magnetometer), a piece of information that is prevalent in the accelerometer and gyrometer signals. One could argue that with networks where modalities are much more different to each other (for instance, audio and video, or text and image), the results might not hold. These critics are entirely valid, and the lack of reproduction with other signals is the improvement we estimate to be the most important for this chapter.

But for now, let us pursue with the SHL and CIFAR datasets. On these problems at least, we showed that B (the fact that class components are equal to the first canonical components up to a linear relationship) happens in practice and that $A \implies B$. The next sections will show how these propositions relate to the CCA fusion we presented at the beginning of the chapter.

6.6 How the equality between class and canonical components implies that a CCA fusion is ineffective

The fact that the networks we consider have good classification accuracies means that the classes are well separated in the feature space (A). The previous sections demonstrated that the class components are equal to the canonical components up to a linear transformation (B, section 6.4) and that $A \implies B$ (section 6.5). This short section aims to show how B implies C (the fact that a CCA fusion is equivalent to a sum of logits).

Let us assume that we have B: the first canonical components are equal to a linear combination of the class components. This means that the classification information is kept in these components and that the other components are less informative about the classification problem.

Now, one could argue that there might still remain some information that is relevant to the classification problem in the $n-n_c$ *last* canonical components. They would be right. In fact, if we train a classifier on these last components, we will obtain a nonzero classification accuracy. However, we argue that this information is less practical, less effective, than the class components. To show it, let us focus on the classification layer in the network. As this layer is linear, it means that the network optimized the classification layer to classify the hidden features.

Now, in the general case, the full set of weights of a network is not always able to reach 100 % accuracy, so a neural network does not always find the perfect weights to solve a given problem. But the classification layer is linear: it means that we know it is able to reach an optimum easily (a linear problem is convex, and gradient descent always finds the optimum of a convex problem).

Note that the full, rigorous, mathematical guarantees applies only to the case where the features that train the linear classifier are fixed. In the case of a neural network, the input features of the classification layer still change during the training, as the weights of the other layers are optimized. There might be a possibility for the hidden features to move away from the classification layer, and the classification layer might be unable to catch up to them. To summarize it is possible that the classification layer and the features play a cat-and-mouse game, without the former ever reaching the latter. According to this possibility, at every instant t, the classification layer is suboptimal to classify the hidden features. However, we argue that this possibility is unlikely to happen in practice for two reasons:

- Neural networks learn their earlier layers first [269], which means that the classification layer will eventually catch up to the hidden features, even if the beginning of the training got the features farther from the class components.
- Linear, Multi-layer models (mathematical approximations of neural networks) trained by gradient descent are known to 'align the layers' ([292]). In other words, the hidden features have been shown to move *towards* the class components, and vice-versa.

To sum up, the class components are directly optimized to focus on the most useful components, and their behaviour is simple enough (linear) for us to assume safely that finding the optimum is feasible.

In other words, given that the class components are the components that classify the best the features, a novel classifier that sees the hidden features will likely look at the class components. When the classifier is presented with canonical variables X'_1, X'_2, this means it will look at the first n_c features first. And when the classifier sees a sum of canonical variables $X'_1 + X'_2$, it will look at the first components of the sum. As the first canonical components are equal to a linear combination of the class logits, this means the classifier will look at a sum of logits first. Now given that the neural network optimized its logits for linear classification, it is unlikely that the classifier learns much more than what the network learnt: the classifier using features that are essentially a sum of logits will not be likely to reach better performances than the mere sum of logits.

6.7 An implementation of CCA fusion with SHL

Up to now, we have shown that the separation of the classes in the feature space (A) imply that the CCA recomputes the logits (B), which implies that the CCA fusion is equivalent to a sum of logits (C). We know

that A takes place in practice because the performance of the networks is high (at least 66 % F1-score), and we know that B occurs thanks to the complex distance experiments we led in section 6.4. To conclude this chapter, we aim to demonstrate experimentally that the fusion method relying on the canonical variables computed from deep features is equivalent to a sum of the logits obtained from the network (C).

We implement a classic CCA fusion and compare it to the performance of a model which merges the data using a sum of logits. Note that even though we call this model a classifier in the rest of the section, we do not train it in itself, because it relies on interpretable logits of trained networks. In order to reduce the variance of the results, and given that both methods use networks that are trained using a single sensor, we re-use the neural networks: each couple of networks is used once for the CCA fusion, and once for the sum of logits. This way, we remove one source of randomness in the experiment. However, we still repeat the experiment (training of a couple of networks) five times. The following pseudo-code algorithm details the experimental protocol:

Require: A couple of sensors s_1, s_2.
  for i in $\{1, 2, 3, 4, 5\}$ **do**
    Train a first neural network on data from s_1, record the embedded training features X_1
    Train a second neural network on data from s_2, record the embedded training features X_2
    Compute the PCA on each of X_1 and X_2, keep only 99.99 % of the variance
    Compute the canonical variables X'_1, X'_2
    Train a SVM classifier to classify the sum $X'_1 + X'_2$, measure its validation performance
    Measure the validation performance of a classifier which only considers the sum of the output logits of the two trained networks
  end for
  return the mean performance of the CCA fusion and sum of logits classifier.

Parameter	value
C (regularization parameter)	1.0
kernel	RBF ($\gamma = \frac{1}{\sigma_X * \sqrt{n}}$)
multiclass strategy	One-versus-rest

Table 6.3: The parameters of the SVM classifier

To classify the sum of canonical variables, we use a SVM classifier which parameters are given in table 6.3. Table 6.4 gives the results. Surprisingly, the CCA fusion is slightly worse than a simple sum of logits. We assume that SVM overfits to the data[5]. The reasoning in the previous section omitted an important point: when we said that the last layer of a network is optimized to the features of the network, we must keep in mind that the network only sees the *training* data. Nothing prevents the network or the SVM classifier to overfit, and they might not overfit the same way.

	Sum of logits	CCA fusion	Restricted CCA fusion
$\lvert Acc\rvert, Gyr_y$	$90.76 \pm 0.87\%$	$88.66 \pm 0.35\%$	$88.90 \pm 0.70\%$
$\lvert Acc\rvert, \lvert Mag\rvert$	$91.27 \pm 0.31\%$	$90.67 \pm 0.40\%$	$90.32 \pm 0.55\%$

Table 6.4: The results of a CCA fusion, using diverse combination of sensors with the SHL dataset. The table displays the average and standard deviation over five random runs

To check this hypothesis, we use only the first components: instead of asking SVM to classify the samples using all the components in the sum $X'_1 + X'_2$, we select the first n_c components of this sum ($n_c = 8$ with the SHL dataset), and train the SVM to classify the samples using the selected components ($(X'_1 + X'_2) * I_n^{n_c}$).

[5]We tried this regularization parameter `C`: we first experimented using a search with a log scale between 10^{-4} and 10^4, with a step factor of $\times\sqrt{10}$, which resulted in having the optimal C equal to 1.0. Then, we experimented using a linear scale in $[0.1, 5.0]$ with a step of 0.1. As the difference in performance between the optimal ($C = 1.7$) and the default ($C = 1.0$) regularization parameters was less than 0.05%, we kept using the default value. We did not try changing the other parameters.

The result is in the column 'restricted CCA fusion' in table 6.3. The fact that the performance of this method is the same as the performance of the CCA fusion with full components means that the trained SVM uses only the first n_c components, which is a strong indication that the n_c most correlated components are the most interesting for classification.

6.8 Varying the layer where features are extracted

In this experiment, we try to see what is the influence of the layer the features are extracted from. To repeat, this section does not appear in the $A \implies B \implies C$ reasoning that serves as a frame for the second half of this chapter. However, this is one important hypothesis we made in section 6.2 to work with when we talk about B.

Previously, we mostly focused on features from the ultimate layer of the network, features from which the class logits can be obtained with a simple linear transformation. What about the feature from earlier layers? If we were to apply the fusion method described in [267, 249], would it be relevant?

This experiment is similar to the projection experiments in section 6.4, except that we always choose to project on a subspace with n_c dimensions, where n_c is the number of classes. The x-axis now denotes the layer the features come from. The fact that we resort to projection experiments is only because there is no obvious equivalent to the class components in the other layers than the last. We use experiments we know to be imperfect because we found no other way to explore the question.

When there is a correspondence between CCA components and class logits, it is likely that the method described in [267, 249] will be equivalent to a simple logits average. In other words, when the performance of the original network is unchanged when the features are projected onto a n_c-dimensional subspace, the CCA operation only picks up directions which are the inverse image of the class logits by the transformation of the layers. Figure 6.36 provides the results: before the last layer of the network, the accuracy drops significantly, which means that proposition B does not really apply to features from earlier layers.

Please note that fig. 6.36 does not guarantee that applying CCA to other layers is *better* than an average of logits. It only means the reason why the results are equal no longer holds. The methods may have equal results for different reasons (or due to mere coincidence), and applying CCA may even be worse than computing an average. In fact, we hypothesize that classifying the most correlated components will be worse because the features that are used are not optimized for direct classification. However, the exact experiments to demonstrate it are out of scope for this work.

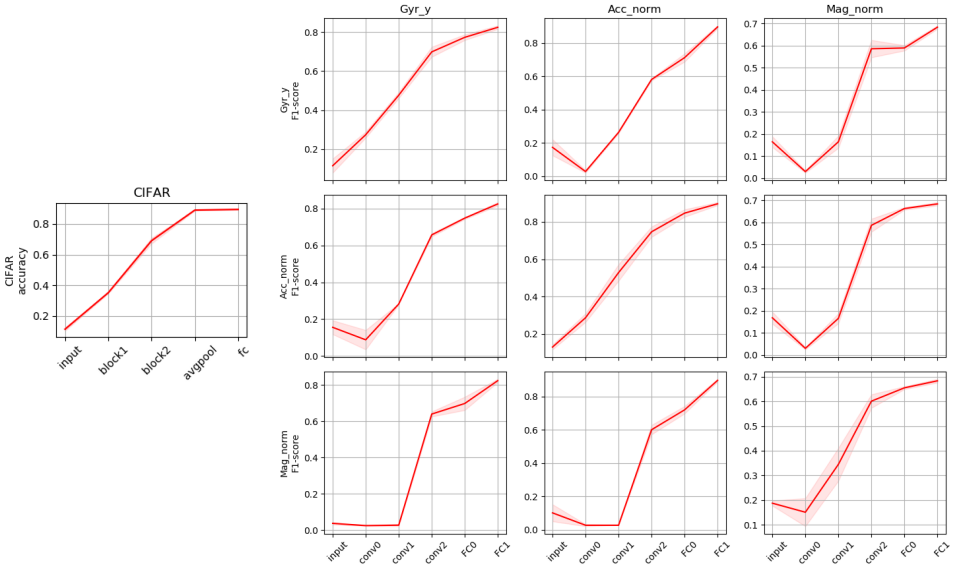


Figure 6.36: The performance when projecting the features from every layer using the sensor on top on the n_c most correlated components. The CCA is computed from the sensor on the left, using features from the same layer. The experiment is repeated across three network initializations, the standard deviation is given by the width of the curve. We pay attention not to use twice the same initialisation when using twice the same sensor.

6.9 Conclusion

A comment on the meaning of deep features

We would like to come back on one of the consequences of proposition B (the proximity between the first canonical components and the class components). We opened this thesis by saying that features from deep neural networks could eventually replace the handcrafted features. If features from deep networks are more effective than handcrafted feature, this means that the features from deep networks encode more interesting properties about the signals (properties that we can compute explicitly, like the power; properties that are harder to express mathematically; and properties that we are unaware of, but are nonetheless important to a general-purpose classification problem). If a neural network reliably computed a feature from a signal (a feature that is not covered by the class components), this feature would appear in several initializations of the network, which means the CCA would find it and put it first.

However, we observed that the CCA only puts first the class components. in other words, it seems that the networks only learns what we need it to learn: the classification information. The results from Kamoi *et al.* (classification information accounts for most of the variance of deep features [286]) also go in this direction.

This seems to contradict the very rationale of feature extraction computer vision. An incredible amount of works used these deep features and reliably showed how these features outperformed the classical handcrafted features from computer vision. We do not question the validity of the many papers which relied on the use of deep features. To show how these works are compatible with our results, we can say that knowing the logits is already a lot.

To be more precise, let us take an example and consider a typical example of the use of deep features: the works of Gu and Tresp [293]. One of their findings states that the feature embeddings can act as a classifier for new semantic concepts. For instance, one can use an ImageNet-pretrained model to detect the presence of reptile scales in an image, even though there is no class to encode this precise concept (the `scale` class designates the tool). To do so, one can regroup a series of images containing scale patterns, and

compute their feature using any pretrained model, and compute a simple average of these vectors. To know if a new, unseen image has scale patterns in it, one can just measure the distance between the vector of the unseen image and the average vector computed earlier, and apply a simple threshold. As we said, we believe these results to be completely valid and compatible with our hypothesis (the classification information summarizes most of the information present in deep features). We said that there was no class explicitly encoding the presence of reptile scales, but there are many classes of reptiles which happen to have scale patterns in them: lizards (`green_lizard`, `alligator_lizard`, and `frilled_lizard` are valid ImageNet classes), crocodiles (`African_crocodile`, `American_alligator`), snakes (thirteen ImageNet classes represent snakes[6]), *etc.* Knowing the classification information could prove useful to distinguish scale patterns: if it looks like a lizard and a snake, chances are the image has scales in it. By averaging the encoding of images with scale patterns, we might just average the class components for the classes of scaled animals.

This explanation looks convincing, but we must raise a warning: we do not state that this is what happens in practice, as we lead no experiment to prove either the hypothesis (the classification information summarizes most of the information present in deep features) or its application to the works of Gu and Tresp [293] (the average vector for a concept correlates with the classes covering this concept). The explanation we gave is only one way to solve the apparent contradiction between the results we presented and the current use of neural networks as feature extractors.

If the only information present in the last features was the classification information (and we cannot underline this *if* enough), we could give a piece of practical advice to a research team who would want to create a dataset equivalent to ImageNet with other sensors: they would need to have extremely diverse *classes*. This way, features extracted from the last layers would still carry relevant information. Were the classification problem to be only partial, the last layers would not be extremely useful. In this case, a researcher willing to use such a model could resort to retraining the model (pretraining improves convergence speeds even if the classification semantics do not intersect [5, 171]) or to use features from other layers than the last, similarly to [294, 295].

In other words, the unproven hypothesis we just formulated states that one of the key explanations for the success of ImageNet-pretrained models is not only the amount of data but also the diversity of *labels* the models were trained with.

How to build a dataset with many labels ?

If building a dataset with a large number of samples is quite straightforward, one could wonder how to gather a large number of varied classes. We can think of various ways, each of them having specific qualities and drawbacks:

- firstly, one could include rarer classes (such as motorbike, boat, for TMD). However, this first option changes the problem because it introduces classes that are extremely unlikely in the dataset (see [296] for examples of solutions).
- one could also add annotations to encompass a type of problem that is more general (for instance, solving general Human Activity Recognition instead of merely TMD). This solution might not help to learn better features for the original problem if the two problems are too different or decorrelated.
- the next solution is to increase the granularity of the labels: instead of asking a network to predict whether the user is in a bus, we could ask whether the bus has an electrical engine or a combustion engine, for instance. However, this requires to have access to levels of precision that are hard to achieve.
- Finally, one could also ask a network to guess information about the user (such as the gender [297]), but such a technology would harm the privacy of the users in such a direct fashion that we estimate preferable not to follow this way.

For instance, young adults are known to be on average 1cm taller in the morning [298] due to changes in the mechanical properties of the spine. We could hypothesize that asking a neural network to predict the time of the day would force it to exploit some information about the posture of the individual.

[6] `thunder_snake`, `ringneck_snake`, `hognose_snake`, `green_snake`, `king_snake`, `garter_snake`, `water_snake`, `vine_snake`, `night_snake`, `boa_constrictor`, `rock_python`, `Indian_cobra`, `green_mamba`, `sea_snake` and `horned_viper`

With this last example, we are approaching the domain of self-supervision, where we ask a network to solve a problem created by researchers, as a pretext to learn interesting features. A dissertation on the exact delimitation of self-supervised learning is out of scope for this work, but we will nonetheless say that we expect self-supervision to also be useful in helping the network to learn a useful general-purpose representation, as it has been with Natural Language Processing.

Summary and future work

We began this chapter with a presentation of Canonical Correlation Analysis and showed how this tool helps us to understand the features different neural networks learnt: we obtained a quantitative measure showing that the accelerometer is closer to the gyrometer than to the magnetometer, and we demonstrated that the network had access to the power of the original signal even before its training, a piece of information it could exploit to solve the TMD problem. We devoted a large portion of the chapter to show why this operation would likely not revolutionize data fusion with deep features: we showed that when the classes are well separated in the feature space, the CCA recomputed the class components. We showed that this implied the fact that a CCA fusion was equivalent to a sum of logits, and finally demonstrated experimentally the ineffectiveness of this fusion method. We briefly tackled the subject of understanding what happens with CCA on features from intermediate features, and showed that the reasoning we led for features from the last layer would likely not hold for other layers.

We hypothesize that we would observe the main result (the sequence of implications) generalizing well to any two networks (either networks using the same sensor or networks using different sensors) because section 6.5 showed that simply having relatively well-separated classes in the feature space is enough for CCA to put the class components first. However, to the current date, no experiment backs up this claim. Leading these experiments with other multimodal sensor problems could be part of future work. Another possibility for future work could be to work on unsupervised problems, using the theoretical formulation introduced by Roeder *et al.* Alternatively, we could think about generalizing the present results to the improvements of CCA we mentioned: PWCCA [274], CKA [299], or kernel-CCA [300]. Finally, the last way to continue the work would be to verify the difference of meaning between deep features and class logits, and in the case our hypothesis is proven, one could verify how to reconcile it with the current use of deep networks as feature extractors.

Chapter 7

Conclusion

7.1 Summary of the contributions

This thesis is aimed at exploring Deep Learning for Transport Mode Detection. If our original goal was to produce a general-purpose feature extractor, the lack of a "go-to" methodology quickly pushed us to try to know how to use deep networks in practice. To do so, we focused on two major research questions: How to preprocess the input signals ? And how to merge the data from different sensors ? During this work, we reached the following conclusions:

7.1.1 Preprocessing of input segments

We dedicated our first chapter (chapter 3) to the preprocessing one can apply to the input data before sending it to a network. We began by improving the padding of short segments we found in the literature (replacing the zero-padding with a wrapping), before trying to answer a question: should we leave our input segments intact, or should we compute a Fourier Transform, spectrogram, or scalogram, for the network to use? Surprisingly, this decision is not explored much in the literature. One publication did conclude that the most efficient representation depended on the number of samples: it seems that when the dataset is small, computing the spectrograms helps the network. If the dataset is large and varied, we ought to let the network earn its own features. By looking at the literature, we saw that the hypothesis seems to be verified, but we could not conclude. Given the lack of definitive proofs we had, we tried making the comparison ourselves. If the comparison "spectrogram versus raw data" was biased in favour of spectrograms, the comparison of the FFT and raw data demonstrated that computing the FFT does help the network. We also justified each of the steps of the computation of the spectrograms: resizing, log scale for the frequencies, and computation of the log of the energy. In a last section (section 3.4), we wanted to go beyond the empirical comparisons. We showed that the spectrograms made the problem linear for one of the classes, thus simplifying the problem. However, this simplification only happened because there was a way to solve the problem using frequency features linearly, and we expect this last result to apply only to Transport Mode Detection.

