Fourier Analysis and Machine Learning for Time-Series

Francisco Andrés Vargas Palomo

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2016

Abstract

This study introduces a fully connected spectral ANN layer, which computes an adaptive Fourier Spectrum using artificial neural networks (ANNs) for the classification of time-series data that exhibits periodic patterns. The Fourier Spectrum is adapted by the backpropagation algorithm when training the network.

Additionally, model compression and Shapely value analysis are conducted in order to deal with having more than one time-series. Shapely value analysis is used to rank the quality of the time-series which are involved. Model compression (distillation of knowledge) is responsible for transferring the knowledge from a model trained on two time-series to a model trained on one for the purpose of this study.

Acknowledgements

To my mother (Ligia Palomo) and brother (Arturo Vargas) for their loving and unconditional support throughout my degree and to my father (Julio Vargas) for his the equal support in addition to having the patience to aid me with brainstorming across this project and many others. Without them, their love, their amazing humour and company having the opportunity to carry out this study would have not taken place.

To my cousin Andersen Palomo who passed away recently in a tragic accident. He was one of my close friends during childhood and highschool, always made sure I was ok and helped me stay on track. A caring and smart young man.

A special thanks to my friends Jason Grealey, Padryk Merkl, Ramona Comanescu and Aleksandra Zaforemska for the help in proofreading this report.

Another special thanks to my friends Seth Oberlander, Danylo Matselyukh, Lukas Danev, Aleksandra Zaforemska and Nedko Nedev for the imnense support during difficult times.

I would also like to thank my supervisor Dr. Amos Storkey for his guidance and technical support throughout this project.

Table of Contents

Chapter 1

Introduction

This project was originally supposed to solve a real world problem for a local Edinburgh based company called IceRobotics.ltd .

IceRobotics' domain is in the livestock industry. Their main product is the iceQube which is a 3 axis 6Hz sample accelerometer. This accelerometer is attached to cattle in a farm in order to monitor and manage the cattle's behavior.

The company maintains a set of basic statistical algorithms that attempt to detect the health states of the animal, such as its oestrus phase or lameness.

Cattle are animals of habit. This means that they exhibit recurring behavioral patterns over a daily and weekly basis. These patterns are best represented in the frequency domain. Therefore, this study will use this domain to represent the feature space of the data.

As there was a problem obtaining the intended data set, an alternate data set was used from the UCI machine learning repository. The alternate data set tracks motion of human subjects, performing a range of sports, using accelerometers in a similar fashion to the intended data set.

This project has attempted to use the UCI machine learning dataset in a way which resembles the nature of the IceRobotics dataset the most.

The main focus of this project is embedding one of the most common methods of spectral decomposition, namely the Fourier transform, into neural networks. This will be done by combining methodologies from both Fourier analysis and machine learning to construct, support, and evaluate this embedding process. The motivation behind this is to create a more flexible and adaptive Fourier transform.

As well as this, some additional machine learning methods are explored to exploit the presence of multiple devices (accelerometers) per subject. This was the case of the IceRobotics dataset (2 devices per cattle).

1.1 Aims

As outlined previously, there are a number of tasks which need to be addressed. This study will:

1. Identify which devices contribute the most to the classification task.
2. Construct a model which is able to efficiently make use of the underlying dynamics of the dataset. Dynamics meaning the periodic behaviour and patterns that are present in the dataset.
3. Use several devices to train a model and compress the knowledge gained into one model that uses only one device. This ensures that the commercial cost of devices is minimized at test time.

1.1.1 Assessing the Contribution of Each Device

In each scenario this study identifies each device as a player (agent) and asks which cooperations are meaningful among these players in the game (coalitional game). One can formally define such a scenario in the following manner:

1. $Accelerometers = A_c = \{1, ..., n\}$, (agents);
2. Coalitions: $\forall C \subset \mathbb{P}(A_c)$, s.t. $(\mathbb{P}(A_c) \equiv 2^{A_c})$;
3. $G = A_c$ is the grand coalition;
4. $\nu : \mathbb{P}(A_c) \to \mathbb{R}$ is the pay-off (utility) function of the game s.t. $\nu(\varnothing) = 0$ (ν could be the accuracy when classifying the test set) ;
5. a cooperative game is the pair $\Gamma = (A_c, \nu)$;

Having set up such game mechanism one can ask :

Which accelerometer placement contributes the most towards the modeling?

1.1.2 Artificial Neural Network (ANN)

Artificial neural networks were selected as the classifier in this project due to their:

1. Capacity to find non linear topologies that discriminate .
2. Ability to be initialized in different ways.
3. Great versatility in terms of structure (number of layers, number of hidden units, etc).

1.1.3 Spectral Analysis

As previously mentioned, as periodicity is underlying in the dynamics of the cattle motion, this implies that the frequency space is a favourable domain in which to represent the data.