7.1.2 Global Pooling methods

In chapter 4, we studied one choice in the architecture design of neural networks. All convolutional architectures used in TMD required a flatten step to obtain a fixed-size vector from a one-dimensional sequence of representations. We introduced the use of Global Pooling methods from the Computer Vision literature in Transport Mode Detection. Not only did it allow to use segments of any size, but it also resulted in a significant decrease in the memory and computational requirements. The convolutional model we obtained to work on the GeoLife dataset could be reduced to 11,000 parameters, an extremely low memory size compared to the millions of weights neural networks usually have.

7.1.3 Data Fusion

The problem of Transport Mode Detection sometimes involves more than one sensor. Chapter 5 talked about the choice of an architecture to efficiently merge information about each sensor. The literature is extremely vast, and we selected several algorithms which had a rationale that we could apply to our problem. However, none of them did succeed in outmatching the most basic data fusion methods. We tried forcing the network to learn complementary features, and failed. But this failure proved useful: we showed that if providing a network with different signals helps it more than redundant signals, trying to sway it out of learning redundant features is ineffective. The network seems able to choose by itself the right optimum between redundancy and complementarity.

7.1.4 Canonical Correlation Analysis for data fusion

The last chapter (chapter 6) studied one specific data fusion method in detail, an algorithm that relied on a statistical operation named *Canonical Correlation Analysis*. Any classification network that succeeds in its task (*i.e.,* that has a high accuracy) will produce features that disentangle the classes: the classes are said to be well separated in the feature space. Our contribution was to demonstrate that this separation influenced the CCA, and made the operation recompute the class components first. Doing so was not easy, and we demonstrated the need to whiten the data in the process. Once we were sure that the CCA recomputed the same components as the classification layer, we came back to the original data fusion algorithm that interested us in the first place and demonstrated our main point: given that the CCA operation recomputes the same information as the classification layer, using CCA to perform data fusion is equivalent to using the class logits returned by the classification layer. Sadly, we showed that our whole reasoning was likely to be proper to the very last layer of the network, and we expect it not to hold for earlier layers. We concluded with some remarks on the fact that the reason why a network trained on ImageNet produces good general-purpose features is that this dataset has a good variety of classes. Pre-training a network to do the same for other signals requires similarly a dataset with a large and diverse set of *classes*.

Some of these problems were extremely vast, to the point that several chapters of this manuscript (pre-processing, multimodal fusion) could have been complete theses in themselves. Others would have deserved some experiments on more problems than only Transport Mode Detection.

What to retain from this thesis ?

Our experiments participate in reaching two general conclusions that might be useful for a future practitioner:

- Neural networks work best with spectrograms when the number of samples is 'small', and we should let the network learn its own features when the number of samples is large enough.
- To be able to use a neural network as a feature extractor, not only do we need to train it using a dataset with many *samples*, we also need the dataset to have many *classes*.

7.2 Future work

There are many ways one could pursue the present work. Obviously, all the possible avenues we presented in the conclusions of the diverse chapters are valid ways to pursue the work at a lower level, we will not repeat them here. What we will do instead, is presenting higher-level research questions. Two major avenues might be interesting both for TMD, and for the types of problem practitioners usually deal with:

7.2.1 Semi, self, or unsupervised learning

In this thesis, all the neural networks were trained using a set of labelled examples, and the network had to learn to classify all of them. This process is called *supervised learning*, for the labels directly tell the network

what to learn. However, in practice, data is often cheap to record (the researcher only needs to passively leave the sensor running); while the labels are harder to obtain: usually, labelling requires the intervention of one or more human annotator(s) who will tell which class corresponds to each sample. To avoid it, several lines of work try to make use of unlabelled data, for example using auto-encoders which learn how the data look like by trying to compress it ([125, 65]). These techniques are said to be *semi-supervised*, for the network is trained by taking into account the unlabeled samples (for example, with an auto-encoder), and learning to classify the labelled ones with a supervised criterion. The GeoLife dataset provides a convenient dataset with a portion of unlabelled data, to experiment with such approaches.

Recent improvements of semi-supervised algorithms involve more than trying to compress the unlabeled samples. In these works, the network learns to solve a *pretext task*, a problem which is not directly of any use for the classification but forces it to learn meaningful features along the way. For instance, the network can be asked to guess if two patches come from the same image, or if one image is the reflection of the other. This is called *self-supervised* classification, because we still use a label to train the network, but the label can be extracted from the raw data itself without the intervention of an annotator [10]. Alternatively, we could also pursue the *unsupervised* avenue, where most of the learning process involves no label at all (such as the unsupervised clustering in [55]).

7.2.2 Domain Adaptation

Domain adaptation is simply the activity that consists of learning to use data from other sources, where the data or labels are easier to obtain [301, 302]. This "other source" can be an external dataset, or a dataset generated using a simulator. We already explained that the design of a simulator for TMD or even HAR is an additional technological barrier to overcome before applying domain adaptation to such data. However, one can apply domain adaptation techniques immediately by trying to use data from another context. For instance, learning to adapt to one user using data from other users (such as [303] does) can be thought of as domain adaptation. This problem even has its own datasets, for the data from the SHL 2019 and SHL 2020 challenges are still readily available and open for anyone to experiment with.

One subfield of domain adaptation that could be interesting to pursue for practitioners is *few-shot learning* [304]: this type of problem consists in starting from a model trained on a diverse and massive dataset (*e.g.*, ImageNet); and using a handful of samples per class (sometimes 1, sometimes 5, never more than 10) to find ways to adapt this model to the new classification problem. Labelling less than a hundred samples in total would be ideal for practitioners but, to the best of our knowledge, few-shot learning is not explored for TMD or even HAR. As applying few-shot learning to our problems is both unexplored and useful in practice, it is the ideal research problem to tackle for longer-term works.

Convolutional networks mostly took off thanks to Computer Vision and the dataset ImageNet. However, building this dataset was an extremely long process, which involved thousands of hours of tedious work [2]. Computer Vision, in turn, influenced many classification tasks, such as TMD, which still nowadays rely on heavily supervised data. In this regard, Natural Language Processing (NLP) is valuable: from the start, the representations of words were learnt from unsupervised sequences of words [305, 118], and even today, current models are taught using no labels [306]. The existence of NLP shows it is possible to learn efficient, general-purpose representations with little to no human intervention. Let us hope that we can make other domains follow this example.

Bibliography

[1] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989). Number: 4, pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[2] Fei-Fei Li. *How we're teaching computers to understand pictures*. en. 2015. URL: https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures (visited on 06/22/2021).

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". en. In: *Communications of the ACM* 60.6 (May 2017). Number: 6, pp. 84–90. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3065386. URL: https://dl.acm.org/doi/10.1145/3065386 (visited on 07/19/2021).

[4] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". en. In: *International Journal of Computer Vision* 115.3 (Dec. 2015). Number: 3, pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y (visited on 07/19/2021).

[5] Maithra Raghu et al. "Transfusion: Understanding Transfer Learning for Medical Imaging". In: *arXiv:1902.07208 [cs, stat]* (Feb. 2019). URL: http://arxiv.org/abs/1902.07208 (visited on 06/14/2019).

[6] Ali Sharif Razavian et al. "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition". en. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. Columbus, OH, USA: IEEE, June 2014, pp. 512–519. ISBN: 978-1-4799-4308-1. DOI: 10.1109/CVPRW.2014.131. URL: https://ieeexplore.ieee.org/document/6910029 (visited on 07/23/2021).

[7] Jeff Donahue et al. "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition". en. In: *International Conference on Machine Learning*. PMLR, Jan. 2014, pp. 647–655. URL: http://proceedings.mlr.press/v32/donahue14.html (visited on 07/23/2021).

[8] Jie Hu et al. "Squeeze-and-Excitation Networks". en. In: *arXiv:1709.01507 [cs]* (Sept. 2017). URL: http://arxiv.org/abs/1709.01507 (visited on 12/21/2018).

[9] Salman Khan et al. "Transformers in Vision: A Survey". In: *arXiv:2101.01169 [cs]* (Feb. 2021). URL: http://arxiv.org/abs/2101.01169 (visited on 05/12/2021).

[10] Spyros Gydaris et al. "CVPR 2020 Tutorial on Annotation-Efficient Learning: Few-Shot, Self-Supervised, and Incremental Learning Approaches". In: *2020 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. online, 2020. URL: https://annotation-efficient-learning.github.io/ (visited on 07/19/2021).

[11] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[12] Oumayma Sakri et al. "A Multi-User Multi-Task Model For Stress Monitoring From Wearable Sensors". In: *2018 21st International Conference on Information Fusion (FUSION)*. July 2018, pp. 761–766. DOI: 10.23919/ICIF.2018.8455378.

[13] Laurence Casteran et al. "Identification de deux sous-types de dysgraphies à partir de l'analyse de paramètres cinématiques et statiques de l'écriture d'enfants typiques et porteurs d'une dysgraphie". fr. In: *ANAE - Approche Neuropsychologique des Apprentissages Chez L'enfant* (2021). URL: https://hal.univ-grenoble-alpes.fr/hal-03101924 (visited on 08/05/2021).

[14] N Saguin-Sprynski, L Jouanet, and M Billeres. "Monitoring system for cable transportation". In: *International Journal of Condition Monitoring* 9.2 (July 2019). Number: 2, pp. 46–49. DOI: 10.1784/204764219826793785.

[15] Andrea Vassilev. "Data Mining Applied to Transportation Mode Classification Problem:" en. In: *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 36–46. ISBN: 978-989-758-293-6. DOI: 10.5220/0006633300360046. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006633300360046 (visited on 08/02/2021).

[16] Umer Majeed, Sheikh Hassan, and Choong Seon Hong. *Vanilla Split Learning for Transportation Mode Detection using Diverse Smartphone Sensors*. June 2021.

[17] *ActivityRecognitionClient \textbackslashtextbar Google Play services*. en. July 2021. URL: https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionClient?hl=fr (visited on 07/28/2021).

[18] *Activity Recognition API*. en. July 2021. URL: https://developers.google.com/location-context/activity-recognition?hl=fr (visited on 07/29/2021).

[19] Claudia Carpineti et al. "Custom Dual Transportation Mode Detection by Smartphone Devices Exploiting Sensor Diversity". en. In: *arXiv:1810.05596 [cs, stat]* (Oct. 2018). URL: http://arxiv.org/abs/1810.05596 (visited on 01/03/2019).

[20] Timothy Sohn et al. "Mobility Detection Using Everyday GSM Traces". en. In: *UbiComp 2006: Ubiquitous Computing*. Ed. by Paul Dourish and Adrian Friday. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 212–224. ISBN: 978-3-540-39635-2. DOI: 10.1007/11853565_13.

[21] Arash Kalatian and Bilal Farooq. "Mobility Mode Detection Using WiFi Signals". In: *2018 IEEE International Smart Cities Conference (ISC2)*. Sept. 2018, pp. 1–7. DOI: 10.1109/ISC2.2018.8656903.

[22] Vlad C Coroamă, Can Türk, and Friedemann Mattern. "Poster: Exploring the Usefulness of Bluetooth and WiFi Proximity for Transportation Mode Recognition". en. In: (2019), p. 4.

[23] Paulo Ferreira, Andriy Zabolotny, and João Barreto. "Bicycle Mode Activity Detection with Bluetooth Low Energy Beacons". In: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. Sept. 2019, pp. 1–4. DOI: 10.1109/NCA.2019.8935030.

[24] Lin Wang and Daniel Roggen. "SOUND-BASED TRANSPORTATION MODE RECOGNITION WITH SMARTPHONES". en. In: (2019), p. 5.

[25] S. Richoz et al. "Transportation Mode Recognition Fusing Wearable Motion, Sound, and Vision Sensors". In: *IEEE Sensors Journal* 20.16 (Aug. 2020). Number: 16, pp. 9314–9328. ISSN: 1558-1748. DOI: 10.1109/JSEN.2020.2987306.

[26] *GPS.gov: GPS Accuracy*. Sept. 2019. URL: https://www.gps.gov/systems/gps/performance/accuracy/ (visited on 09/26/2019).

[27] Mohammad Etemad. "Transportation Modes Classification Using Feature Engineering". en. In: *arXiv:1807.10876 [cs, stat]* (July 2018). URL: http://arxiv.org/abs/1807.10876 (visited on 12/14/2018).

[28] Sina Dabiri and Kevin Heaslip. "Inferring transportation modes from GPS trajectories using a convolutional neural network". In: *Transportation Research Part C: Emerging Technologies* 86 (Jan. 2018), pp. 360–371. ISSN: 0968-090X. DOI: 10.1016/j.trc.2017.11.021. URL: http://www.sciencedirect.com/science/article/pii/S0968090X17303509 (visited on 12/14/2018).

[29] Lin Liao et al. "Learning and inferring transportation routines". en. In: *Artificial Intelligence* 171.5-6 (Apr. 2007). Number: 5-6, pp. 311–331. ISSN: 00043702. DOI: 10.1016/j.artint.2007.01.006. URL: http://linkinghub.elsevier.com/retrieve/pii/S0004370207000380 (visited on 12/20/2018).

[30] Anindya Das Antar et al. "A Comparative Approach to Classification of Locomotion and Transportation Modes Using Smartphone Sensor Data". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1497–1502. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267516. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267516 (visited on 03/11/2019).

[31] Martin Gjoreski et al. "Applying Multiple Knowledge to Sussex-Huawei Locomotion Challenge". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1488–1496. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267515. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267515 (visited on 02/01/2019).

[32] Ali Akbari et al. "Hierarchical Signal Segmentation and Classification for Accurate Activity Recognition". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1596–1605. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267528. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267528 (visited on 03/11/2019).

[33] Beidi Zhao, Shuai Li, and Yanbo Gao. "IndRNN based long-term temporal recognition in the spatial and frequency domain". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 368–372. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414355. URL: https://dl.acm.org/doi/10.1145/3410530.3414355 (visited on 10/16/2020).

[34] Neoklis Polyzotis et al. "Data Lifecycle Challenges in Production Machine Learning: A Survey". In: *ACM SIGMOD Record* 47.2 (Dec. 2018). Number: 2, pp. 17–28. ISSN: 0163-5808. DOI: 10.1145/3299887.3299891. URL: https://doi.org/10.1145/3299887.3299891 (visited on 07/21/2021).

[35] Yu Zheng et al. "Learning transportation mode from raw gps data for geographic applications on the web". en. In: *Proceeding of the 17th international conference on World Wide Web - WWW '08*. Beijing, China: ACM Press, 2008, p. 247. ISBN: 978-1-60558-085-2. DOI: 10.1145/1367497.1367532. URL: http://portal.acm.org/citation.cfm?doid=1367497.1367532 (visited on 12/14/2018).

[36] Yu Zheng et al. "Understanding transportation modes based on GPS data for web applications". en. In: *ACM Transactions on the Web* 4.1 (Jan. 2010). Number: 1, pp. 1–36. ISSN: 15591131. DOI: 10.1145/1658373.1658374. URL: http://portal.acm.org/citation.cfm?doid=1658373.1658374 (visited on 12/14/2018).

[37] Ali Yazdizadeh, Zachary Patterson, and Bilal Farooq. "Ensemble Convolutional Neural Networks for Mode Inference in Smartphone Travel Survey". en. In: (2019), p. 8.

[38] Yuki Endo et al. "Classifying spatial trajectories using representation learning". en. In: *International Journal of Data Science and Analytics* 2.3-4 (Dec. 2016). Number: 3-4, pp. 107–117. ISSN: 2364-415X, 2364-4168. DOI: 10.1007/s41060-016-0014-1. URL: http://link.springer.com/10.1007/s41060-016-0014-1 (visited on 01/24/2019).

[39] Yida Zhu et al. "DenseNetX and GRU for the sussex-huawei locomotion-transportation recognition challenge". In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. UbiComp-ISWC '20. New York, NY, USA: Association for Computing Machinery, Sept. 2020, pp. 373–377. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414349. URL: https://doi.org/10.1145/3410530.3414349 (visited on 10/13/2020).

[40] Meng-Chieh Yu et al. "Big data small footprint: the design of a low-power classifier for detecting transportation modes". en. In: *Proceedings of the VLDB Endowment* 7.13 (Aug. 2014). Number: 13, pp. 1429–1440. ISSN: 21508097. DOI: 10.14778/2733004.2733015. URL: http://dl.acm.org/citation.cfm?doid=2733004.2733015 (visited on 01/21/2019).

[41] Jian Wu et al. "A Decision Level Fusion and Signal Analysis Technique for Activity Segmentation and Recognition on Smart Phones". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1571–1578. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267525. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267525 (visited on 03/11/2019).

[42] Peter Widhalm, Maximilian Leodolter, and Norbert Brändle. "Top in the Lab, Flop in the Field?: Evaluation of a Sensor-based Travel Activity Classifier with the SHL Dataset". In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*. UbiComp '18. New York, NY, USA: ACM, 2018, pp. 1479–1487. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267514. URL: http://doi.acm.org/10.1145/3267305.3267514 (visited on 01/21/2019).

[43] Björn Friedrich et al. "Transportation mode classification from smartphone sensors via a long-short-term-memory network". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 709–713. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344855. URL: https://dl.acm.org/doi/10.1145/3341162.3344855 (visited on 07/16/2021).

[44] Md Sadman Siraj et al. "UPIC: user and position independent classical approach for locomotion and transportation modes recognition". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 340–345. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414343. URL: https://dl.acm.org/doi/10.1145/3410530.3414343 (visited on 07/16/2021).

[45] Vito Janko et al. "A New Frontier for Activity Recognition: The Sussex-Huawei Locomotion Challenge". In: 2018, pp. 1511–1520. DOI: 10.1145/3267305.3267518.

[46] Masud Ahmed et al. "POIDEN: position and orientation independent deep ensemble network for the classification of locomotion and transportation modes". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 674–679. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3345570. URL: https://dl.acm.org/doi/10.1145/3341162.3345570 (visited on 07/16/2021).

[47] Chan Naseeb and Bilal Al Saeedi. "Activity recognition for locomotion and transportation dataset using deep learning". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 329–334. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414348. URL: https://dl.acm.org/doi/10.1145/3410530.3414348 (visited on 07/16/2021).

[48] B. Alotaibi. "Transportation Mode Detection by Embedded Sensors Based on Ensemble Learning". In: *IEEE Access* 8 (2020), pp. 145552–145563. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3014901.

[49] Anindya Das Antar, Masud Ahmed, and Md Atiqur Rahman Ahad. "Recognition of human locomotion on various transportations fusing smartphone sensors". en. In: *Pattern Recognition Letters* (Apr. 2021). ISSN: 0167-8655. DOI: 10.1016/j.patrec.2021.04.015. URL: https://www.sciencedirect.com/science/article/pii/S0167865521001549 (visited on 05/26/2021).

[50] Chihiro Ito et al. "Application of CNN for Human Activity Recognition with FFT Spectrogram of Acceleration and Gyro Sensors". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1503–1510. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267517. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267517 (visited on 03/11/2019).

[51] Ryoichi Sekiguchi et al. "Ensemble learning for human activity recognition". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 335–339. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414346. URL: https://dl.acm.org/doi/10.1145/3410530.3414346 (visited on 10/13/2020).

[52] Chihiro Ito, Masaki Shuzo, and Eisaku Maeda. "CNN for human activity recognition on small datasets of acceleration and gyro sensors using transfer learning". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 724–729. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344868. URL: https://dl.acm.org/doi/10.1145/3341162.3344868 (visited on 07/16/2021).

[53] Kei Yaguchi et al. "Human activity recognition using multi-input CNN model with FFT spectrograms". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 364–367. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414342. URL: https://dl.acm.org/doi/10.1145/3410530.3414342 (visited on 07/16/2021).

[54] Beidi Zhao et al. "A Framework of Combining Short-Term Spatial/Frequency Feature Extraction and Long-Term IndRNN for Activity Recognition". en. In: *Sensors* 20.23 (Jan. 2020). Number: 23, p. 6984. DOI: 10.3390/s20236984. URL: https://www.mdpi.com/1424-8220/20/23/6984 (visited on 08/05/2021).

[55] Christos Markos and James J.Q. Yu. "Unsupervised Deep Learning for GPS-Based Transportation Mode Identification". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. Sept. 2020, pp. 1–6. DOI: 10.1109/ITSC45102.2020.9294673.

[56] R. Killick, P. Fearnhead, and I. A. Eckley. "Optimal Detection of Changepoints With a Linear Computational Cost". In: *Journal of the American Statistical Association* 107.500 (Dec. 2012). Number: 500, pp. 1590–1598. ISSN: 0162-1459. DOI: 10.1080/01621459.2012.737745. URL: https://doi.org/10.1080/01621459.2012.737745 (visited on 05/22/2019).

[57] Sina Dabiri. "Semi-Supervised Deep Learning Approach for Transportation Mode Identification Using GPS Trajectory Data". en. PhD Thesis. Virginia Polytechnic Institute and State University: Virginia Polytechnic Institute and State University, 2018. URL: https://vtechworks.lib.vt.edu/handle/10919/86845.

[58] Adrian C. Prelipcean, Gyözö Gidofalvi, and Yusak O. Susilo. "Measures of transport mode segmentation of trajectories". en. In: *International Journal of Geographical Information Science* 30.9 (Sept. 2016). Number: 9, pp. 1763–1784. ISSN: 1365-8816, 1362-3087. DOI: 10.1080/13658816.2015.1137297. URL: http://www.tandfonline.com/doi/full/10.1080/13658816.2015.1137297 (visited on 07/12/2021).

[59] Kensaku Akamine et al. "SHL Recognition Challenge: Team TK-2 - Combining Results of Multisize Instances-". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1557–1562. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267523. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267523 (visited on 03/11/2019).

[60] Vito Janko et al. "Cross-location transfer learning for the sussex-huawei locomotion recognition challenge". en. In: *Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers - UbiComp/ISWC '19*. London, United Kingdom: ACM Press, 2019, pp. 730–735. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344856. URL: http://dl.acm.org/citation.cfm?doid=3341162.3344856 (visited on 10/13/2020).

[61] Michael Sloma, Makan Arastuie, and Kevin S. Xu. "Activity Recognition by Classification with Time Stabilization for the SHL Recognition Challenge". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1616–1625. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267530. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267530 (visited on 03/11/2019).

[62] Hitoshi Matsuyama et al. "Short Segment Random Forest with Post Processing Using Label Constraint for SHL Recognition Challenge". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1636–1642. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267532. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267532 (visited on 03/11/2019).

[63] Shuai Li et al. "Smartphone-sensors Based Activity Recognition Using IndRNN". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1541–1547. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267521. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267521 (visited on 03/11/2019).

[64] M. Amac Guvensan et al. "A Novel Segment-Based Approach for Improving Classification Performance of Transport Mode Detection". en. In: *Sensors* 18.1 (Jan. 2018). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 87. ISSN: 1424-8220. DOI: 10.3390/s18010087. URL: https://www.mdpi.com/1424-8220/18/1/87 (visited on 03/02/2022).

[65] Zhishuai Li et al. "A Semi-supervised End-to-end Framework for Transportation Mode Detection by Using GPS-enabled Sensing Devices". In: *IEEE Internet of Things Journal* (2021), pp. 1–1. ISSN: 2327-4662. DOI: 10.1109/JIOT.2021.3115239.

[66] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". en. In: *arXiv:1505.04597 [cs]* (May 2015). URL: http://arxiv.org/abs/1505.04597 (visited on 02/27/2019).

[67] Zhibin Xiao et al. "Identifying Different Transportation Modes from Trajectory Data Using Tree-Based Ensemble Classifiers". en. In: *ISPRS International Journal of Geo-Information* 6.2 (Feb. 2017). Number: 2, p. 57. ISSN: 2220-9964. DOI: 10.3390/ijgi6020057. URL: http://www.mdpi.com/2220-9964/6/2/57 (visited on 12/14/2018).

[68] Swapnil Sayan Saha et al. "Supervised and Neural Classifiers for Locomotion Analysis". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1563–1570. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267524. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267524 (visited on 03/11/2019).

[69] Swapnil Sayan Saha et al. "Position independent activity recognition using shallow neural architecture and empirical modeling". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 808–813. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3345572. URL: https://dl.acm.org/doi/10.1145/3341162.3345572 (visited on 07/16/2021).

[70] Arash Jahangiri and Hesham A. Rakha. "Applying Machine Learning Techniques to Transportation Mode Recognition Using Mobile Phone Sensor Data". In: *IEEE Transactions on Intelligent Transportation Systems* 16.5 (Oct. 2015). Number: 5, pp. 2406–2417. ISSN: 1558-0016. DOI: 10.1109/TITS.2015.2405759.

[71] Heikki Mäenpää, Andrei Lobov, and Jose L. Martinez Lastra. "Travel mode estimation for multi-modal journey planner". In: *Transportation Research Part C: Emerging Technologies* 82 (Sept. 2017), pp. 273–289. ISSN: 0968-090X. DOI: 10.1016/j.trc.2017.06.021. URL: http://www.sciencedirect.com/science/article/pii/S0968090X17301808 (visited on 05/22/2019).

[72] Peter Widhalm et al. "Tackling the SHL recognition challenge with phone position detection and nearest neighbour smoothing". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 359–363. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414344. URL: https://dl.acm.org/doi/10.1145/3410530.3414344 (visited on 07/16/2021).

[73] Peter Widhalm, Maximilian Leodolter, and Norbert Brändle. "Ensemble-based domain adaptation for transport mode recognition with mobile sensors". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 857–861. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344857. URL: https://dl.acm.org/doi/10.1145/3341162.3344857 (visited on 07/16/2021).

[74] Hong Lu et al. "Locomotion recognition using XGBoost and neural network ensemble". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 757–760. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344870. URL: https://dl.acm.org/doi/10.1145/3341162.3344870 (visited on 07/16/2021).

[75] Tarek Bin Zahid and Sadman Ishraq Mohiuddin. "A Fast Resource Efficient Method for Human Action Recognition". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1589–1595. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267527. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267527 (visited on 03/11/2019).

[76] Yugo Nakamura et al. "Multi-Stage Activity Inference for Locomotion and Transportation Analytics of Mobile Users". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1579–1588. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267526. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267526 (visited on 03/11/2019).

[77] Pino Castrogiovanni et al. "Smartphone Data Classification Technique for Detecting the Usage of Public or Private Transportation Modes". In: *IEEE Access* 8 (2020), pp. 58377–58391. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2982218.

[78] Huang qiu et al. "ConvLSTM based Transportation Mode Learning from raw GPS trajectories". en. In: *IET Intelligent Transport Systems* (Feb. 2020). ISSN: 1751-956X, 1751-9578. DOI: 10.1049/iet-its.2019.0017. URL: https://digital-library.theiet.org/content/journals/10.1049/iet-its.2019.0017 (visited on 03/05/2020).

[79] Thomas Kjær Rasmussen et al. "Improved methods to deduct trip legs and mode from travel surveys using wearable GPS devices: A case study from the Greater Copenhagen area". en. In: *Computers, Environment and Urban Systems* 54 (Nov. 2015), pp. 301–313. ISSN: 0198-9715. DOI: 10.1016/j.compenvurbsys.2015.04.001. URL: https://www.sciencedirect.com/science/article/pii/S0198971515000423 (visited on 07/13/2021).

[80] J. Rodríguez-Echeverría, S. Gautama, and D. Ochoa. "A methodology for train trip identification in mobility campaigns based on smart-phones". In: *2017 IEEE First Summer School on Smart Cities (S3C)*. Aug. 2017, pp. 141–144. DOI: 10.1109/S3C.2017.8501397.

[81] X. Zhu et al. "Learning Transportation Annotated Mobility Profiles from GPS Data for Context-Aware Mobile Services". In: *2016 IEEE International Conference on Services Computing (SCC)*. June 2016, pp. 475–482. DOI: 10.1109/SCC.2016.68.

[82] F. Asgari and S. Clemencon. "Transport Mode Detection when Fine-grained and Coarse-grained Data Meet". In: *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. Sept. 2018, pp. 301–307. DOI: 10.1109/ICITE.2018.8492673.

[83] Gulustan Dogan et al. "Where are you?: human activity recognition with smartphone sensor data". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 301–304. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414354. URL: https://dl.acm.org/doi/10.1145/3410530.3414354 (visited on 07/16/2021).

[84] Martin Gjoreski et al. "Classical and deep learning methods for recognizing human activities and modes of transportation with smartphone sensors". en. In: *Information Fusion* 62 (Oct. 2020), pp. 47–62. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2020.04.004. URL: http://www.sciencedirect.com/science/article/pii/S1566253520302566 (visited on 10/14/2020).

[85] Ifigenia Drosouli, Athanasios Voulodimos, and Georgios Miaoulis. "Transportation mode detection using machine learning techniques on mobile phone sensor data". In: *Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA '20. New York, NY, USA: Association for Computing Machinery, June 2020, pp. 1–8. ISBN: 978-1-4503-7773-7. DOI: 10.1145/3389189.3397996. URL: https://doi.org/10.1145/3389189.3397996 (visited on 10/29/2020).

[86] Somayeh Dodge, Robert Weibel, and Ehsan Forootan. "Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects". en. In: *Computers, Environment and Urban Systems* 33.6 (Nov. 2009). Number: 6, pp. 419–434. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2009.07.008. URL: http://linkinghub.elsevier.com/retrieve/pii/S0198971509000556 (visited on 12/14/2018).

[87] Mohsen Rezaie. "Knowledge inference from smartphone GPS data". en. MA thesis. Concordia University, Apr. 2018. URL: https://spectrum.library.concordia.ca/983733/ (visited on 07/19/2019).

[88] Björn Friedrich, Carolin Lübbe, and Andreas Hein. "Analyzing the Importance of Sensors for Mode of Transportation Classification". en. In: *Sensors* 21.1 (Jan. 2021). Number: 1, p. 176. DOI: 10.3390/s21010176. URL: https://www.mdpi.com/1424-8220/21/1/176 (visited on 03/10/2021).

[89] Yu Zheng et al. *GeoLife User Guide*. 2012. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/User20Guide-1.2.pdf (visited on 01/21/2019).

[90] Yi-Ting Tseng et al. "Hierarchical classification using ML/DL for sussex-huawei locomotion-transportation (SHL) recognition challenge". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 346–350. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414347. URL: https://dl.acm.org/doi/10.1145/3410530.3414347 (visited on 07/16/2021).

[91] Stefan Kalabakov et al. "Tackling the SHL challenge 2020 with person-specific classifiers and semi-supervised learning". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 323–328. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414848. URL: https://dl.acm.org/doi/10.1145/3410530.3414848 (visited on 10/13/2020).

[92] Mariana Avezum, Jens Klinker, and Bernd Bruegge. "Transportation Mode Recognition on Multi Modal Routes based on Mobile GPS Data". In: *2019 4th International Conference on Intelligent Transportation Engineering (ICITE)*. Sept. 2019, pp. 141–146. DOI: 10.1109/ICITE.2019.8880187.

[93] Muhammad Awais Shafique and Eiji Hato. "Incorporating MNL Model into Random Forest for Travel Mode Detection". en. In: *Mehran University Research Journal of Engineering and Technology* 40.3 (July 2021). Number: 3, pp. 496–501. ISSN: 2413-7219. DOI: 10.22581/muet1982.2103.04. URL: https://publications.muet.edu.pk/index.php/muetrj/article/view/2156 (visited on 07/07/2021).

[94] Philippe Nitsche et al. "Supporting large-scale travel surveys with smartphones – A practical approach". en. In: *Transportation Research Part C: Emerging Technologies* 43 (June 2014), pp. 212–221. ISSN: 0968090X. DOI: 10.1016/j.trc.2013.11.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0968090X13002325 (visited on 07/23/2019).

[95] Elton F. de S. Soares, Carlos Alberto V. Campos, and Sidney C. de Lucena. "Online travel mode detection method using automated machine learning and feature engineering". In: *Future Generation Computer Systems* 101 (Dec. 2019), pp. 1201–1212. ISSN: 0167-739X. DOI: 10.1016/j.future.2019.07.056. URL: http://www.sciencedirect.com/science/article/pii/S0167739X19305874 (visited on 08/06/2019).

[96] Y. Shen et al. "A Method of Traffic Travel Status Segmentation Based on Position Trajectories". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. Sept. 2015, pp. 2877–2882. DOI: 10.1109/ITSC.2015.462.

[97] Anke Sauerländer-Biebl et al. "Evaluation of a transport mode detection using fuzzy rules". In: *Transportation Research Procedia*. World Conference on Transport Research - WCTR 2016 Shanghai. 10-15 July 2016 25 (Jan. 2017), pp. 591–602. ISSN: 2352-1465. DOI: 10.1016/j.trpro.2017.05.444. URL: http://www.sciencedirect.com/science/article/pii/S2352146517307512 (visited on 07/18/2019).

[98] Rahul Deb Das and Stephan Winter. "A fuzzy logic based transport mode detection framework in urban environment". In: *Journal of Intelligent Transportation Systems* 22.6 (Nov. 2018). Number: 6, pp. 478–489. ISSN: 1547-2450. DOI: 10.1080/15472450.2018.1436968. URL: https://doi.org/10.1080/15472450.2018.1436968 (visited on 07/24/2019).

[99] Yanjun Qin, Chenxing Wang, and Haiyong Luo. "Transportation recognition with the Sussex-Huawei Locomotion challenge". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 798–802. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344862. URL: https://dl.acm.org/doi/10.1145/3341162.3344862 (visited on 07/16/2021).

[100] Sunidhi Brajesh and Indraneel Ray. "Ensemble approach for sensor-based human activity recognition". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 296–300. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414352. URL: https://dl.acm.org/doi/10.1145/3410530.3414352 (visited on 07/16/2021).

[101] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. "Accelerometer-based transportation mode detection on smartphones". en. In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems - SenSys '13*. Roma, Italy: ACM Press, 2013, pp. 1–14. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517367. URL: http://dl.acm.org/citation.cfm?doid=2517351.2517367 (visited on 03/02/2022).

[102] Hao Wang et al. "Detecting Transportation Modes Using Deep Neural Network". en. In: *IEICE Transactions on Information and Systems* E100.D.5 (May 2017). Number: 5, pp. 1132–1135. ISSN: 0916-8532, 1745-1361. DOI: 10.1587/transinf.2016EDL8252. (Visited on 12/14/2018).

[103] Rui Zhang et al. "Classifying transportation mode and speed from trajectory data via deep multi-Scale learning". In: *Computer Networks* 162 (Oct. 2019), p. 106861. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2019.106861. URL: http://www.sciencedirect.com/science/article/pii/S1389128618314397 (visited on 07/22/2019).

[104] Andréa Vassilev. "Reconnaissance des modes de transport par apprentissage profond à partir de signaux GPS". Lille, France, 2019.

[105] Linchao Li et al. "Coupled application of generative adversarial networks and conventional neural networks for travel mode detection using GPS data". en. In: *Transportation Research Part A: Policy and Practice* 136 (June 2020), pp. 282–292. ISSN: 0965-8564. DOI: 10.1016/j.tra.2020.04.005. URL: https://www.sciencedirect.com/science/article/pii/S0965856420305607 (visited on 07/15/2021).

[106] Massinissa Hamidi, Aomar Osmani, and Pegah Alizadeh. "A multi-view architecture for the SHL challenge". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 317–322. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414351. URL: https://dl.acm.org/doi/10.1145/3410530.3414351 (visited on 07/16/2021).

[107] Longfei Zheng et al. "Application of IndRNN for human activity recognition: the Sussex-Huawei locomotion-transportation challenge". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 869–872. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344851. URL: https://dl.acm.org/doi/10.1145/3341162.3344851 (visited on 07/16/2021).

[108] H. Liu and I. Lee. "End-to-end trajectory transportation mode classification using Bi-LSTM recurrent neural network". In: *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. Nov. 2017, pp. 1–5. DOI: 10.1109/ISKE.2017.8258799.

[109] James J. Q. Yu. "Travel Mode Identification With GPS Trajectories Using Wavelet Transform and Deep Learning". In: *IEEE Transactions on Intelligent Transportation Systems* (2019), pp. 1–11. ISSN: 1558-0016. DOI: 10.1109/TITS.2019.2962741.

[110] Arman Golshan et al. "Master thesis in Microdata Analysis". en. In: (2021), p. 31.

[111] Azzam Alwan, Vincent Frey, and Gaël Le Lan. "Orange labs contribution to the Sussex-Huawei locomotion-transportation recognition challenge". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 680–684. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344860. URL: https://dl.acm.org/doi/10.1145/3341162.3344860 (visited on 07/16/2021).

[112] Yida Zhu, Fang Zhao, and Runze Chen. "Applying 1D sensor DenseNet to Sussex-Huawei locomotion-transportation recognition challenge". en. In: *Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers - UbiComp/ISWC '19*. London, United Kingdom: ACM Press, 2019, pp. 873–877. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3345571. URL: http://dl.acm.org/citation.cfm?doid=3341162.3345571 (visited on 10/16/2020).

[113] Yuta Yuki et al. "Activity Recognition Using Dual-ConvLSTM Extracting Local and Global Features for SHL Recognition Challenge". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1643–1651. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267533. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267533 (visited on 03/11/2019).

[114] Aomar Osmani and Massinissa Hamidi. "Hybrid and Convolutional Neural Networks for Locomotion Recognition". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1531–1540. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267520. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267520 (visited on 03/11/2019).

[115] Y. Qin et al. "Toward Transportation Mode Recognition Using Deep Convolutional and Long Short-Term Memory Recurrent Neural Networks". In: *IEEE Access* 7 (2019), pp. 142353–142367. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2944686.

[116] Björn Friedrich, Carolin Lübbe, and Andreas Hein. "Combining LSTM and CNN for mode of transportation classification from smartphone sensors". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 305–310. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414350. URL: https://dl.acm.org/doi/10.1145/3410530.3414350 (visited on 07/16/2021).

[117] Lin Wang et al. "Summary of the sussex-huawei locomotion-transportation recognition challenge 2020". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 351–358. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414341. URL: https://dl.acm.org/doi/10.1145/3410530.3414341 (visited on 10/13/2020).

[118] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv:1301.3781 [cs]* (Jan. 2013). URL: http://arxiv.org/abs/1301.3781 (visited on 05/27/2019).