Real world constraints such as small datasets, make it hard for a machine learning algorithm to learn a transformation that projects the input vector into a frequency space. Additionally, non linear representations such as the phase and the amplitude of projections into Fourier space are even harder to learn.

Thus, some of these transformations will be embedded in the models and are allowed to be relaxed when optimizing the likelihood of the ANNs.

1.1.4 Model Compression

One can compress the knowledge of an ensemble of models into a single model. As outlined previously, the motivation for doing this is training a model using several devices. Then, use the knowledge gained from training this combination of devices to train a model which uses only a subset of the devices used in the larger model.

This gives the advantage that at test time one may only need to use a single device per person (or cattle).

The general terminology of a procedure which efficiently allows us to do this is model compression.

Chapter 2

Notation

1. $\vec{a}$ denotes a column vector.
2. i denotes $\sqrt{-1}$.
3. $\mathcal{F}\{x\}$ denotes the Fourier transform of x.
4. $|\mathcal{F}\{x\}|$ denotes the Fourier Spectrum of x.
5. $a(t) * b(t)$ denotes the convolution between signals $a(t)$ and $b(t)$.
6. $a(t) \star b(t)$ denotes the crosscorrelation between signals $a(t)$ and $b(t)$.
7. $\mathcal{D}$ denotes the data set.
8. N usually denotes the number of instances in the data set.
9. $x[n]$ denotes the nth element of the discrete signal x.
10. x_l in context denotes the lth element of the vector $\vec{x}$.
11. $\vec{x}_k$ denotes the kth vector in the data set.
12. $\vec{a} \otimes \vec{b}$ denotes the outer product of $\vec{a}$ and $\vec{b}$.
13. $\vec{a} \cdot \vec{b}$ denotes the inner product of $\vec{a}$ and $\vec{b}$.
14. $A \circ B$ denotes the Hadamard element-wise product of two matrices A and B.
15. $\bar{z}$ is the complex conjugate of z.
16. A^T denotes the transpose of A.
17. $A^\dagger$ denotes the complex conjugate transpose of A.
18. $\{\vec{x}_k\}_{k=1}^N$ denotes the union of N data points: $\bigcup_{k=1}^N \{\vec{x}_k\}$.
19. RLAx denotes right leg x-axes accelerometer.
20. LLAx denotes left leg x-axes accelerometer.
21. RHAx denotes right hand x-axes accelerometer.

22. LHAx denotes left hand x-axes accelerometer.

Chapter 3

Background

3.1 Fourier Transform

One can formally describe motion of the animal as a signal $\vec{x} : \mathbb{R}^n \to \mathbb{R}^n$ where n may be the number of time-series involved in the problem.

As described before $\vec{x}$ exhibits periodic behaviour which characterizes the state the person or animal is in. Due to the periodicity of $\vec{x}$ it may be tempting to assume that the signal a superposition of sinusoids with a definite period T, Which is known as a Fourier series:

$$x_j(t) = \sum_{n=-\infty}^{\infty} C_n e^{\frac{-in\pi y}{T}}, \quad s.t. C_n \in \mathbb{C} \wedge x_j \in \mathbb{R}$$

Where C_n are orthonormal coefficients defined by:

$$C_n = \int_{-T}^{T} \frac{dt}{2T} x_j(t) \cdot e^{\frac{-in\pi t}{T}}$$

The problem with this representation of the frequency domain is that it assumes that $\vec{x}_j(t)$ repeats itself in an identical manner outside the range $[-T, T]$. This repetition can, however, not be assumed since as previously discussed, the above period of the subjects in the study does evolve over time. Therefore a more suitable representation must be used.

One can take the limit $T \to \infty$ under the reasoning that a function with no well defined period is a function whose period is infinite:

$$C_n = \lim_{2T \to \infty} \left\{ \int_{-T}^{T} \frac{dt}{2T} x_j(t) \cdot e^{\frac{-in\pi t}{T}} \right\}$$

The limit will make $\frac{-in\pi t}{T}$ become a continuous variable namely ω. Labeling $2T$ as the period T' results in:

$$C_n = \lim_{T'\to\infty}\left\{\int_{-\frac{T'}{2}}^{\frac{T'}{2}} \frac{dt}{T'} x_j(t)\cdot e^{\frac{-2in\pi t}{T'}}\right\}$$

Multiplying both sides by T' one arrives at a transformation known as the Fourier transform [22]:

$$\mathcal{F}\{x_j(t)\} \equiv X_j(\omega) = C_n \cdot T' = \lim_{T'\to\infty}\left\{\int_{-\frac{T'}{2}}^{\frac{T'}{2}} dt x_j(t)\cdot e^{-i\omega t}\right\}$$

$$X_j(\omega) = \int_{-\infty}^{\infty} dt x_j(t)\cdot e^{-i\omega t}$$

This yields a much better representation in the frequency domain for time-series based signals than the original Fourier series from which the limit was taken.
This captures that the function does more than copying itself outside a defined range, nonetheless it does not capture that the frequency space itself evolves through time.