[119] Jun-Ho Choi and Jong-Seok Lee. "Confidence-based Deep Multimodal Fusion for Activity Recognition". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1548–1556. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267522. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267522 (visited on 03/11/2019).

[120] Jun-Ho Choi and Jong-Seok Lee. "EmbraceNet for activity: a deep multimodal fusion architecture for activity recognition". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 693–698. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344871. URL: https://dl.acm.org/doi/10.1145/3341162.3344871 (visited on 07/16/2021).

[121] Kelvin Xu et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". en. In: *International Conference on Machine Learning*. PMLR, June 2015, pp. 2048–2057. URL: http://proceedings.mlr.press/v37/xuc15.html (visited on 05/20/2021).

[122] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf (visited on 03/11/2020).

[123] Lukas Günthermann, Ivor Simpson, and Daniel Roggen. "Smartphone location identification and transport mode recognition using an ensemble of generative adversarial networks". en. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. Virtual Event Mexico: ACM, Sept. 2020, pp. 311–316. ISBN: 978-1-4503-8076-8. DOI: 10.1145/3410530.3414353. URL: https://dl.acm.org/doi/10.1145/3410530.3414353 (visited on 07/16/2021).

[124] Ali Yazdizadeh, Zachary Patterson, and Bilal Farooq. "Semi-supervised GANs to Infer Travel Modes in GPS Trajectories". en. In: *Journal of Big Data Analytics in Transportation* (July 2021). ISSN: 2523-3564. DOI: 10.1007/s42421-021-00047-y. URL: https://doi.org/10.1007/s42421-021-00047-y (visited on 11/22/2021).

[125] Dmitrijs Balabka. "Semi-supervised learning for human activity recognition using adversarial autoencoders". en. In: *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*. London United Kingdom: ACM, Sept. 2019, pp. 685–688. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344854. URL: https://dl.acm.org/doi/10.1145/3341162.3344854 (visited on 07/16/2021).

[126] Alireza Makhzani et al. "Adversarial Autoencoders". In: *arXiv:1511.05644 [cs]* (May 2016). URL: http://arxiv.org/abs/1511.05644 (visited on 07/16/2021).

[127] James J. Q. Yu. "Semi-supervised deep ensemble learning for travel mode identification". en. In: *Transportation Research Part C: Emerging Technologies* 112 (Mar. 2020), pp. 120–135. ISSN: 0968-090X. DOI: 10.1016/j.trc.2020.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0968090X19309416 (visited on 07/13/2021).

[128] R. Qi Charles et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 77–85. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.16. URL: http://ieeexplore.ieee.org/document/8099499/ (visited on 08/05/2019).

[129] Xiaoyuan Liang and Guiling Wang. "A Convolutional Neural Network for Transportation Mode Detection Based on Smartphone Platform". en. In: *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. Orlando, FL: IEEE, Oct. 2017, pp. 338–342. ISBN: 978-1-5386-2324-4. DOI: 10.1109/MASS.2017.81. URL: http://ieeexplore.ieee.org/document/8108764/ (visited on 03/11/2019).

[130] F. Taia Alaoui et al. "Urban Transportation Mode Detection from Inertial and Barometric Data in Pedestrian Mobility". In: *IEEE Sensors Journal* (2021), pp. 1–1. ISSN: 1558-1748. DOI: 10.1109/JSEN.2021.3065848.

[131] Michael P. J. Camilleri et al. "$VJA\textbackslashtextbackslashdot{G}\textbackslashtextbackslashdot{G}$ – A Thick-Client Smart-Phone Journey Detection Algorithm". In: *arXiv:1908.10725 [cs]* (Aug. 2019). URL: http://arxiv.org/abs/1908.10725 (visited on 09/03/2019).

[132] Minh Hieu Nguyen, Jimmy Armoogum, and Cédric Garcia. "Experiment on mobility survey using smartphone in Hanoi, Vietnam". In: Aug. 2019.

[133] Adam Bako. "Transport Mode Detection and Classification from Smartphone Sensor Data Using Convolutional Neural Networks". en. In: (2020), p. 63.

[134] Filip Biljecki, Hugo Ledoux, and Peter van Oosterom. "Transportation mode-based segmentation and classification of movement trajectories". en. In: *International Journal of Geographical Information Science* 27.2 (Feb. 2013). Number: 2, pp. 385–407. ISSN: 1365-8816, 1362-3087. DOI: 10.1080/13658816.2012.692791. URL: http://www.tandfonline.com/doi/abs/10.1080/13658816.2012.692791 (visited on 07/15/2021).

[135] Philip Sedgwick and Nan Greenwood. "Understanding the Hawthorne effect". en. In: *BMJ* (Sept. 2015), h4672. ISSN: 1756-1833. DOI: 10.1136/bmj.h4672. URL: https://www.bmj.com/lookup/doi/10.1136/bmj.h4672 (visited on 07/15/2021).

[136] Laszlo A. Jeni, Jeffrey F. Cohn, and Fernando De La Torre. "Facing Imbalanced Data–Recommendations for the Use of Performance Metrics". en. In: *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*. Geneva, Switzerland: IEEE, Sept. 2013, pp. 245–251. ISBN: 978-0-7695-5048-0. DOI: 10.1109/ACII.2013.47. URL: http://ieeexplore.ieee.org/document/6681438/ (visited on 07/15/2021).

[137] Paria Sadeghian, Johan Håkansson, and Xiaoyun Zhao. "Review and evaluation of methods in transport mode detection based on GPS tracking data". en. In: *Journal of Traffic and Transportation Engineering (English Edition)* (July 2021). ISSN: 2095-7564. DOI: 10.1016/j.jtte.2021.04.004. URL: https://www.sciencedirect.com/science/article/pii/S2095756421000623 (visited on 07/15/2021).

[138] Yu Zheng et al. "Mining interesting locations and travel sequences from GPS trajectories". en. In: *Proceedings of the 18th international conference on World wide web - WWW '09*. Madrid, Spain: ACM Press, 2009, p. 791. ISBN: 978-1-60558-487-4. DOI: 10.1145/1526709.1526816. URL: http://portal.acm.org/citation.cfm?doid=1526709.1526816 (visited on 07/08/2020).

[139] Yu Zheng et al. "Understanding mobility based on GPS data". en. In: *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp '08*. Seoul, Korea: ACM Press, 2008, p. 312. ISBN: 978-1-60558-136-1. DOI: 10.1145/1409635.1409677. URL: http://portal.acm.org/citation.cfm?doid=1409635.1409677 (visited on 07/08/2020).

[140] Lin Wang et al. "Summary of the Sussex-Huawei Locomotion-Transportation Recognition Challenge". en. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1521–1530. ISBN: 978-1-4503-5966-5. DOI: 10.1145/3267305.3267519. URL: http://dl.acm.org/citation.cfm?doid=3267305.3267519 (visited on 01/21/2019).

[141] Lin Wang et al. "Summary of the Sussex-Huawei locomotion-transportation recognition challenge 2019". en. In: *Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers - UbiComp/ISWC '19*. London, United Kingdom: ACM Press, 2019, pp. 849–856. ISBN: 978-1-4503-6869-8. DOI: 10.1145/3341162.3344872. URL: http://dl.acm.org/citation.cfm?doid=3341162.3344872 (visited on 09/17/2019).

[142] X. Wang et al. "Two-Stream 3-D convNet Fusion for Action Recognition in Videos With Arbitrary Size and Length". In: *IEEE Transactions on Multimedia* 20.3 (Mar. 2018). Number: 3, pp. 634–644. ISSN: 1520-9210. DOI: 10.1109/TMM.2017.2749159.

[143] Mirco Ravanelli and Yoshua Bengio. "Speech and Speaker Recognition from Raw Waveform with SincNet". In: *arXiv:1812.05920 [cs, eess]* (Feb. 2019). URL: http://arxiv.org/abs/1812.05920 (visited on 11/20/2019).

[144] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". en. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 818–833. ISBN: 978-3-319-10590-1. DOI: 10.1007/978-3-319-10590-1_53.

[145] Gulrukh Turabee, Yuan Shen, and Georgina Cosma. "Interpreting the Filters in the First Layer of a Convolutional Neural Network for Sleep Stage Classification". en. In: *Advances in Computational Intelligence Systems*. Ed. by Zhaojie Ju et al. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2020, pp. 142–154. ISBN: 978-3-030-29933-0. DOI: 10.1007/978-3-030-29933-0_12.

[146] Marta C. González, César A. Hidalgo, and Albert-László Barabási. "Understanding individual human mobility patterns". en. In: *Nature* 453.7196 (June 2008). Number: 7196, pp. 779–782. ISSN: 1476-4687. DOI: 10.1038/nature06958. URL: https://www.nature.com/articles/nature06958 (visited on 06/25/2019).

[147] Chaoming Song et al. "Limits of Predictability in Human Mobility". en. In: *Science* 327.5968 (Feb. 2010). Number: 5968, pp. 1018–1021. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1177170. URL: https://science.sciencemag.org/content/327/5968/1018 (visited on 06/25/2019).

[148] Y. Huang et al. "Exploring Individual Travel Patterns Across Private Car Trajectory Data". In: *IEEE Transactions on Intelligent Transportation Systems* 21.12 (Dec. 2020). Number: 12, pp. 5036–5050. ISSN: 1558-0016. DOI: 10.1109/TITS.2019.2948188.

[149] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *arXiv:1212.5701 [cs]* (Dec. 2012). URL: http://arxiv.org/abs/1212.5701 (visited on 03/14/2019).

[150] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. URL: http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html (visited on 05/06/2019).

[151] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". en. In: *Nature* (July 2021), pp. 1–11. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: https://www.nature.com/articles/s41586-021-03819-2 (visited on 07/27/2021).

[152] Jordi Pons et al. "End-to-end learning for music audio tagging at scale". In: *arXiv:1711.02520 [cs, eess]* (June 2018). URL: http://arxiv.org/abs/1711.02520 (visited on 12/13/2019).

[153] Pin Wang, En Fan, and Peng Wang. "Comparative analysis of image classification algorithms based on traditional machine learning and deep learning". en. In: *Pattern Recognition Letters* 141 (Jan. 2021), pp. 61–67. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2020.07.042. URL: https://www.sciencedirect.com/science/article/pii/S0167865520302981 (visited on 10/05/2021).

[154] Lin Shu et al. "A Review of Emotion Recognition Using Physiological Signals". eng. In: *Sensors (Basel, Switzerland)* 18.7 (June 2018). Number: 7. ISSN: 1424-8220. DOI: 10.3390/s18072074.

[155] Samir Khan and Takehisa Yairi. "A review on the application of deep learning in system health management". en. In: *Mechanical Systems and Signal Processing* 107 (July 2018), pp. 241–265. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2017.11.024. URL: http://www.sciencedirect.com/science/article/pii/S0888327017306064 (visited on 11/19/2019).

[156] Oliver Faust et al. "Deep learning for healthcare applications based on physiological signals: A review". In: *Computer Methods and Programs in Biomedicine* 161 (July 2018), pp. 1–13. ISSN: 0169-2607. DOI: 10.1016/j.cmpb.2018.04.005. URL: http://www.sciencedirect.com/science/article/pii/S0169260718301226 (visited on 10/11/2019).

[157] Gen Li et al. "Deep learning for EEG data analytics: A survey". en. In: *Concurrency and Computation: Practice and Experience* 32.18 (2020). Number: 18, e5199. ISSN: 1532-0634. DOI: 10.1002/cpe.5199. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5199 (visited on 09/01/2021).

[158] Manuel J. Rivera et al. "Diagnosis and prognosis of mental disorders by means of EEG and deep learning: a systematic mapping study". en. In: *Artificial Intelligence Review* (Mar. 2021). ISSN: 1573-7462. DOI: 10.1007/s10462-021-09986-y. URL: https://doi.org/10.1007/s10462-021-09986-y (visited on 09/01/2021).

[159] Zhen-Hua Ling et al. "Deep Learning for Acoustic Modeling in Parametric Speech Generation: A systematic review of existing techniques and future trends". In: *IEEE Signal Processing Magazine* 32.3 (May 2015). Number: 3, pp. 35–52. ISSN: 1558-0792. DOI: 10.1109/MSP.2014.2359987.

[160] S. Karpagavalli and Edy Chandra. "A Review on Automatic Speech Recognition Architecture and Approaches". In: 2016. DOI: 10.14257/ijsip.2016.9.4.34.

[161] Tara N. Sainath et al. "Improvements to Deep Convolutional Neural Networks for LVCSR". In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. Dec. 2013, pp. 315–320. DOI: 10.1109/ASRU.2013.6707749.

[162] Haytham Fayek. *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*. en. Apr. 2016. URL: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html (visited on 12/04/2019).

[163] Hendrik Purwins et al. "Deep Learning for Audio Signal Processing". In: *IEEE Journal of Selected Topics in Signal Processing* 13.2 (May 2019). Number: 2, pp. 206–219. ISSN: 1941-0484. DOI: 10.1109/JSTSP.2019.2908700.

[164] Daniel Michelsanti et al. "An Overview of Deep-Learning-Based Audio-Visual Speech Enhancement and Separation". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), pp. 1368–1396. ISSN: 2329-9304. DOI: 10.1109/TASLP.2021.3066303.

[165] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. "Wav2Letter: an End-to-End ConvNet-based Speech Recognition System". In: *arXiv:1609.03193 [cs]* (Sept. 2016). URL: http://arxiv.org/abs/1609.03193 (visited on 01/12/2021).

[166] Dario Bertero and Pascale Fung. "A first look into a Convolutional Neural Network for speech emotion detection". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2017, pp. 5115–5119. DOI: 10.1109/ICASSP.2017.7953131.

[167] William Chan et al. "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2016, pp. 4960–4964. DOI: 10.1109/ICASSP.2016.7472621.

[168] Yu Zhang, William Chan, and Navdeep Jaitly. "Very deep convolutional networks for end-to-end speech recognition". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2017, pp. 4845–4849. DOI: 10.1109/ICASSP.2017.7953077.

[169] Suyoun Kim, Takaaki Hori, and Shinji Watanabe. "Joint CTC-attention based end-to-end speech recognition using multi-task learning". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2017, pp. 4835–4839. DOI: 10.1109/ICASSP.2017.7953075.

[170] Ying Zhang et al. "Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks". In: *arXiv:1701.02720 [cs, stat]* (Jan. 2017). URL: http://arxiv.org/abs/1701.02720 (visited on 11/26/2019).

[171] Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. "Rethinking CNN Models for Audio Classification". In: *arXiv:2007.11154 [cs, eess]* (July 2020). URL: http://arxiv.org/abs/2007.11154 (visited on 10/07/2020).

[172] Han Zhang et al. "Thunder Signal Detection via Deep Learning". en. In: *Journal of Physics: Conference Series* 1828.1 (Feb. 2021). Number: 1, p. 012023. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1828/1/012023. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1828/1/012023 (visited on 04/06/2021).

[173] Keisuke Kinoshita et al. "A summary of the REVERB challenge: state-of-the-art and remaining challenges in reverberant speech processing research". In: *EURASIP Journal on Advances in Signal Processing* 2016.1 (Jan. 2016). Number: 1, p. 7. ISSN: 1687-6180. DOI: 10.1186/s13634-016-0306-6. URL: https://doi.org/10.1186/s13634-016-0306-6 (visited on 11/28/2019).

[174] Shinji Watanabe et al. "CHiME-6 Challenge:Tackling Multispeaker Speech Recognition for Unsegmented Recordings". In: *arXiv:2004.09249 [cs, eess]* (May 2020). URL: http://arxiv.org/abs/2004.09249 (visited on 07/30/2021).

[175] Thomas Pellegrini et al. "The Airbus Air Traffic Control speech recognition 2018 challenge: towards ATC automatic transcription and call sign detection". In: *arXiv:1810.12614 [cs, eess]* (Oct. 2018). URL: http://arxiv.org/abs/1810.12614 (visited on 11/28/2019).

[176] Jon Barker et al. "The fifth 'CHiME' Speech Separation and Recognition Challenge: Dataset, task and baselines". In: *arXiv:1803.10609 [cs, eess]* (Mar. 2018). URL: http://arxiv.org/abs/1803.10609 (visited on 11/27/2019).

[177] Sonal Joshi et al. "CHiME 2018 Workshop: Enhancing beamformed audio using time delay neural network denoising autoencoder". en. In: *CHiME 2018 Workshop on Speech Processing in Everyday Environments*. ISCA, Sept. 2018, pp. 46–48. DOI: 10.21437/CHiME.2018-10. URL: http://www.isca-speech.org/archive/CHiME_2018/abstracts/CHiME_2018_paper_joshi.html (visited on 07/30/2021).

[178] *Results of the sixth CHiME challenge*. en. July 2021. URL: https://chimechallenge.github.io/chime6/results.html (visited on 07/30/2021).

[179] Tianfu Li et al. "WaveletKernelNet: An Interpretable Deep Neural Network for Industrial Intelligent Diagnosis". In: *arXiv:1911.07925 [cs]* (Nov. 2019). URL: http://arxiv.org/abs/1911.07925 (visited on 11/20/2019).

[180] Patrick Nectoux et al. "PRONOSTIA: An experimental platform for bearings accelerated degradation tests." en. In: Denver, Colorado, United States, June 2012, p. 9.

[181] Hai Qiu et al. "Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics". en. In: *Journal of Sound and Vibration* 289.4 (Feb. 2006). Number: 4, pp. 1066–1090. ISSN: 0022-460X. DOI: 10.1016/j.jsv.2005.03.007. URL: http://www.sciencedirect.com/science/article/pii/S0022460X0500221X (visited on 11/18/2019).

[182] Wade A. Smith and Robert B. Randall. "Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study". en. In: *Mechanical Systems and Signal Processing* 64-65 (Dec. 2015), pp. 100–131. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2015.04.021. URL: http://www.sciencedirect.com/science/article/pii/S0888327015002034 (visited on 11/22/2019).

[183] *Turbofan engine degradation simulation data set - Data.gov*. Nov. 2019. URL: https://catalog.data.gov/dataset/turbofan-engine-degradation-simulation-data-set (visited on 11/18/2019).

[184] Ruonan Liu et al. "Artificial intelligence for fault diagnosis of rotating machinery: A review". en. In: *Mechanical Systems and Signal Processing* 108 (Aug. 2018), pp. 33–47. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2018.02.016. URL: http://www.sciencedirect.com/science/article/pii/S0888327018300748 (visited on 11/19/2019).

[185] Jindong Wang et al. "Deep learning for sensor-based activity recognition: A survey". en. In: *Pattern Recognition Letters*. Deep Learning for Pattern Recognition 119 (Mar. 2019), pp. 3–11. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2018.02.010. URL: https://www.sciencedirect.com/science/article/pii/S016786551830045X (visited on 09/02/2021).

[186] Måns Klingspor. *Hilbert Transform : Mathematical Theory and Applications to Signal processing*. eng. 2015. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-122736 (visited on 10/09/2019).

[187] Olivier Janssens et al. "Convolutional Neural Network Based Fault Detection for Rotating Machinery". en. In: *Journal of Sound and Vibration* 377 (Sept. 2016), pp. 331–345. ISSN: 0022-460X. DOI: 10.1016/j.jsv.2016.05.027. URL: http://www.sciencedirect.com/science/article/pii/S0022460X16301638 (visited on 11/19/2019).