3.1.0.1 Short Time Fourier Transform

The short time Fourier transform is a Fourier related transformation which is carried out by transforming only short windows of the signal:

$$X(\tau,\omega) = \int_{-\infty}^{\infty} dt \Pi_{l/2}(t-\tau-l/2) x_j(t) e^{-i\omega t}$$

Where $\Pi_{l/2}(t)$ is the top hat function centered at 0 with width l.

The issue of taking out rectangular windows of the signal (multiplying by the top hat function) is that in Fourier space this equates to convolution with the sinc function (convolution theorem):

$$\mathcal{F}\{\Pi_{l/2}(t)\cdot x_j(t)\} = \frac{1}{2\pi}\mathcal{F}\{\Pi_{l/2}(t)\} * \mathcal{F}\{x_j(t)\}$$

Where $*$ denotes convolution.

$$\mathcal{F}\{\Pi_{l/2}(t)\} = \int_{-\infty}^{\infty} dt \Pi_{l/2}(t) e^{-i\omega t}$$

$$\mathcal{F}\{\Pi_{l/2}(t)\} = \int_{-l/2}^{l/2} dt e^{-i\omega t}$$

$$\mathcal{F}\{\Pi_{l/2}(t)\} = -\frac{e^{-i\omega t}}{i\omega}\Big|_{t=l/2}^{t=-l/2}$$

$$\mathcal{F}\{\Pi_{l/2}(t)\} = \frac{e^{i\omega l/2}}{i\omega} - \frac{e^{-i\omega l/2}}{i\omega} = l\frac{\mathbf{sin}(l/2\omega)}{l/2\omega} = l \cdot \mathbf{sinc}(l/2\omega)$$

Thus:

$$\mathcal{F}\{\Pi_{l/2}(t) \cdot x_j(t)\} = \frac{l}{2\pi}\mathbf{sinc}(l/2\omega) * \mathcal{F}\{x_j(t)\}$$

Effectively, the Fourier transform of the motion signal is blurred by a sinc function which heavily distorts it. This is also known as the uncertainty principle in signal processing. It represents the trade-off between measuring accurately in the time domain vs measuring accurately in the frequency domain and is similar to Heisenberg's uncertainty principle which models the same over momentum and space.

In order to solve this issue, one can multiply the top hat function with a smoothing function such as a Gaussian for example, which equates to having a window function W that is 0 outside a specified range yielding the general form of the short time Fourier transform:

$$\mathbf{STFT}\{x_j(t)\} = X(\tau,\omega) = \int_{-\infty}^{\infty} dt W(t+a-\tau)x_j(t+a)e^{-i\omega t}$$

With an appropriate choice for the window length of W and the overlap term a it is possible to find a good spectral representation of the signal. In which, the evolution of the spectrum is taken into account (thus modeling the model signal as a non-stationary process). The magnitude of this result is referred to as the spectrogram.

3.2 Discrete Time Fourier Transform

The examples above are for causal continuous time dependent signals. In the digital world there are only discrete measurements that sample real continuous signals. For example, whilst a person with an accelerometer moves in continuous time one can only take discrete samples of this continuous time motion signal. This means one needs to adapt the Fourier technologies to this setting. Fortunately, this is something that has already been formalized,the Shah function is defined as:

$$\text{Ш}(t) = \sum_{n=-\infty}^{\infty} \delta(t + \Delta t\, n)$$

Where $\delta(t)$ is the Dirac-delta distribution centered at zero and Δt is the sampling rate (the spacing in between samples). Sampling the signal can be represented by multiplying the original signal $x_j(t)$ by the Shah function. This will yield information about the samples contained within each of the infinite peaks:

$$\sum_{n=-\infty}^{\infty} x_j(t)\delta(t+\Delta t\, n)$$

One use of this representation is to extract information of interest, such as the average of the signal within a range $[t_1, t_2]$ [22]:

$$\sum_{n=-\infty}^{\infty} \int_{t_1}^{t_2} x_j(t)\delta(t+\Delta t\, n) = \sum_{n=t_1}^{t_2} x_j(\Delta t\, n)$$

Further derivations from this formulation allows one to arrive at the discrete time Fourier Transform representation, which takes a rather intuitive form:

$$X_j(\omega) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-i\omega n}$$

Such a formulation is not possible for real signals since one only has chunks of the signal. One can approximate this with a discrete short time Fourier Transform (DSTFT) with reasonable ranges (not $-\infty, \infty$). For simplicity, one can use a Discrete Fourier Transform which equates to a DSTFT with the segment size being that of the whole available signal and the windowing function being the top hat function.