[188] David Verstraete et al. "Deep Learning Enabled Fault Diagnosis Using Time-Frequency Image Analysis of Rolling Element Bearings". en. In: *Shock and Vibration* 2017 (2017), pp. 1–17. ISSN: 1070-9622, 1875-9203. DOI: 10.1155/2017/5067651. URL: https://www.hindawi.com/journals/sv/2017/5067651/ (visited on 11/22/2019).

[189] Rui Qiao et al. "A novel deep-learning based framework for multi-subject emotion recognition". In: *2017 4th International Conference on Information, Cybernetics and Computational Social Systems (ICCSS)*. July 2017, pp. 181–185. DOI: 10.1109/ICCSS.2017.8091408.

[190] Robin Tibor Schirrmeister et al. "Deep learning with convolutional neural networks for EEG decoding and visualization". en. In: *Human Brain Mapping* 38.11 (2017). Number: 11, pp. 5391–5420. ISSN: 1097-0193. DOI: 10.1002/hbm.23730. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.23730 (visited on 10/25/2019).

[191] Li Wang et al. "Temporal-spatial-frequency depth extraction of brain-computer interface based on mental tasks". en. In: *Biomedical Signal Processing and Control* 58 (Apr. 2020), p. 101845. ISSN: 1746-8094. DOI: 10.1016/j.bspc.2020.101845. URL: http://www.sciencedirect.com/science/article/pii/S174680942030001X (visited on 02/18/2020).

[192] Mohamed Limam and Frederic Precioso. "Atrial fibrillation detection and ECG classification based on convolutional recurrent neural network". In: *2017 Computing in Cardiology (CinC)*. Sept. 2017, pp. 1–4. DOI: 10.22489/CinC.2017.171-325.

[193] Trieu Hai-Nguyen Le et al. "Feature Extraction Techniques for Automatic Detection of Some Specific Cardiovascular Diseases Using ECG: A Review and Evaluation Study". en. In: *7th International Conference on the Development of Biomedical Engineering in Vietnam (BME7)*. Ed. by Vo Van Toi et al. IFMBE Proceedings. Singapore: Springer, 2020, pp. 543–549. ISBN: 9811358583. DOI: 10.1007/978-981-13-5859-3_94.

[194] Jingshan Huang et al. "ECG Arrhythmia Classification Using STFT-Based Spectrogram and Convolutional Neural Network". In: *IEEE Access* 7 (2019), pp. 92871–92880. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2928017.

[195] Xiang Li et al. "Emotion recognition from multi-channel EEG data through Convolutional Recurrent Neural Network". In: *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. Dec. 2016, pp. 352–359. DOI: 10.1109/BIBM.2016.7822545.

[196] Yannick Roy et al. "Deep learning-based electroencephalography analysis: a systematic review". en. In: *Journal of Neural Engineering* 16.5 (Aug. 2019). Number: 5, p. 051001. ISSN: 1741-2552. DOI: 10.1088/1741-2552/ab260c. URL: https://iopscience.iop.org/article/10.1088/1741-2552/ab260c (visited on 09/01/2021).

[197] Ali Al-Saegh, Shefa A. Dawwd, and Jassim M. Abdul-Jabbar. "Deep learning for motor imagery EEG-based classification: A review". en. In: *Biomedical Signal Processing and Control* 63 (Jan. 2021), p. 102172. ISSN: 1746-8094. DOI: 10.1016/j.bspc.2020.102172. URL: https://www.sciencedirect.com/science/article/pii/S1746809420303116 (visited on 02/11/2021).

[198] F. Lotte et al. "A review of classification algorithms for EEG-based brain–computer interfaces: a 10 year update". en. In: *Journal of Neural Engineering* 15.3 (Apr. 2018). Number: 3, p. 031005. ISSN: 1741-2552. DOI: 10.1088/1741-2552/aab2f2. URL: https://doi.org/10.1088%2F1741-2552%2Faab2f2 (visited on 10/24/2019).

[199] Pouya Bashivan et al. "Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks". In: *ICLR 2016* (Feb. 2016). URL: http://arxiv.org/abs/1511.06448 (visited on 12/03/2019).

[200] Weizheng Qiao and Xiaojun Bi. "Ternary-task convolutional bidirectional neural turing machine for assessment of EEG-based cognitive workload". en. In: *Biomedical Signal Processing and Control* 57 (Mar. 2020), p. 101745. ISSN: 1746-8094. DOI: 10.1016/j.bspc.2019.101745. URL: http://www.sciencedirect.com/science/article/pii/S174680941930326X (visited on 12/02/2019).

[201] Cosmin Stamate, George D. Magoulas, and Michael S. C. Thomas. "Deep Learning Topology–Preserving EEG–Based Images for Autism Detection in Infants". en. In: *Proceedings of the 22nd Engineering Applications of Neural Networks Conference*. Ed. by Lazaros Iliadis et al. Proceedings of the International Neural Networks Society. Cham: Springer International Publishing, 2021, pp. 71–82. ISBN: 978-3-030-80568-5. DOI: 10.1007/978-3-030-80568-5_6.

[202] J. Wang et al. "A Weighted Overlook Graph Representation of EEG Data for Absence Epilepsy Detection". In: *2020 IEEE International Conference on Data Mining (ICDM)*. Nov. 2020, pp. 581–590. DOI: 10.1109/ICDM50108.2020.00067.

[203] Selcan Kaplan Berkaya et al. "A survey on ECG analysis". In: *Biomedical Signal Processing and Control* 43 (May 2018), pp. 216–235. ISSN: 1746-8094. DOI: 10.1016/j.bspc.2018.03.003. URL: http://www.sciencedirect.com/science/article/pii/S1746809418300636 (visited on 10/18/2019).

[204] Shenda Hong et al. "Opportunities and challenges of deep learning methods for electrocardiogram data: A systematic review". en. In: *Computers in Biology and Medicine* 122 (July 2020), p. 103801. ISSN: 0010-4825. DOI: 10.1016/j.compbiomed.2020.103801. URL: https://www.sciencedirect.com/science/article/pii/S0010482520301694 (visited on 08/31/2021).

[205] Zehua Sun et al. "Human Action Recognition from Various Data Modalities: A Review". In: *arXiv:2012.11866 [cs]* (July 2021). URL: http://arxiv.org/abs/2012.11866 (visited on 09/02/2021).

[206] Ming Zeng et al. "Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors". In: Nov. 2014. ISBN: 978-1-63190-024-2. URL: https://eudl.eu/doi/10.4108/icst.mobicase.2014.257786 (visited on 09/27/2019).

[207] Tahmina Zebin, Patricia J Scully, and Krikor B. Ozanyan. "Human activity recognition with inertial sensors using a deep learning approach". In: *2016 IEEE SENSORS*. Oct. 2016, pp. 1–3. DOI: 10.1109/ICSENS.2016.7808590.

[208] Heeryon Cho and Sang Min Yoon. "Divide and Conquer-Based 1D CNN Human Activity Recognition Using Test Data Sharpening". en. In: *Sensors* 18.4 (Apr. 2018). Number: 4, p. 1055. DOI: 10.3390/s18041055. URL: https://www.mdpi.com/1424-8220/18/4/1055 (visited on 10/01/2019).

[209] Connor Daly. "Recognition and Synthesis of Object Transport Motion". In: *arXiv:2009.12967 [cs]* (Sept. 2020). URL: http://arxiv.org/abs/2009.12967 (visited on 09/29/2020).

[210] Nils Y. Hammerla, Shane Halloran, and Thomas Ploetz. "Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables". In: *arXiv:1604.08880 [cs, stat]* (Apr. 2016). URL: http://arxiv.org/abs/1604.08880 (visited on 09/18/2019).

[211] Chi Yoon Jeong and Mooseop Kim. "An Energy-Efficient Method for Human Activity Recognition with Segment-Level Change Detection and Deep Learning". en. In: *Sensors* 19.17 (Jan. 2019). Number: 17, p. 3688. DOI: 10.3390/s19173688. URL: https://www.mdpi.com/1424-8220/19/17/3688 (visited on 09/30/2019).

[212] Mingqi Lv, Wei Xu, and Tieming Chen. "A hybrid deep convolutional and recurrent neural network for complex activity recognition using multimodal sensors". In: *Neurocomputing* 362 (Oct. 2019), pp. 33–40. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2019.06.051. URL: http://www.sciencedirect.com/science/article/pii/S0925231219309361 (visited on 09/02/2019).

[213] Andrey Ignatov. "Real-time human activity recognition from accelerometer data using Convolutional Neural Networks". en. In: *Applied Soft Computing* 62 (Jan. 2018), pp. 915–922. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2017.09.027. URL: https://www.sciencedirect.com/science/article/pii/S1568494617305665 (visited on 09/02/2021).

[214] Fernando Moya Rueda et al. "Convolutional Neural Networks for Human Activity Recognition Using Body-Worn Sensors". en. In: *Informatics* 5.2 (June 2018). Number: 2, p. 26. DOI: 10.3390/informatics5020026. URL: https://www.mdpi.com/2227-9709/5/2/26 (visited on 10/01/2019).

[215] Madhuri Panwar et al. "CNN based approach for activity recognition using a wrist-worn accelerometer". In: *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. July 2017, pp. 2438–2441. DOI: 10.1109/EMBC.2017.8037349.

[216] Francisco Javier Ordóñez and Daniel Roggen. "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition". en. In: *Sensors* 16.1 (Jan. 2016). Number: 1, p. 115. DOI: 10.3390/s16010115. URL: https://www.mdpi.com/1424-8220/16/1/115 (visited on 10/29/2020).

[217] Jen-Yung Tsai et al. "Deep Learning Model to Recognize the Different Progression Condition Patterns of Manual Wheelchair Users for Prevention of Shoulder Pain". en. In: *Advances in Physical Ergonomics and Human Factors*. Ed. by Ravindra S. Goonetilleke and Waldemar Karwowski. Advances in Intelligent Systems and Computing. Springer International Publishing, 2020, pp. 3–13. ISBN: 978-3-030-20142-5.

[218] Taeho Hur et al. "Iss2Image: A Novel Signal-Encoding Technique for CNN-Based Human Activity Recognition". en. In: *Sensors* 18.11 (Nov. 2018). Number: 11, p. 3910. DOI: 10.3390/s18113910. URL: https://www.mdpi.com/1424-8220/18/11/3910 (visited on 10/08/2019).

[219] Mohammad Abu Alsheikh et al. "Deep Activity Recognition Models with Triaxial Accelerometers". en. In: *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*. Mar. 2016. URL: https://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12627 (visited on 09/23/2019).

[220] Wenchao Jiang and Zhaozheng Yin. "Human Activity Recognition Using Wearable Sensors by Deep Convolutional Neural Networks". en. In: *Proceedings of the 23rd ACM international conference on Multimedia - MM '15*. Brisbane, Australia: ACM Press, 2015, pp. 1307–1310. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806333. URL: http://dl.acm.org/citation.cfm?doid=2733373.2806333 (visited on 09/30/2019).

[221] Xiaochen Zheng, Meiqing Wang, and Joaquín Ordieres-Meré. "Comparison of Data Preprocessing Approaches for Applying Deep Learning to Human Activity Recognition in the Context of Industry 4.0". en. In: *Sensors* 18.7 (July 2018). Number: 7, p. 2146. DOI: 10.3390/s18072146. URL: https://www.mdpi.com/1424-8220/18/7/2146 (visited on 10/04/2019).

[222] Jianjie Lu and Kai-Yu Tong. "Robust Single Accelerometer-Based Activity Recognition Using Modified Recurrence Plot". In: *IEEE Sensors Journal* 19.15 (Aug. 2019). Number: 15, pp. 6317–6324. ISSN: 1558-1748. DOI: 10.1109/JSEN.2019.2911204.

[223] Otávio A. B. Penatti and Milton F. S. Santos. "Human activity recognition from mobile inertial sensors using recurrence plots". In: *arXiv:1712.01429 [cs]* (Dec. 2017). URL: http://arxiv.org/abs/1712.01429 (visited on 10/02/2019).

[224] Raphael Memmesheimer, Nick Theisen, and Dietrich Paulus. "Gimme Signals: Discriminative signal encoding for multimodal activity recognition". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2020, pp. 10394–10401. DOI: 10.1109/IROS45743.2020.9341699.

[225] Olasimbo Ayodeji Arigbabu. "Entropy Decision Fusion for Smartphone Sensor based Human Activity Recognition". In: *arXiv:2006.00367 [cs]* (May 2020). URL: http://arxiv.org/abs/2006.00367 (visited on 09/01/2020).

[226] Kavya Sree Gajjala et al. "Comparative Performance Study on Human Activity Recognition with Deep Neural Networks". en. In: *Intelligence Science III*. Ed. by Zhongzhi Shi, Mihir Chakraborty, and Samarjit Kar. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2021, pp. 241–251. ISBN: 978-3-030-74826-5. DOI: 10.1007/978-3-030-74826-5_21.

[227] Y. Chen and Y. Xue. "A Deep Learning Approach to Human Activity Recognition Based on Single Accelerometer". In: *2015 IEEE International Conference on Systems, Man, and Cybernetics*. Oct. 2015, pp. 1488–1492. DOI: 10.1109/SMC.2015.263.

[228] Marjan Gholamrezaii and Seyed Mohammad Taghi Almodarresi. "Human Activity Recognition Using 2D Convolutional Neural Networks". In: *2019 27th Iranian Conference on Electrical Engineering (ICEE)*. Apr. 2019, pp. 1682–1686. DOI: 10.1109/IranianCEE.2019.8786578.

[229] Ramy Hussein et al. "Semi-dilated convolutional neural networks for epileptic seizure prediction". en. In: *Neural Networks* 139 (July 2021), pp. 212–222. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2021.03.008. URL: https://www.sciencedirect.com/science/article/pii/S0893608021000885 (visited on 04/07/2021).

[230] Pete Warden. *How many images do you need to train a neural network?* en. Dec. 2017. URL: https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/ (visited on 08/27/2021).

[231] Saleh Shahinfar, Paul Meek, and Greg Falzon. ""How many images do I need?" Understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring". en. In: *Ecological Informatics* 57 (May 2020), p. 101085. ISSN: 1574-9541. DOI: 10.1016/j.ecoinf.2020.101085. URL: https://www.sciencedirect.com/science/article/pii/S1574954120300352 (visited on 07/26/2021).

[232] Guto Leoni Santos et al. "Accelerometer-Based Human Fall Detection Using Convolutional Neural Networks". en. In: *Sensors* 19.7 (Jan. 2019). Number: 7, p. 1644. DOI: 10.3390/s19071644. URL: https://www.mdpi.com/1424-8220/19/7/1644 (visited on 08/28/2020).

[233] Zhuotun Zhu, Lingxi Xie, and Alan L. Yuille. "Object Recognition with and without Objects". en. In: *arXiv:1611.06596 [cs]* (Nov. 2016). URL: http://arxiv.org/abs/1611.06596 (visited on 02/15/2019).

[234] Terrance DeVries and Graham W. Taylor. "Improved Regularization of Convolutional Neural Networks with Cutout". In: *arXiv:1708.04552 [cs]* (Nov. 2017). URL: http://arxiv.org/abs/1708.04552 (visited on 01/07/2020).

[235] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for interpreting and understanding deep neural networks". en. In: *Digital Signal Processing* 73 (Feb. 2018), pp. 1–15. ISSN: 1051-2004. DOI: 10.1016/j.dsp.2017.10.011. URL: http://www.sciencedirect.com/science/article/pii/S1051200417302385 (visited on 02/18/2020).

[236] Weili Nie, Yang Zhang, and Ankit Patel. "A Theoretical Explanation for Perplexing Behaviors of Backpropagation-based Visualizations". en. In: *International Conference on Machine Learning*. PMLR, July 2018, pp. 3809–3818. URL: http://proceedings.mlr.press/v80/nie18a.html (visited on 08/02/2021).

[237] Beomsu Kim et al. "Why are Saliency Maps Noisy? Cause of and Solution to Noisy Saliency Maps". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. Oct. 2019, pp. 4149–4157. DOI: 10.1109/ICCVW.2019.00510.

[238] Yi-Shan Lin, Wen-Chuan Lee, and Z. Berkay Celik. "What Do You See? Evaluation of Explainable Artificial Intelligence (XAI) Interpretability through Neural Backdoors". In: *arXiv:2009.10639 [cs]* (Sept. 2020). URL: http://arxiv.org/abs/2009.10639 (visited on 09/23/2020).

[239] Suraj Srinivas and Francois Fleuret. "Gradient Alignment in Deep Neural Networks". In: *arXiv:2006.09128 [cs, stat]* (June 2020). URL: http://arxiv.org/abs/2006.09128 (visited on 06/17/2020).

[240] Harshay Shah, Prateek Jain, and Praneeth Netrapalli. "Do Input Gradients Highlight Discriminative Features?" In: *arXiv:2102.12781 [cs, stat]* (Feb. 2021). URL: http://arxiv.org/abs/2102.12781 (visited on 03/10/2021).

[241] David Griffiths and Jan Boehm. "A Review on Deep Learning Techniques for 3D Sensed Data Classification". en. In: *Remote Sensing* 11.12 (Jan. 2019). Number: 12, p. 1499. DOI: 10.3390/rs11121499. URL: https://www.mdpi.com/2072-4292/11/12/1499 (visited on 08/26/2019).

[242] Giorgos Tolias, Ronan Sicre, and Hervé Jégou. "Particular object retrieval with integral max-pooling of CNN activations". In: *arXiv:1511.05879 [cs]* (Nov. 2015). URL: http://arxiv.org/abs/1511.05879 (visited on 04/03/2019).

[243] Shigeng Zhang et al. "Towards Real-time Cooperative Deep Inference over the Cloud and Edge End Devices". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4.2 (June 2020). Number: 2, 69:1–69:24. DOI: 10.1145/3397315. URL: https://doi.org/10.1145/3397315 (visited on 12/10/2020).

[244] Vladislav Sovrasov. *sovrasov/flops-counter.pytorch*. Jan. 2020. URL: https://github.com/sovrasov/flops-counter.pytorch (visited on 01/22/2020).

[245] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and \textbackslashtextless0.5MB model size". In: *arXiv:1602.07360 [cs]* (Nov. 2016). URL: http://arxiv.org/abs/1602.07360 (visited on 02/02/2021).

[246] Biao Wang et al. "Deep separable convolutional network for remaining useful life prediction of machinery". en. In: *Mechanical Systems and Signal Processing* 134 (Dec. 2019), p. 106330. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2019.106330. URL: http://www.sciencedirect.com/science/article/pii/S0888327019305515 (visited on 11/05/2019).

[247] Zhen Qin et al. "Imaging and fusing time series for wearable sensor-based human activity recognition". In: *Information Fusion* 53 (Jan. 2020), pp. 80–87. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2019.06.014. URL: http://www.sciencedirect.com/science/article/pii/S1566253519302180 (visited on 09/03/2019).

[248] Saurabh Gupta et al. "Learning Rich Features from RGB-D Images for Object Detection and Segmentation". en. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 345–360. ISBN: 978-3-319-10584-0. DOI: 10.1007/978-3-319-10584-0_23.

[249] Javed Imran and Balasubramanian Raman. "Evaluating fusion of RGB-D and inertial sensors for multimodal human action recognition". en. In: *Journal of Ambient Intelligence and Humanized Computing* 11.1 (Jan. 2020). Number: 1, pp. 189–208. ISSN: 1868-5145. DOI: 10.1007/s12652-019-01239-9. URL: https://doi.org/10.1007/s12652-019-01239-9 (visited on 03/09/2020).