3.3 Approximations to Spectral Representations to be Used

For the purpose of this project the DFT will be used as the principle feature extraction mechanism and one can justify this choice with the following points:

1. The data segments only last 5 seconds, therefore one is only interested in the allowed frequencies within the whole band. Looking at smaller segments such as 2, 3 seconds may not provide a range of frequencies useful enough for classification;

2. Using the DFT implies that the windowing function is a top-hat and as discussed previously, this brings downsides in terms of the time-frequency trade-off. This will be addressed later by allowing the weights of the DFT to be adapted by the ANN's that are used.

3.4 DFT Matrix

The Discrete Fourier Transform can be formulated as a linear transformation where a structure which is known as a DFT matrix, multiplies the time-series as a vector [31]. Every row of the DFT matrix (or column depending on what order the multiplication with the time-series vector is carried out) represents a harmonic (i.e $cos(\omega x) + isin(\omega x)$). As the row number increases in the matrix, so does the frequency of the harmonic vectors in each row. This allows each DFT coefficient to be interpreted as the projection of the time-series vector onto a harmonic vector. This quantifies how strong that particular harmonic is in the time-series. The overall transformation is a unitary [30] transformation, which can be interpreted as a change of basis (rotation) onto this orthonormal harmonic (Fourier) space.

One can visualize the real or imaginary parts of the DFT matrix in 3D and contour plots, these are shown below in Figures 3.1 and 3.2:

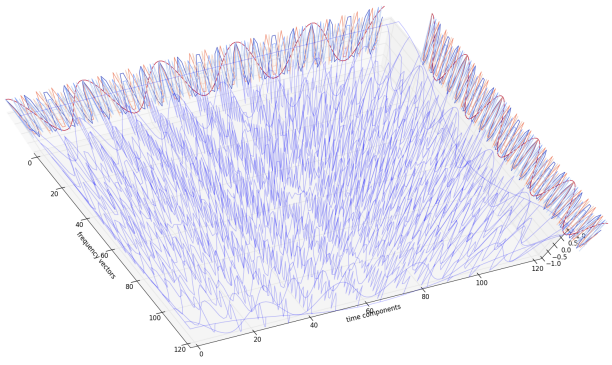


Figure 3.1: A DFT matrix 3D plot.

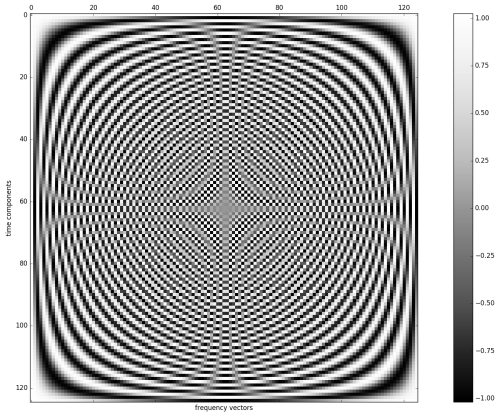


Figure 3.2: A DFT matrix contour plot.

3.5 MLE

Given parameters θ and model $\mathcal{M}$ the likelihood is:

$$p(\mathcal{D}|\theta,\mathcal{M}) = \prod_{n=1}^{N} p(y_n|\vec{x}_n,\theta,\mathcal{M})$$

Minimizing the likelihood with respect θ yields:

$$\hat{\theta}_{MLE} = argmax_{\theta}\{p(\mathcal{D}|\theta,\mathcal{M})\}$$

To ease algebraic manipulation when calculating the gradient vector, the log likelihood can be optimized instead, this is equivalent to optimizing the likelihood. In the case of neural networks, this optimization problem does not have an analytic solution and therefore, a numerical method of gradient descent is used to arrive at a maximum. This models in this study use this method to learn the training set. Due to choices of vocabulary used across this study, it is important to establish that for neural networks with softmax activation functions as output layer, maximising the likelihood is equivalent to minimizing the cross entropy error (negative log likelihood) [4] section 5.2.

3.6 Distillation of Knowledge (Model Compression)

Distillation of knowledge is the procedure that is to be used for the model compression aims outlined in the introduction. The more general problem that this procedure aims to solve is the model complexity at test time. ”Model compression works by passing unlabeled data through the large, accurate model to collect the scores produced by that model. This synthetically labeled data is then used to train the smaller mimic model” [2].

This procedure would reduce both the classification complexity and the economic cost of buying extra devices at the test time. This would have great benefits in the commercial setting in which Icerobotic operates. In this commercial setting, the more accurate model uses several devices and is therefore more complex than the simpler mimic model, which uses less devices.

When the ensemble of models has its output as a big N-way softmax, then the transfer function can be softened with a temperature parameter [14]:

$$p_i = \frac{e^{\frac{a_i}{T}}}{\sum_j e^{\frac{a_j}{T}}}$$

Determining this additional temperature parameter adds an extra layer of complexity at training time, therefore in this project T was set to 1. The knowledge transfer of

models via these ”soft labels” is expected to act as a regularizer by making the model less confident [14].

A further improvement that can be made is to weight the soft labels with the real labels when fitting to the mimicking model [14]. Like the temperature parameter, doing this will add an extra layer of complexity at training time. For this reason, this was not explored in this research due to time constraints.

Chapter 4

Data Set Specification

The data set to be used can be found `http://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities` . This data set was created by measuring motion activity from 8 distinct people whilst they were carrying out 19 different sport based activities.

4.1 Devices

4.1.1 Class labels

The labels in this data are made up of sports and daily activities:

1. sitting
2. standing
3. lying on back
4. lying on right side
5. ascending stairs
6. descending stairs
7. standing in an elevator still
8. moving around in an elevator
9. walking in a parking lot
10. walking on a treadmill with a speed of 4km/h in flat
11. deg inclined positions
12. running on a treadmill with a speed of 8km/h
13. exercising on a stepper

14. exercising on a cross trainer
15. cycling on an exercise bike horizontal
16. cycling on an exercise bike vertical
17. rowing
18. jumping
19. playing basketball

4.1.2 Sensors and Placements

Every individual in the experiment had 5 devices attached to them in the following positions:

1. Chest;
2. Left arm;
3. Right arm;
4. Left leg;
5. Right leg.

In addition to 5 devices each of them had the following measuring sensors:

1. 3 axes Accelerometer;
2. 3 axes Magnetometer;
3. 3 axes Gyroscope;

Yielding a total of 9 different readings per placement(device) and 5 different placements, thus a total of 45 different time-series per individual.

4.1.3 Data Format per Device

The original data set contains $N = 9120$ data points where each individual data point consists of a person performing a particular activity for 5 seconds, which is sampled at $25Hz$ giving us a total of 125 samples per sensor, which yields 5625 dimensions per data point in the training set.

Data points are spread uniformly across classes, thus there are a total of 480 devices per class.

4.2 Choice of Devices

In order to make a reasonable mapping between the original problem using the IceRobotics dataset and the sports and activities data set it is reasonable to limit the experiments to one or two devices depending on the nature of the experiment.

Another choice this research makes is the use of devices corresponding to the legs, since just like with cattle, human legs are constantly making contact with the ground. Therefore, this choice will yield physical similarities between the readings of both devices.

4.3 Training, Validation, and Test Sets Construction

The validation and test sets were created by picking out a different person at random from the data set for each set, then assigning all the corresponding data points to each of the sets. After carrying out this assignment, the training set was constructed from the remainder of the data points.

The choice of selecting a random person rather than a percentage of data points was taken because if the latter is chosen, then one will effectively be over-fitting the characteristic elements of individual people to both the test and validation results. This makes the results less significant. This latter procedure is used in the original paper for which the data set was made [1]. The original paper also used all 5625 dimensions in their feature space (vs 125 used in this study). For these reasons, the Author of this study has chosen not to compare the final results of this study and the original paper.

Chapter 5

Software Framework

The framework used for this project was the one which was built for the Machine Learning Practical course. This choice was made since the author had a strong knowledge of, usage experience in and had contributed to this package. The modules and relevant classes which are to be explained in the mlp package can be visualized in the following tree:

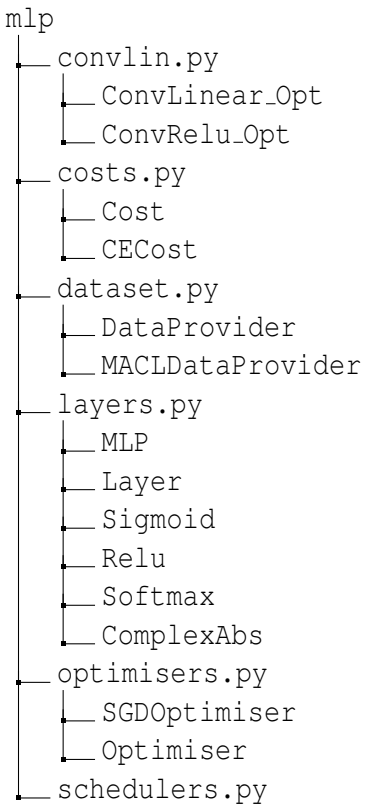


Figure 5.1: Tree Structure for MLP framework

5.1 MLP (Multi Layer Perceptron) Object

This object implements a standard multilayer perceptron (MLP) [3]. The object itself acts as an orchestrator for triggering forwards and backwards propagation for each layer within the network. The MLP object is initialized with a cost function and its main utilities are:

1. Forwards and backwards propagation in the model.
2. Adding layers.

5.1.1 Layers Object

The Layer object is the abstract class for the objects which are added to the MLP object when creating a multilayer perceptron. The Layer abstract class has two main implementations, Linear and ConvLinear_Opt, the first being a fully connected layer and the second a 1D convolutional layer [18].