[250] Lin Ma et al. "Multimodal Convolutional Neural Networks for Matching Image and Sentence". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2623–2631. URL: https://openaccess.thecvf.com/content_iccv_2015/html/Ma_Multimodal_Convolutional_Neural_ICCV_2015_paper.html (visited on 10/12/2020).

[251] Mustafa Sercan Amac et al. "Procedural Reasoning Networks for Understanding Multimodal Procedures". In: *arXiv:1909.08859 [cs]* (Sept. 2019). URL: http://arxiv.org/abs/1909.08859 (visited on 09/25/2019).

[252] Di Feng et al. "Leveraging Uncertainties for Deep Multi-modal Object Detection in Autonomous Driving". In: *arXiv:2002.00216 [cs]* (Feb. 2020). URL: http://arxiv.org/abs/2002.00216 (visited on 02/10/2020).

[253] Jiquan Ngiam et al. "Multimodal Deep Learning". en. In: *ICML*. Jan. 2011. (Visited on 10/12/2020).

[254] Kihyuk Sohn, Wenling Shang, and Honglak Lee. "Improved Multimodal Deep Learning with Variation of Information". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2141–2149. URL: http://papers.nips.cc/paper/5279-improved-multimodal-deep-learning-with (visited on 06/17/2020).

[255] Yuqi Li et al. "IVFuseNet: Fusion of infrared and visible light images for depth prediction". en. In: *Information Fusion* 58 (June 2020), pp. 1–12. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2019.12.014. URL: http://www.sciencedirect.com/science/article/pii/S1566253519301034 (visited on 03/11/2020).

[256] Weiyao Wang, Du Tran, and Matt Feiszli. "What Makes Training Multi-Modal Networks Hard?" In: *arXiv:1905.12681 [cs]* (May 2019). URL: http://arxiv.org/abs/1905.12681 (visited on 06/24/2019).

[257] Changhao Chen et al. "Selective Sensor Fusion for Neural Visual-Inertial Odometry". en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 10534–10543. ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.01079. URL: https://ieeexplore.ieee.org/document/8954338/ (visited on 03/12/2020).

[258] Kuan Liu et al. "Learn to Combine Modalities in Multimodal Deep Learning". In: *arXiv:1805.11730 [cs, stat]* (May 2018). URL: http://arxiv.org/abs/1805.11730 (visited on 04/05/2019).

[259] Jing Gao et al. "A Survey on Deep Learning for Multimodal Data Fusion". en. In: *Neural Computation* 32.5 (May 2020). Number: 5, pp. 829–864. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco_a_01273. URL: https://www.mitpressjournals.org/doi/abs/10.1162/neco_a_01273 (visited on 07/02/2020).

[260] Jia Liu et al. "Urban big data fusion based on deep learning: An overview". In: *Information Fusion* 53 (Jan. 2020), pp. 123–133. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2019.06.016. URL: http://www.sciencedirect.com/science/article/pii/S1566253519301393 (visited on 09/05/2019).

[261] Paul Liang. *Reading List for Topics in Multimodal Machine Learning*. Sept. 2021. URL: https://github.com/pliang279/awesome-multimodal-ml (visited on 09/22/2021).

[262] Bolei Zhou et al. "Learning Deep Features for Discriminative Localization". In: 2016, pp. 2921–2929. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Zhou_Learning_Deep_Features_CVPR_2016_paper.html (visited on 11/14/2019).

[263] Bolei Zhou et al. "OBJECT DETECTORS EMERGE IN DEEP SCENE CNN S". In: 2015.

[264] Xinxin Han et al. "The Effect of Axis-Wise Triaxial Acceleration Data Fusion in CNN-Based Human Activity Recognition". en. In: *IEICE Transactions on Information and Systems* E103.D.4 (Apr. 2020). Number: 4, pp. 813–824. ISSN: 0916-8532, 1745-1361. DOI: 10.1587/transinf.2018EDP7409. URL: https://www.jstage.jst.go.jp/article/transinf/E103.D/4/E103.D_2018EDP7409/_article (visited on 10/23/2020).

[265] Joao Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: 2017, pp. 6299–6308. URL: http://openaccess.thecvf.com/content_cvpr_2017/html/Carreira_Quo_Vadis_Action_CVPR_2017_paper.html (visited on 03/11/2020).

[266] Harold Hotelling. "Relations Between Two Sets of Variates". en. In: *Breakthroughs in Statistics: Methodology and Distribution*. Ed. by Samuel Kotz and Norman L. Johnson. Springer Series in Statistics. New York, NY: Springer, 1992, pp. 162–190. ISBN: 978-1-4612-4380-9. DOI: 10.1007/978-1-4612-4380-9_14. URL: https://doi.org/10.1007/978-1-4612-4380-9_14 (visited on 07/05/2021).

[267] Zeeshan Ahmad and Naimul Khan. "Human Action Recognition Using Deep Multilevel Multimodal (M^2) Fusion of Depth and Inertial Sensors". In: *IEEE Sensors Journal* 20.3 (Feb. 2020). Number: 3, pp. 1445–1455. ISSN: 2379-9153. DOI: 10.1109/JSEN.2019.2947446.

[268] J. R. KETTENRING. "Canonical analysis of several sets of variables". In: *Biometrika* 58.3 (Dec. 1971). Number: 3, pp. 433–451. ISSN: 0006-3444. DOI: 10.1093/biomet/58.3.433. URL: https://doi.org/10.1093/biomet/58.3.433 (visited on 07/05/2021).

[269] Maithra Raghu et al. "SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 6076–6085. URL: http://papers.nips.cc/paper/7188-svcca-singular-vector-canonical-correlation-analysis-for-deep-learning-dynamics-and-interpretability.pdf (visited on 11/05/2019).

[270] Galen Andrew et al. "Deep Canonical Correlation Analysis". en. In: *Proceedings of the 30th International Conference on Machine Learning*. PMLR, May 2013, pp. 1247–1255. URL: https://proceedings.mlr.press/v28/andrew13.html (visited on 09/23/2021).

[271] Nicolle M. Correa et al. "Canonical Correlation Analysis for Data Fusion and Group Inferences". In: *IEEE Signal Processing Magazine* 27.4 (2010). Number: 4, pp. 39–50. ISSN: 1558-0792. DOI: 10.1109/MSP.2010.936725.

[272] "Canonical Correlation Analysis". en. In: *Applied Multivariate Statistical Analysis*. Ed. by Wolfgang Härdle and Léopold Simar. Berlin, Heidelberg: Springer, 2007, pp. 321–330. ISBN: 978-3-540-72244-1. DOI: 10.1007/978-3-540-72244-1_14. URL: https://doi.org/10.1007/978-3-540-72244-1_14 (visited on 04/19/2021).

[273] S. Akaho. "A kernel method for canonical correlation analysis". In: *ArXiv* (2006).

[274] Ari Morcos, Maithra Raghu, and Samy Bengio. "Insights on representational similarity in neural networks with canonical correlation". en. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 5727–5736. URL: https://papers.nips.cc/paper/2018/hash/a7a3d70c6d17a73140918996d03c014f-Abstract.html (visited on 12/18/2020).

[275] M. Haghighat, M. Abdel-Mottaleb, and W. Alhalabi. "Discriminant correlation analysis for feature level fusion with application to multimodal biometrics". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2016, pp. 1866–1870. DOI: 10.1109/ICASSP.2016.7472000.

[276] Quan-Sen Sun et al. "Feature fusion method based on canonical correlation analysis and handwritten character recognition". In: *ICARCV 2004 8th Control, Automation, Robotics and Vision Conference, 2004.* Vol. 2. Dec. 2004, 1547–1552 Vol. 2. DOI: 10.1109/ICARCV.2004.1469081.

[277] Mustafa Zuhaer Nayef Al-Dabagh et al. "Face Recognition System Based on Fusion Features of Local Methods Using CCA". In: *2020 8th International Electrical Engineering Congress (iEECON)*. Mar. 2020, pp. 1–4. DOI: 10.1109/iEECON48109.2020.229489.

[278] N. S. Lakshmiprabha. "Fusing Face and Periocular biometrics using Canonical correlation analysis". In: *arXiv:1604.01683 [cs]* (Mar. 2016). URL: http://arxiv.org/abs/1604.01683 (visited on 06/22/2021).

[279] Tahsina Farah Sanam and Hana Godrich. "FuseLoc: A CCA Based Information Fusion for Indoor Localization Using CSI Phase and Amplitude of Wifi Signals". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2019, pp. 7565–7569. DOI: 10.1109/ICASSP.2019.8683316.

[280] Geoffrey Roeder, Luke Metz, and Diederik P. Kingma. "On Linear Identifiability of Learned Representations". In: *arXiv:2007.00810 [cs, stat]* (July 2020). URL: http://arxiv.org/abs/2007.00810 (visited on 02/22/2021).

[281] David McNeely-White et al. "Exploring the Interchangeability of CNN Embedding Spaces". In: *arXiv:2010.02323 [cs]* (Feb. 2021). URL: http://arxiv.org/abs/2010.02323 (visited on 02/22/2021).

[282] David McNeely-White, J. Ross Beveridge, and Bruce A. Draper. "Inception and ResNet features are (almost) equivalent". en. In: *Cognitive Systems Research* 59 (Jan. 2020), pp. 312–318. ISSN: 1389-0417. DOI: 10.1016/j.cogsys.2019.10.004. URL: https://www.sciencedirect.com/science/article/pii/S1389041719305066 (visited on 05/06/2021).

[283] Horia Mania et al. "Model Similarity Mitigates Test Set Overuse". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/48237d9f2dea8c74c2a72126cf63d933-Paper.pdf.

[284] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". en. In: (), p. 8.

[285] *3.2.2 ResNet_ Cifar10 - PyTorch Tutorial*. Dec. 2020. URL: https://pytorch-tutorial.readthedocs.io/en/latest/tutorial/chapter03_intermediate/3_2_2_cnn_resnet_cifar10/ (visited on 12/17/2020).

[286] Ryo Kamoi and Kei Kobayashi. "Why is the Mahalanobis Distance Effective for Anomaly Detection?" In: *arXiv:2003.00402 [cs, stat]* (Feb. 2020). URL: http://arxiv.org/abs/2003.00402 (visited on 03/05/2020).

[287] I. Garg, P. Panda, and K. Roy. "A Low Effort Approach to Structured CNN Design Using PCA". In: *IEEE Access* 8 (2020), pp. 1347–1360. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2961960.

[288] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". en. In: Sept. 2018. URL: https://openreview.net/forum?id=rJl-b3RcF7&noteId=ryemP68v2m (visited on 12/18/2020).

[289] Alan Edelman, Tomás A. Arias, and Steven T. Smith. "The Geometry of Algorithms with Orthogonality Constraints". en. In: *SIAM Journal on Matrix Analysis and Applications* 20.2 (Jan. 1998). Number: 2, pp. 303–353. ISSN: 0895-4798, 1095-7162. DOI: 10.1137/S0895479895290954. URL: http://epubs.siam.org/doi/10.1137/S0895479895290954 (visited on 10/09/2020).

[290] Xichen Sun and Qiansheng Cheng. "On Subspace Distance". en. In: *Image Analysis and Recognition*. Ed. by Aurélio Campilho and Mohamed Kamel. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 81–89. ISBN: 978-3-540-44896-9. DOI: 10.1007/11867661_8.

[291] Liwei Wang, Xiao Wang, and Jufu Feng. "Subspace distance analysis with application to adaptive Bayesian algorithm for face recognition". en. In: *Pattern Recognition* 39.3 (Mar. 2006). Number: 3, pp. 456–464. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2005.08.015. URL: http://www.sciencedirect.com/science/article/pii/S003132030500381X (visited on 08/28/2020).

[292] Ziwei Ji and Matus Telgarsky. "Gradient descent aligns the layers of deep linear networks". In: *arXiv:1810.02032 [cs, math, stat]* (Feb. 2019). URL: http://arxiv.org/abs/1810.02032 (visited on 09/13/2021).

[293] Jindong Gu and Volker Tresp. "Semantics for Global and Local Interpretation of Deep Neural Networks". In: *arXiv:1910.09085 [cs]* (Oct. 2019). URL: http://arxiv.org/abs/1910.09085 (visited on 11/05/2019).

[294] Bingyi Cao, Andre Araujo, and Jack Sim. "Unifying Deep Local and Global Features for Image Search". In: *arXiv:2001.05027 [cs]* (Sept. 2020). URL: http://arxiv.org/abs/2001.05027 (visited on 07/02/2021).

[295] Artem Babenko and Victor Lempitsky. "Aggregating Deep Convolutional Features for Image Retrieval". In: *arXiv:1510.07493 [cs]* (Oct. 2015). URL: http://arxiv.org/abs/1510.07493 (visited on 07/02/2021).

[296] Ruy Luiz Milidiú and Luis Felipe Müller. "SeismoFlow – Data augmentation for the class imbalance problem". In: *arXiv:2007.12229 [cs]* (Sept. 2020). URL: http://arxiv.org/abs/2007.12229 (visited on 09/03/2020).

[297] Mohammad Malekzadeh et al. "Protecting Sensory Data against Sensitive Inferences". en. In: *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems - W-P2DS'18* (2018), pp. 1–6. DOI: 10.1145/3195258.3195260. URL: http://arxiv.org/abs/1802.07802 (visited on 01/21/2019).

[298] J Nixon. "Intervertebral Disc Mechanics: A Review". en. In: *Journal of the Royal Society of Medicine* 79.2 (Feb. 1986). Publisher: SAGE Publications, pp. 100–104. ISSN: 0141-0768. DOI: 10.1177/014107688607900211. URL: https://doi.org/10.1177/014107688607900211 (visited on 01/29/2022).

[299] Simon Kornblith et al. "Similarity of Neural Network Representations Revisited". en. In: *International Conference on Machine Learning*. PMLR, May 2019, pp. 3519–3529. URL: http://proceedings.mlr.press/v97/kornblith19a.html (visited on 12/18/2020).

[300] Huaxiu Yao et al. "Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019). Number: 01, pp. 5668–5675. ISSN: 2374-3468. DOI: 10.1609/aaai.v33i01.33015668. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4511 (visited on 06/03/2021).

[301] Cheng Ouyang et al. "Data Efficient Unsupervised Domain Adaptation for Cross-Modality Image Segmentation". In: *arXiv:1907.02766 [cs, eess]* (July 2019). URL: http://arxiv.org/abs/1907.02766 (visited on 07/08/2019).

[302] Sagie Benaim et al. "Domain Intersection and Domain Difference". In: *arXiv:1908.11628 [cs]* (Aug. 2019). URL: http://arxiv.org/abs/1908.11628 (visited on 09/02/2019).

[303] S. Matsui et al. "User adaptation of convolutional neural network for human activity recognition". In: *2017 25th European Signal Processing Conference (EUSIPCO)*. Aug. 2017, pp. 753–757. DOI: 10.23919/EUSIPCO.2017.8081308.

[304] Yaqing Wang et al. "Generalizing from a Few Examples: A Survey on Few-shot Learning". en. In: *ACM Computing Surveys* 53.3 (July 2020). Number: 3, pp. 1–34. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3386252. URL: https://dl.acm.org/doi/10.1145/3386252 (visited on 09/17/2020).

[305] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global Vectors for Word Representation". en. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: http://aclweb.org/anthology/D14-1162 (visited on 07/01/2021).

[306] Matthew E. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://aclanthology.org/N18-1202 (visited on 10/15/2021).

[307] Amirata Ghorbani and James Zou. "Neuron Shapley: Discovering the Responsible Neurons". In: *arXiv:2002.09815 [cs, stat]* (Feb. 2020). URL: http://arxiv.org/abs/2002.09815 (visited on 03/04/2020).

[308] Marc Claesen and Bart De Moor. "Hyperparameter Search in Machine Learning". In: *arXiv:1502.02127 [cs, stat]* (Feb. 2015). URL: http://arxiv.org/abs/1502.02127 (visited on 03/05/2019).

[309] Li Yang and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice". en. In: *Neurocomputing* 415 (Nov. 2020), pp. 295–316. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2020.07.061. URL: https://www.sciencedirect.com/science/article/pii/S0925231220311693 (visited on 09/17/2021).

[310] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey". In: *The Journal of Machine Learning Research* 20.1 (2019). Number: 1, pp. 1997–2017.

[311] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html (visited on 09/17/2021).

[312] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13.Feb (2012). Number: Feb, pp. 281–305. ISSN: ISSN 1533-7928. URL: http://www.jmlr.org/papers/v13/bergstra12a.html (visited on 07/08/2019).

[313] A. Caduff et al. "Exploring the Limits of Vanilla CNN Architectures for Fine-Grained Vision-Based Vehicle Classification". en. In: *Proceedings of the 22nd Engineering Applications of Neural Networks Conference*. Ed. by Lazaros Iliadis et al. Proceedings of the International Neural Networks Society. Cham: Springer International Publishing, 2021, pp. 202–212. ISBN: 978-3-030-80568-5. DOI: 10.1007/978-3-030-80568-5_17.

[314] Ilja Manakov, Markus Rohm, and Volker Tresp. "Walking the Tightrope: An Investigation of the Convolutional Autoencoder Bottleneck". In: *arXiv:1911.07460 [cs]* (May 2020). URL: http://arxiv.org/abs/1911.07460 (visited on 10/07/2020).

[315] fei fei li fei fei. *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition*. July 2021. URL: http://cs231n.stanford.edu/ (visited on 07/21/2021).

[316] Mara Graziani, Henning Muller, and Vincent Andrearczyk. "Interpreting Intentionally Flawed Models with Linear Probes". en. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 743–747. ISBN: 978-1-72815-023-9. DOI: 10.1109/ICCVW.2019.00096. URL: https://ieeexplore.ieee.org/document/9022025/ (visited on 10/08/2020).

[317] Sanjeev Arora et al. "Implicit Regularization in Deep Matrix Factorization". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 7413–7424. URL: http://papers.nips.cc/paper/8960-implicit-regularization-in-deep-matrix-factorization.pdf (visited on 10/23/2020).

[318] Sanjeev Arora et al. "A Convergence Analysis of Gradient Descent for Deep Linear Neural Networks". In: *arXiv:1810.02281 [cs, stat]* (Oct. 2019). URL: http://arxiv.org/abs/1810.02281 (visited on 10/23/2020).

[319] Thomio Watanabe and Denis F. Wolf. "Image classification in frequency domain with 2SReLU: a second harmonics superposition activation function". In: *arXiv:2006.10853 [cs]* (June 2020). URL: http://arxiv.org/abs/2006.10853 (visited on 06/22/2020).

[320] Guillaume Alain, Nicolas Le Roux, and Pierre-Antoine Manzagol. "Negative eigenvalues of the Hessian in deep neural networks". In: *arXiv:1902.02366 [cs, math, stat]* (Feb. 2019). URL: http://arxiv.org/abs/1902.02366 (visited on 11/26/2019).

[321] Anna Choromanska et al. "The Loss Surfaces of Multilayer Networks". en. In: (), p. 13.

Appendix A

Methodology: Computing a F1 score from an Intersection over Union

In this section, we will demonstrate how we computed a F1-score in table 2.4 from the per-class Intersection over Union (IoU) in [65].