Additionally, the two layer objects mentioned are wrapped such that non-linear activation functions such as Relu, Sigmoid, and Softmax can be applied to their forward propagations and their respective derivatives to the backwards propagation.

All subclasses of the Layer object have the following functions that:

1. Specify the dimensionality of the weight matrix (number of input neurons and number of output neurons);
2. Compute a forward propagation given an input vector;
3. Compute the backwards propagation and gradients given an error signal;
4. Update the weight matrix (and bias)

All operations in the subclasses of the Layer object are heavily vectorized using the numpy framework, such that minimal loops are implemented in python and the majority of operations are both matrix and Hadamard (element wise) products.

5.1.2 Cost Object

The cost object is an abstract class that specifies the different costs (error function) that the MLP will be optimized relative to. For the standard multiclass MLP, softmax activation units are used for the output layer [6] and a cross entropy function as the cost [12].

5.2 Data Providers

The data provider objects allow access to the training, test, and validation sets in this project, which are in a container class that eases the interaction with all the other components. The functions that this class performs are:

1. Randomizing (Shuffling) the samples in the data set;
2. Separating the data set into batches;
3. Prepossessing of samples (i.e. DFT, DSTFT);

The MACLDataProvider also has the feature which enables it to select which particular device or group of devices to use. This will be a useful feature when carrying out the coalition analysis in the following chapter.

5.3 Optimizer Object

This object is the abstract class which specifies the general data structure for carrying out the gradient descent procedure in order to minimize the cost function.

It operates as an orchestrator by using the MLP object and then training by updating each layer's weights when carrying out the gradient descent (GD) procedure.

The main parameters it deals with are :

1. The learning rate.
2. The Number of epochs.
3. the Learning scheduler objects (which deals with both the learning rate and the number of epochs during the GD procedure).

The implementation of gradient descent is done in the subclass SGDOptimiser. Which is a stochastic gradient descent based optimiser, it was selected due to its state of the art performance in large scale machine learning problems [5].

5.4 ComplexAbs Object (Spectral Layer)

This object is a layer subclass in which the spectral layer is implemented. The technical details behind this layer will be developed in chapters below.

Unlike the traditional layer class, this class has two weight matrices and a non linear activation function. This activation function is calculated as the square root of the sum of the squares of the two input neurons (absolute value of complex number).

Its backpropagation function and gradient updates are implemented effectively in order to pass the correct gradients. This was implemented because of the unusual representations for the weight matrices and the non traditional choice for the activation function.

5.5 Documentation

The mlp package has been heavily documented with both docstrings and inline comments. These detail technical and mathematical facts involved across the implementations. Additionally, the author has ported the whole package to a clickable and more accessible html form using epydocs (`http://epydoc.sourceforge.net/`).The CSS was also customized, to be more approachable and also to render LaTeX based doc strings. The documentation website can be found at `http://franciscovargas.github.io/MLPHonoursExtension/`.

Below are some examples of navigation across these docs, which allow the work to be understood without having to have read the sourcecode:

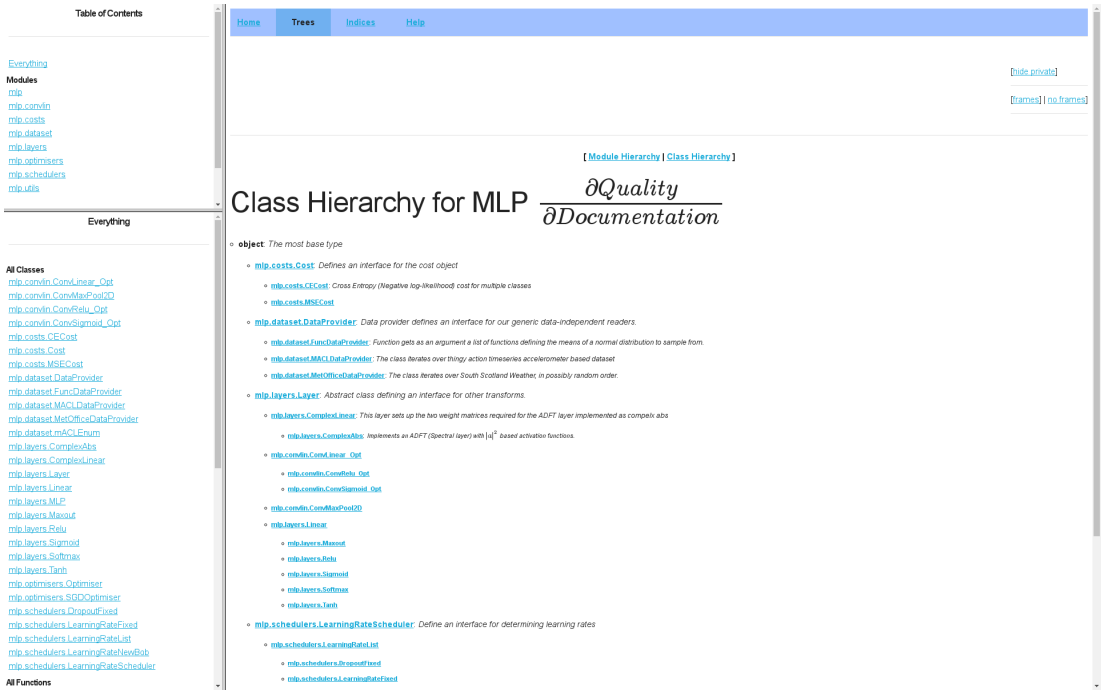


Figure 5.2: Class tree in API Documentation

Due to the mathematical nature of machine learning, usage can become inaccessible to non-specialists. Accessibility can be improved by providing approachable documentation [23].