Firstly, let us consider one class, and work with per-class scores. TP is the number of true positives, FP the number of false positives (the points that were classified as this class but without actually belonging to it) and FN the number of false negatives (points that the classifier did not detect as belonging to the class we considered). The Intersection over Union (IoU) is a score that compares the prediction of the network ($TP + FP$), to the number of data points (pixels or, in our case, time steps, $TP + FN$). The number of samples in the intersection of these two sets is TP, while the number of samples in their union is $TP + FN + FP$. Hence,

$$IoU = \frac{TP}{TP+FN+FP}$$

The F1 score is computed by taking the per-class recall and precision:

$$F1 = \frac{1}{\frac{1}{2}(\frac{1}{Recall} + \frac{1}{Precision})}$$

Knowing that :

$$Recall = \frac{TP}{TP + FN}$$
$$Precision = \frac{TP}{TP + FP}$$

We obtain:

$$\begin{aligned} F1 &= \frac{2}{(\frac{TP+FN}{TP} + \frac{TP+FP}{TP})} \\ &= \frac{2.TP}{TP + FN + TP + FP} \\ &= \frac{2TP}{2TP + FN + FP} \end{aligned}$$

If we call the error $E = FP + FN$

$$F1 = \frac{TP}{TP + E/2}$$
$$IoU = \frac{TP}{TP + E}$$

The last two lines show us that the IoU score is harsher than the F1 score. However, it is not enough yet to know by how much. To find one from the other, simply say that:

$$
\begin{aligned}
IoU &= \frac{TP}{TP+E} \\
\frac{TP}{2} &= IoU\frac{TP+E}{2} \\
TP &= IoU\frac{TP+E}{2} + \frac{TP}{2} \\
&= IoU\frac{TP+E}{2} + \frac{TP}{2}(IoU + (1-IoU)) \\
TP &= IoU(TP + \frac{E}{2}) + \frac{TP}{2}(1-IoU) \\
\frac{TP}{TP+\frac{E}{2}} &= IoU + \frac{\frac{TP}{2}(1-IoU)}{TP+\frac{E}{2}} \\
F1 &= IoU + \frac{F1}{2}(1-IoU) \\
F1(1-\frac{1}{2}+\frac{IoU}{2}) &= IoU \\
F1\frac{1}{2}(1+IoU) &= IoU \\
F1 &= \frac{2IoU}{1+IoU}
\end{aligned}
$$

We obtain the F1 score per class. We only have to compute the mean across classes to get the average F1.

Appendix B

Automatic sensor selection

Some of the fusion methods we presented in chapter 5 allow us to explicitly assign a weight to each of the sensors used. We wondered if these methods could be used to let a neural network automatically select the optimal sensor combination among all the sensors we had. The idea would be to give all possible signals to a network for classification and to look at which sensors were chosen at the end of the training.

B.1 Data fusion for sensor selection

Each fusion method has a different way of assigning importance to a sensor. The general idea is that we obtain a positive value for each sensor which is higher if the sensor is important.

- *bottleneck filters*: The first layer of the network makes an explicit combination of spectrograms. If a coefficient is close to zero, this will mean the network ignores the corresponding sensor. There is no limit, however, to the values each coefficient can take. In particular, the network can use negative values, which still allows conveying information from the sensor. We record the absolute value of the weight to assess the importance of each sensor
- *attention*: For each sensor, the attention assigns one weight (between 0 and 1) per input pixel and for each sample. We simply average the values of these weights over the number of pixels and the number of samples, to obtain one value between 0 and 1 per sensor.
- *selective fusion*: Similarly to attention, we compute one average per sensor.
- *weighted probabilities fusion*: The network explicitly computes a single weight per sensor, we simply record this weight.
- *Gradient blend*: Similarly, we use the weight computed by gradient blend as-is. Even though these weights do not indicate which sensor is useful (the weights are computed to reduce overfitting), but we still check if they have meaning nonetheless.
- *weighted scores fusion*: Contrary to the weighted probabilities, one weight is not enough to know which sensor impacts most of the decision. The reason for this is that scores can be systematically higher for one sensor. To account for scores variation, we *multiply* each weight (between 0 and 1) by the standard deviation of all the scores returned by each sensor.

B.2 Three known scenarios

Before giving all the sensors to the network, we want to make sure that the networks are able to correctly evaluate the sensors when the usefulness of these sensors is already known (from the experiments in the previous). We evaluate three scenarios:

- *Accelerometer + Gyrometer*: As the gyrometer brings a small increase to the performance of the model when added to the accelerometer (about 2 percentage points), we expect the networks to favour the accelerometer most, while still listening at the gyrometer to some extent.
- *Accelerometer + Norm of the Orientation*: The norm of the orientation is almost useless, so we expect the network to only listen to the accelerometer.
- *Four times the accelerometer*: the network needs to select the best sensor, and should ignore the redundancies. To test the network's ability to ignore redundant information, we create a scenario with four sensors, and each of these sensors is a copy of the norm of the accelerometer. Ideally, the network should affect a weight of zero to all but one of the sensors each time. *Note*: if the network chooses to listen to a single accelerometer, but this accelerometer is not the same between initializations (*eg* the network listens at the first accelerometer during the first run, and to the last accelerometer the next run), the average weight per accelerometer will not show that a single accelerometer was selected each time. In order to avoid this situation, and instead of identifying the weights by their initial order, we order them by value. In other words, accelerometer 1 is always the accelerometer with the highest weight (or average weight), accelerometer 2 is the second highest, etc. It doesn't matter whether with one initialization the 2nd copy of the norm has been selected or the 3rd, etc. What matters is that only one copy is selected each time. In the end, we expect that the mean of the highest weights should exceed largely the others.

The results are displayed in fig. B.1. For the first scenario (accelerometer and gyrometer, first line), all networks but the weighted probabilities fusion succeed to give some importance to the gyrometer. For the second scenario, only Gradient Blend fails the test: others assign a small value to the useless sensor. For the third scenario, only the weighted probabilities method succeeds clearly. One could argue that selective fusion also passes the test, but the standard deviations in the weights of the second and third weights are still quite high, indicating that some runs still fail nonetheless.

As the 'weighted probabilities' method did not succeed with the first scenario, none of the methods we used succeeded in all three scenarios. Selecting the sensors automatically by giving them all to a single network seems unlikely to succeed.

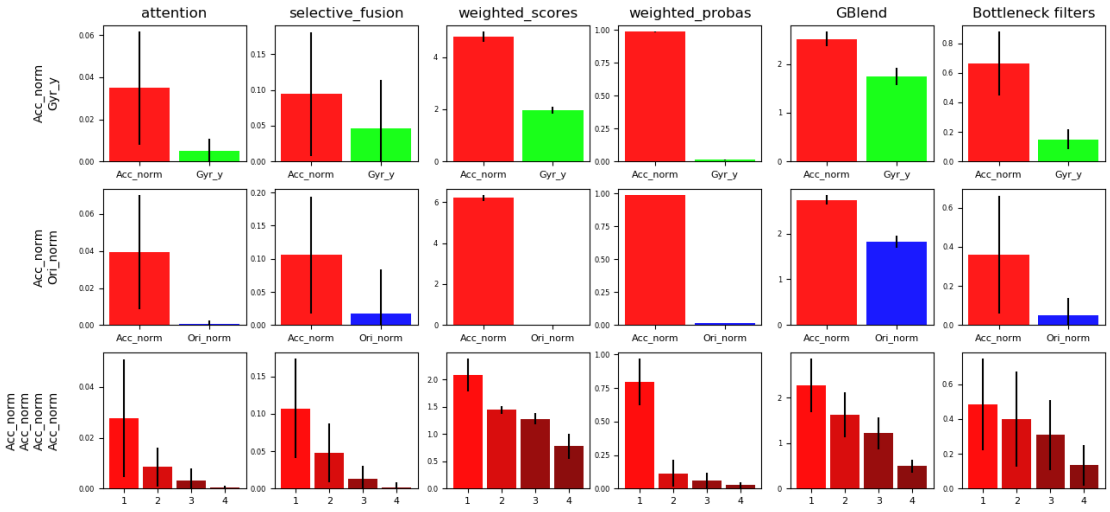


Figure B.1: The weights of each sensor in the three scenarios for different automatic sensor selection methods. Error bars indicate the standard deviation between each of the five runs. The values of the weights in the last scenario are decreasing because we sorted them (as the sensors are equivalent, we wanted to see if the network listened to only one sensor).

B.3 Conclusion

Even with an evaluation of each of the individual sensors (see section 2.3.3), choosing an optimal sensor *combination* is a non-trivial combinatorial problem. We could cite works that rely on the intelligent removal of features to evaluate their usefulness (see [307], for instance), and how they can be adapted to sensor selection. However, these works are out of scope for this chapter.

For the automatic sensor selection, our rationale was more to use what we had developed as data fusion methods, and see what happens. As this method did not pass the tests we set up for it, we see no other choice than simply keeping the sensors we selected using prior knowledge. We realize that the idea which pushed us to experiment in this direction was that a neural network would always find the best combination of sensors. The results show that neural networks are not necessarily optimal, nor efficient.

Several works did try to quantify the energy spent using each sensor by trying to get rid of each of them, and seeing if the loss of performance is acceptable given the time, or energy saved by not using them [40, 133, 131]. However, these methods rely on evaluating each sensor combination once.

Appendix C

Random Search for hyperparameters on the GeoLife dataset

Deep neural networks are trained by the update of thousands, or even millions of values called *parameters* of the network. However, the training process is guided by a handful of variables specified by the user at the beginning of the training process. These variables, which remain constant all the way through, are called *hyperparameters*. Finding the optimal combination of values for the hyperparameters is a common problem to all Machine Learning practitioners, and hyperparameter optimization has become a separate field of research ([308, 309]). We could also consider Neural Architecture Search ([310]), a type of problem where the researchers try to optimize the architecture of a network (number of filters and layers, organization of the operations *etc.*) using methods that are specific to deep neural networks.

There are several competing methods in the literature [309], and, in particular, Bayesian optimization ([311]) is a popular option for hyperparameter selection. However, we selected Random Search for its simplicity and ease of implementation. This short chapter is devoted to presenting how we found the hyperparameters of the baseline architecture for the GeoLife dataset.

C.1 Experimental setup

The most straightforward way to know which combination of hyperparameters is the most optimal is to test all combinations. Unfortunately, this method, named *grid search*, has a complexity that grows exponentially with the numbers of hyperparameters to evaluate. For instance, in our case, this would mean evaluating nearly six million combinations, with only thirteen hyperparameters to choose (see table C.1 for a list of all possible hyperparameters). This is wasteful if we consider that some hyperparameters will end up having a low influence on the performance.

Random Search [312] is a method that relies on the independent sampling of a set of hyperparameters. To avoid iterating over the low-influence hyperparameters as grid search does, Random search simply consists of setting intervals for each hyperparameter, and selecting a series of hyperparameters at random, uniformly, to evaluate the architecture. After a significant number of sets of hyperparameters are evaluated, one should see different trends when they plot the performance against the diverse hyperparameter values.

hyperparameter	possible values	chosen value
normalisation	none, max-min, p-robust (1, 5, and 10 percentile)	none
filtering	none, median, savitzky-golay (order 3, 9 points)	none
segment size	256, 512, 1024	1024
signal types	speed, acceleration, bearing [28]	speed, acceleration
learning rate	0.01, 0.03, 0.1, 0.3, 1.0, 3.0	0.01
regularization parameter	1.10^{-4}, 3.10^{-4}, 1.10^{-3} 3.10^{-3}, 1.10^{-2}, 3.10^{-2}	3.10^{-3}
batch size	8, 16, 32, 64, 128	128
number of conv blocks	1, 2, 3, 4	3
architecture (see fig. C.4)	classic, residual	residual
size of the first block	8, 16, 32, 64	16
number of hidden FC layers	1, 2	1
size of the FC layers	8, 16, 32, 64, 128	16
dropout	0, 0.1, 0.2, 0.3, 0.4, 0.5	0.2

Table C.1: The search space of random search for the GeoLife architecture

There are many hyperparameters to set in order to create our GeoLife neural network. We decided to be broad and include many of them (see table C.1 for an exhaustive list): architecture (number of layers, number of filters per layer, ...), learning process (*e.g.,* learning rate, batch size). We also included several values that are not hyperparameters *per se*, but might have an influence on the final performance: what signals to include (speed only; speed and acceleration; or speed, acceleration, and bearing [28]), how to normalize the data (no normalization, max-min, or p-robust using the first, fifth, or tenth percentile); or input data cleaning (no cleaning, median filter with size 3, or Savitzky-Golay filter with 9 points and order 3). Contrary to the GeoLife model we presented in chapter 2, all the architectures we used in the Random Search required the input segments to have the same size. We saw in chapter 4 how to make use of segments of any size, but for now, we keep on having fixed-size segments. One work [28] used 200 points-long segments on the uninterpolated dataset, which is why we compare similar values: 256, 512, and 1024 points-long segments. The hardware we used is the same as the one we described in chapter 4 (section 4.2).

C.2 Results

We created and trained 562 models with hyperparameters drawn uniformly, and looked at the influence of each of the hyperparameters separately from the rest. When a clear trend was observed (see fig. C.1 for examples), we chose the best hyperparameter; and we opted for the less complex solution (in terms of computational requirements or number of parameters) when no clear choice could be made (which happened for the input filtering and signal normalization). We select the best hyperparameters visually, table C.1 displays the values we chose.

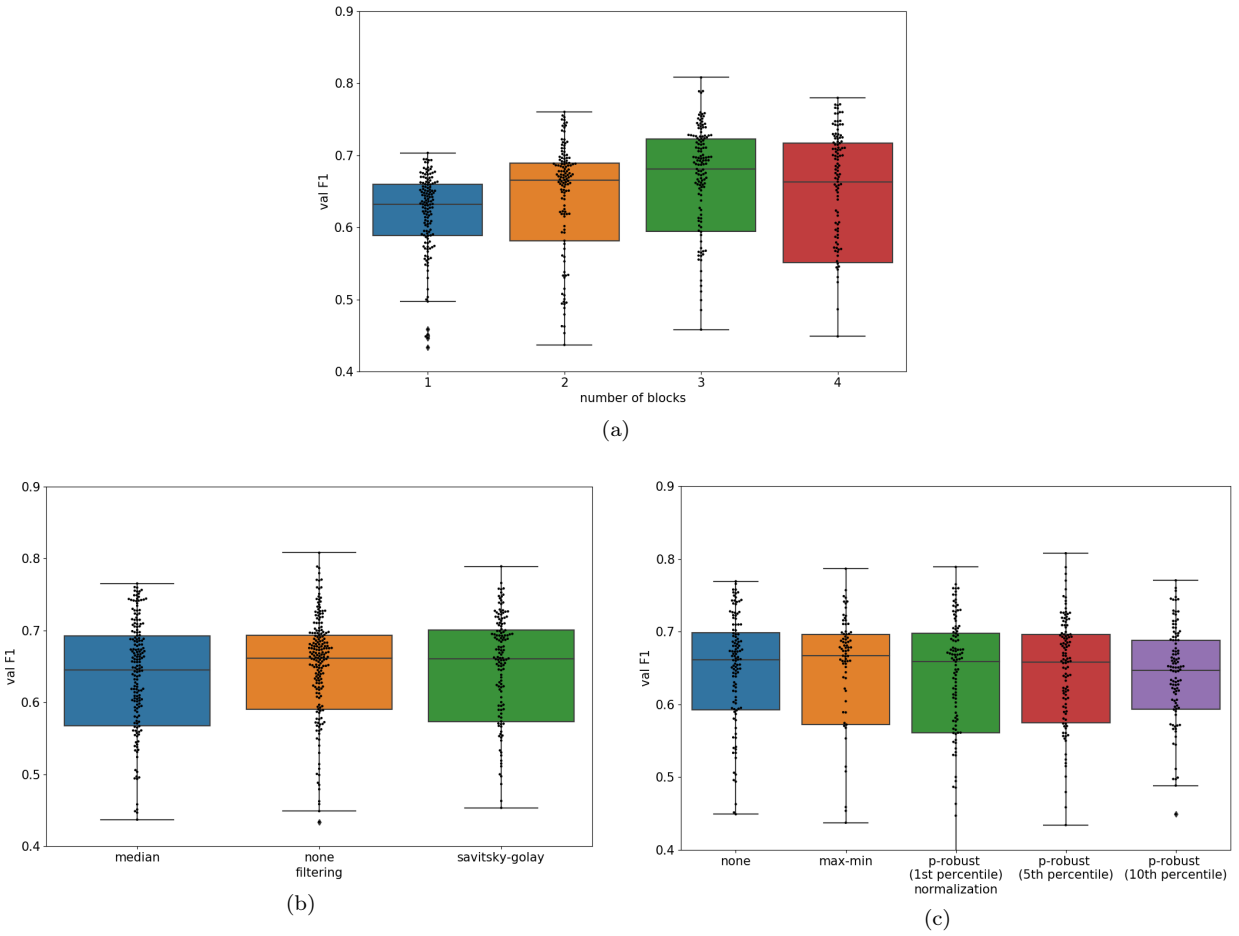
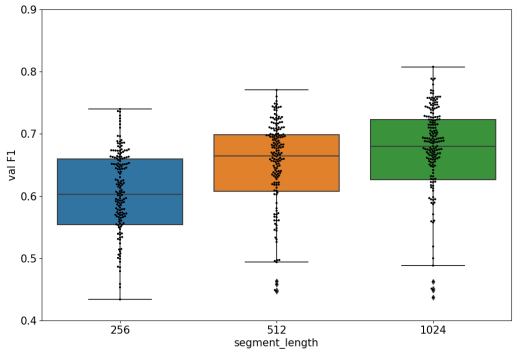
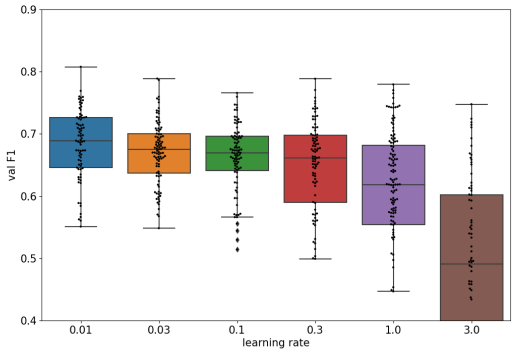


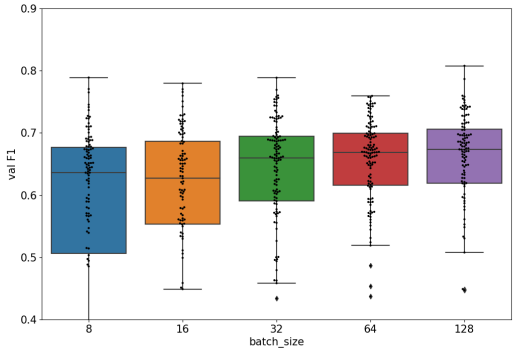
Figure C.1: Swarm plots representing the influence of several hyperparameters from random search. Each black dot represents one neural network. Subfigure (a) represents an example of significant choice: N = 3 blocks is strictly better, performance-wise. (b) and (c) are examples of unimportant hyperparameters: using savitzky-golay filters, for instance, is the same as using no filtering.



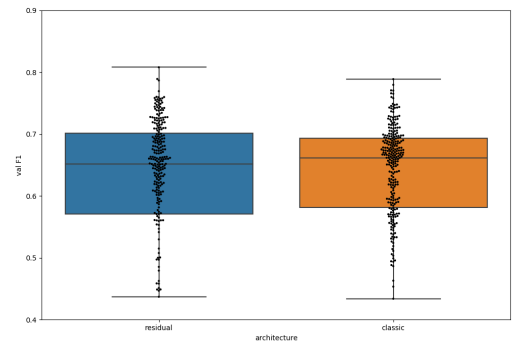
(a) segment length



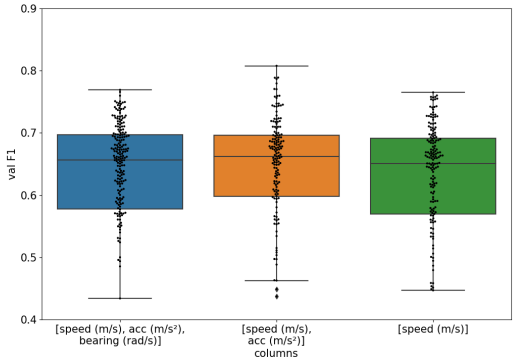
(b) learning rate



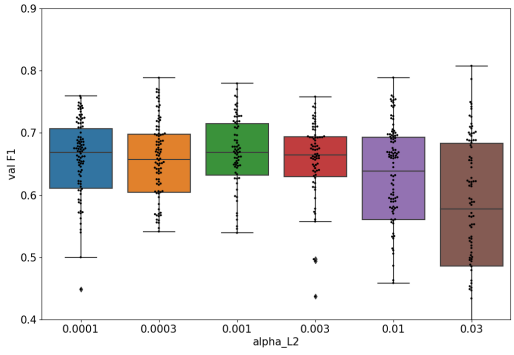
(c) batch size



(d) architecture

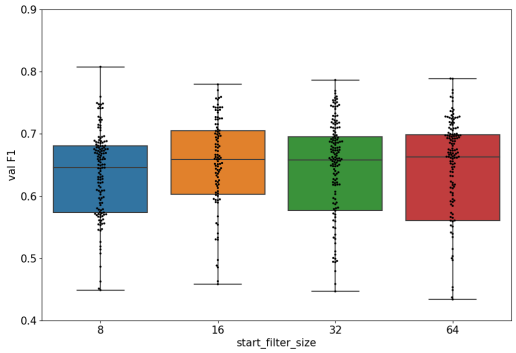


(e) type of inputs

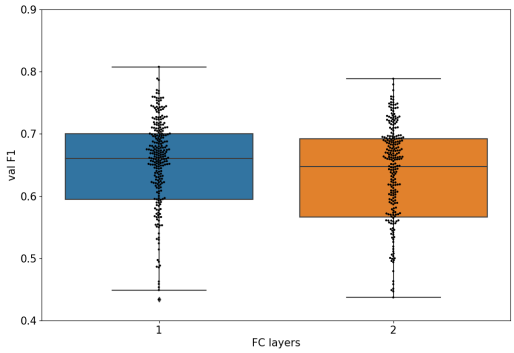


(f) regularization parameter

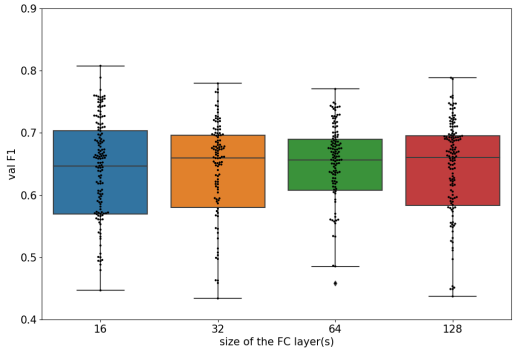
Figure C.2: Swarm plots representing the influence of several hyperparameters from random search. Each black dot represents one neural network.



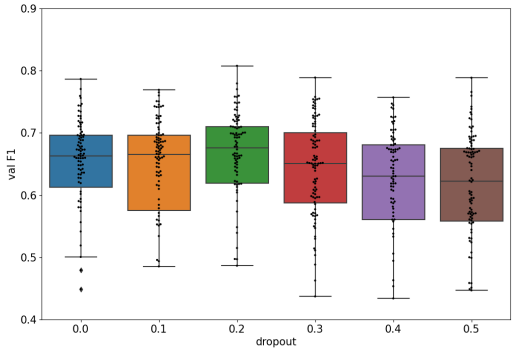
(a) size of the first block



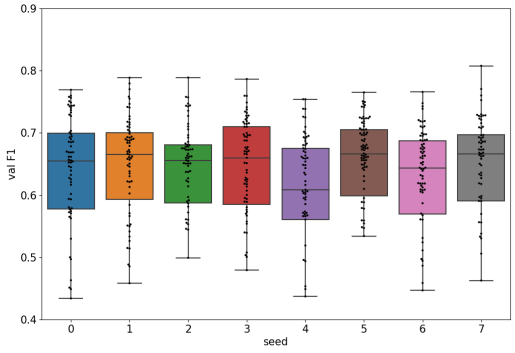
(b) number of hidden FC layers



(c) size of the FC layers



(d) dropout



(e) seed

Figure C.3: The swarm plots representing the influence of all hyperparameters from random search. Each black dot represents one neural network.

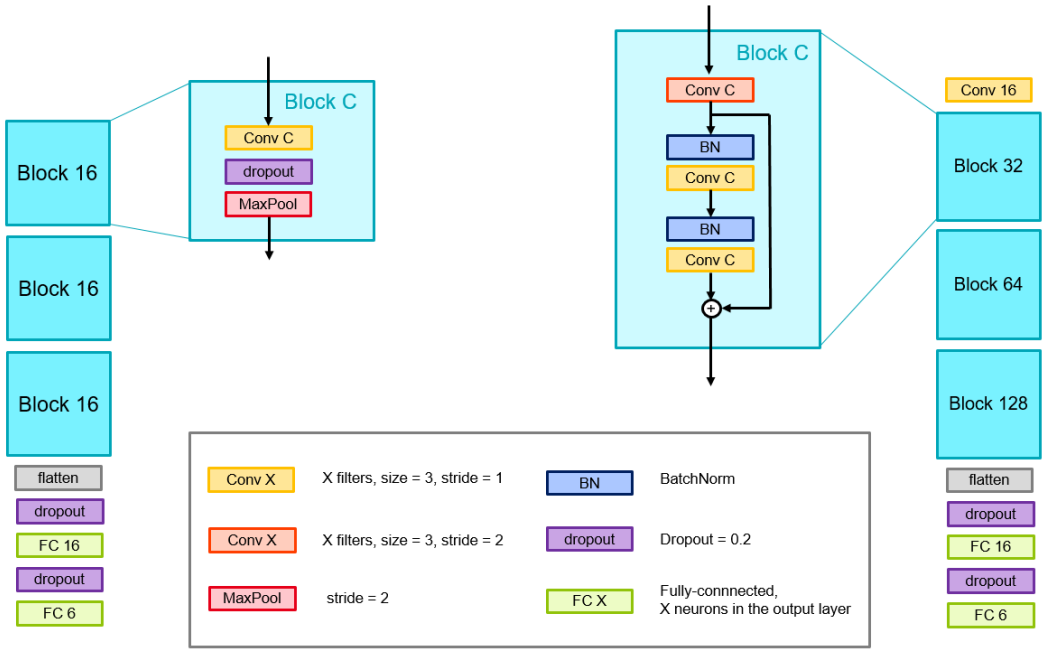
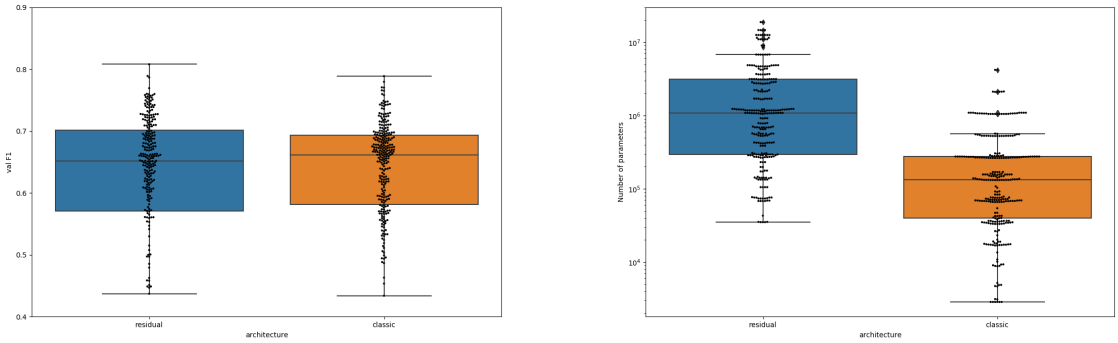


Figure C.4: The classic (left) and residual (right) architectures we used for the Random Search. The classic architecture comes directly from [104], while the second one is inspired from ResNet [150].

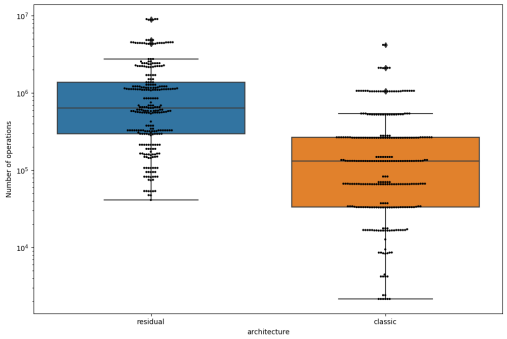
C.3 Influence of the architecture

The architecture of a network is the most determinant hyperparameter to its performance, which is why we will spend some additional time to evaluate it, even though the architecture type was one of the parameters covered by random search. We remind the reader that we chose between the two architectures depicted in fig. C.4: the 'classic' architecture, using a regular sequence of layers, coming from [104], and a slightly more complex one, using residual connections [150]. The result of the random search is that both architectures have similar performances (fig. C.5a).



(a)

(b)



(c)

Figure C.5: Swarm plots detailing the influence of the architecture, in terms of performance (a), number of weights (b), and number of operations (c). Each black dot represents one neural network

When looking at the results of the Random Search (fig. C.5), the classic architecture seems better: it has similar performances (fig. C.5a), while having fewer parameters (fig. C.5b) and operations (fig. C.5c) than the residual architecture.

However, we will see that the residual architecture is in fact, more efficient than the classic one. Here, we use similar architectures as in fig. C.4, except that the number of input in a convolution layer is set to 2, 4, 8, 16, and 32. We trained five models with each architecture, with each number of filters. The results are illustrated in fig. C.6, which contrast with the result of the random search (fig. C.5a): for an equal number of filters, the residual block significantly outperforms the classic convolution. We hypothesize that this is due to the choice of hyperparameters: even if we focused on medians and quartiles to look for tendencies, we somehow selected, unwillingly, a combination of hyperparameters that works better with residual blocks. In particular, a small number of filters is detrimental to the classic architecture only. We missed this interaction between hyperparameters because of the visualisation in the Random Search. This illustrates one of the limits of the univariate analysis we adopted by plotting the results. Alternatively, maybe that the use of ANOVA could have separated the influence of the interactions of the variables, as it did in a previous study [210].

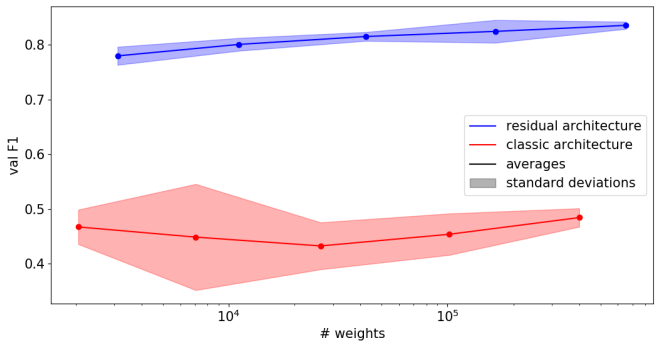


Figure C.6: The performance of each type of network as a function of the number of parameters.

C.4 Conclusion

This chapter was dedicated to our implementation of Random Search. There are two points to retain: firstly, the type of cleaning of the GPS signal does not seem to influence the performance much, which means the network is fairly robust to the noise of the GPS signal. Secondly, we showed an example of the shortcomings of Random Search: choosing one type of architecture (the regular one) with a small number of filters leads to surprisingly low performance levels. This is due to the interaction between variables, which Random Search does not take into account.

Here, we must say that the choice of a value for each hyperparameter was not the most rigorous: we simply displayed the graphs of the performance (see fig. C.1) and selected manually the value that looked best. As hyperparameter selection algorithms are usually employed to reduce the tediousness of manual selection, resorting to manual selection of parameters halfway through is counterproductive. Worse, when we say that a hyperparameter does not influence the performance, we rely on a visual inspection of the figures. Given that we have hundreds of samples, we could have employed statistical tests to assert the significance of each parameter.

We mentioned Bayesian optimization in our introduction, but the most important is to find an optimization method that takes into account the interaction between variables.

Appendix D

The curious behaviour of the spectrogram of the orientation

In chapter 3 (table 3.4 and figure 3.5), we presented a network a 250×550 spectrogram with the log-power, computed from the w axis of the orientation vector. Out of the five models we trained, two learnt nothing and kept a validation F1 of 2.8% (the score of a classifier that predicts the most occurring mode). The three others had a F1-score of $76.4 \pm 1.6\%$. This chapter is devoted to the study of this discrepancy.

The discrepancy we observed is surprising: usually, the influence of the randomness is quite small (most of our experiments lead to F1 sores that have a standard deviation of one to two percentage points). This stability is not proper to our experiments, for many publications found empirically that the seed has little influence on the final performance (typical standard deviations in the performances of deep neural networks are below a handful of percentage points, see [127, 313] for a few examples). Things become even more surprising when we look at the rate of convergence of the networks.

D.1 An observation: irregular learning of the network

Some of the time, the validation F1 score plateaus at 2.8%, and before reaching higher values, of about 70%. It turns out that leaving only 50 epochs was too short for these networks. We tried letting 30 networks learn for 200 epochs instead: fig. D.1 shows most of them learnt to generalize fairly well after only 100 epochs. The shape of the curve, however, is highly unusual: the performance remains constant for some of the epochs, before seeing its performance suddenly spike, only to reach its maximum in about 10 epochs. Moreover, this does not happen for all networks: depending on the initialization, some networks learn right away. This phenomenon seems to only occur for the full-size spectrograms with the log of the energy: with the Ori_w spectrograms with raw power (which also seemed to fail to learn anything in chapter 3), the abrupt performance increase does not seem to happen: we tried extending the learning period up to 1,000 epochs instead of 200, but the networks could not generalize.

Two questions emerge from the curves in figure D.1: why does the network learn so irregularly ? and why are the results so different between two different random seeds?

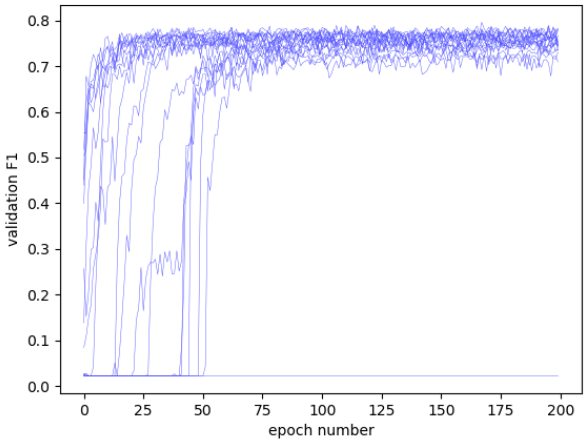


Figure D.1: The validation F1-score of 30 initializations of the same model working with full-size spectrograms of the Ori_w signal

D.2 Why does the network learn so irregularly ?

In many works (see [150, 5, 314], for instance), the loss as a function of time is fairly regular: it starts by decreasing abruptly, and, as the time increases, it keeps decreasing, albeit at a slower rate. The classic representation of a loss curve is a decreasing exponential (such as the one in the Stanford Course on Deep Learning [315]). Usually, only external actions from the researcher (such as lowering the learning rate [150] or freezing some weights [269]) change the rate of convergence of the curve to such degrees. In our case, we keep using the same learning process, without alterations, for the entire 200 epochs.

Without providing a full explanation, we can give some insights about it: Firstly, the fact that the F1-score seems to remain constant at the beginning of the learning process does not mean that nothing happens. As gradient descent occurs, the weights are being changed, but this does not seem to change the loss by much. We make the hypothesis that the network starts by overfitting, that is, memorizing the inputs one by one. When we train one network to learn randomized labels (which corresponds to pure overfitting or memorization), the training decreases exponentially, but its decrease rate is very slow: the training loss is equal to 2 after the first epoch, and equal to 0.5 after 5,000 epochs. As a comparison, when using clean labels, the training loss reaches the same value after 50 epochs only. This means that the decrease in the loss function is not likely to remain invisible after only fifty epochs. But this does not mean that the network learns nothing: Graziani *et al.* [316] showed that when training models on randomized labels, the models still learn some discriminant feature with their first layers (the memorization only takes place in the intermediate layers). In other words, even overfitting networks learn some discriminant features. We do not know if this behaviour appears because the features help the network in his memorization task, or if it is a by-product of the properties of gradient descent [292, 317, 318]. What matters to us is that if this behaviour takes place in our problem, this means the networks takes some time to develop features while beginning to memorize the samples, and suddenly finds a way to use the features that generalizes well. However, we do not confirm this reasoning with any experiment, and we only have assumptions so far.

D.3 Why does the behaviour depend so much on the random seed ?

When looking at the performance curves in fig. D.1, it seems that the speed at which the network learns depends on the initial random seed: most networks learn quite fast, but some take more time (ten to fifty epochs) to really start generalizing. Some others did not even start learning at the end of the 200 epochs. This high variability is, to the best of our knowledge, unheard of in deep learning (see [319], fig. 8, for

an example of different learning curves of the same initialization). Even with our other experiments, the learning curves did not seem to change much when we changed the initial random seed.

In our case, there are three sources of randomness in the training process of our neural networks: the initialization of the weights, the selection of samples in the training set to form a batch, and the dropout during the training (Dropout is turned off for evaluation). However neural networks (for the weight initialization), Stochastic Gradient Descent (for the batch formation), and Dropout have been used for years, and we cannot think of any publication attesting that one of these three sources of randomness has a significant impact on the final performance. The question remains unanswered.

D.4 Why are we even talking about this ?

It might seem unusual to linger over a behaviour we know to be extremely marginal. We could handwave it away as a statistical outlier, an inconsistency of a virtual signal computed from unknown heuristics, and used to solve a sub-sub-field of research that only matters to a handful of practitioners. We will not do it. In fact, we believe this example to be precious.

There is still much to be understood about deep learning. We see publications trying to model the ability of neural networks to find global minima from using mathematical considerations [320] or mathematical models from theoretical physics [321]. In this regard, having an example where things *do not work* is helpful: it could help us understand why things work. If we could pin down the exact reason why all these inconsistencies appear here and not with other neural networks, we might answer these broad research questions about neural networks. Granted, the reason might be peculiar to TMD, or this very sensor. But even in this case, knowing this reason might help us to learn more about other sensors, or even images, and the reason why they work so well with deep networks.

This is another proposition to add to the never-ending list of interesting avenues for future work.