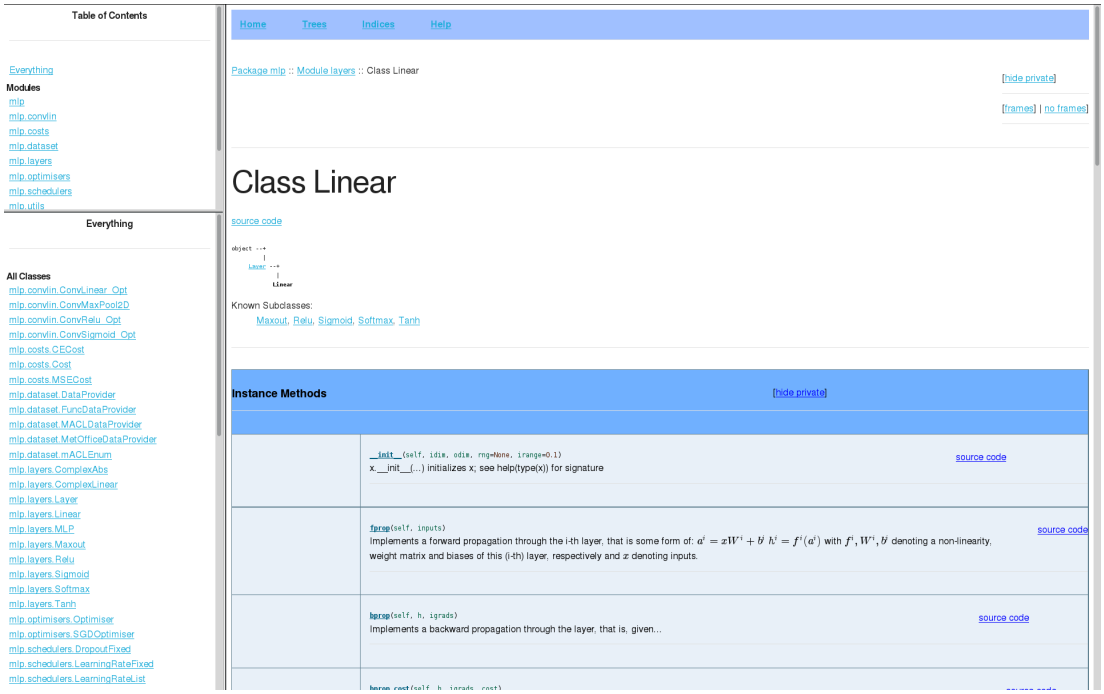


Figure 5.3: Docstrings for methods

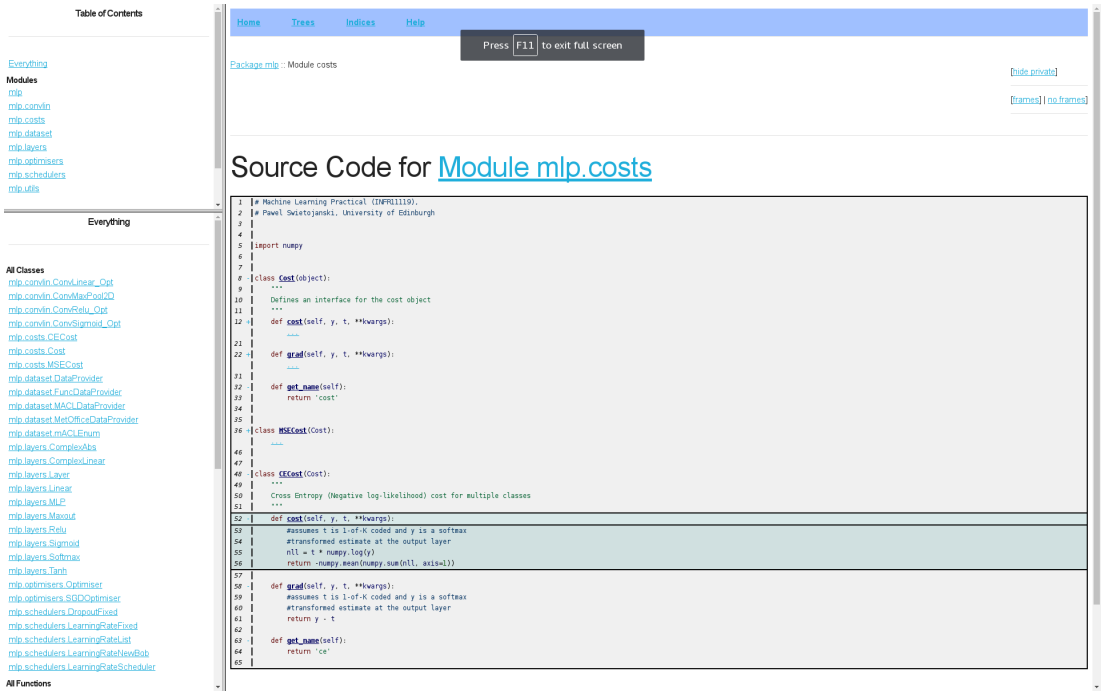


Figure 5.4: Source code view

Chapter 6

Data Mining

6.1 Frequency vs Time

It is important to examine the data in the early stages of the project in order to deduce whether or not a representation in frequency space will yield a basis that contains more discrimination power. This stage will explore both visualizations and clustering based methods that assess the new topology granted by this transformation of basis.

The new basis consists of the Fourier spectrum. The Fourier spectrum is the absolute value of the Discrete time Fourier transform components of the original time-series.

6.1.1 Selected Individual Examples

Two datapoints are picked at random from classes which are difficult to differentiate and are plotted in both bases. This is done to motivate the discrimination power of Fourier Spectrum basis. In the following example walking on a treadmill vs running on a treadmill is compared:

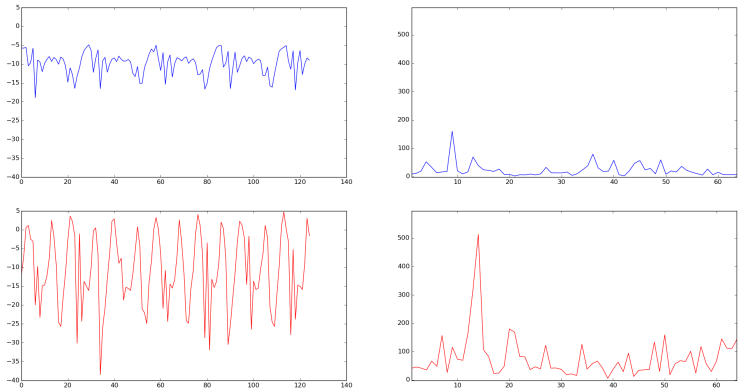


Figure 6.1: Red: running, Blue: walking
Left-x: 1/25 seconds, Right-x: 25Hz

Inspecting Figure 6.1, it is clear to see that the time-series signal (left) encodes periodicity. Also, the Fourier spectrum (right) shows that higher frequencies have significantly higher amplitudes (spikes) in the running signal vs the walking one.

6.1.2 Scatter Plot Matrix

A scatter plot matrix is a visual data-structure that allows one to explore a high dimensional space with 2D projections [9]. Each individual non diagonal subplot in the scatter plot matrix shows one dimension vs another, and the diagonal (top right to bottom left in this implementation) subplots show one dimension against itself.

Different classes have been assigned different colors in order to help assess the topology of the different feature spaces. Additionally, the author has noted in Figure 6.1 that Fourier Spectrum coefficients at high frequencies were low and relatively invariant. Therefore only the first ten Fourier Spectrum coefficients are examined in comparison to the first 10 time steps, since 125 dimensions becomes impossible to keep track of and will not fit in the margin of this page.

From observing Figures 6.2 and 6.3, it can be seen that the Fourier Spectrum basis scatter plots are more spread out. Where spread out implies that the different classes are occupying their own regions in this dimensional space. Whereas,in the time basis they are all centered on top of each other. The plots were obtained using the weka data mining software [19] (data set was parsed into arff format [24]).

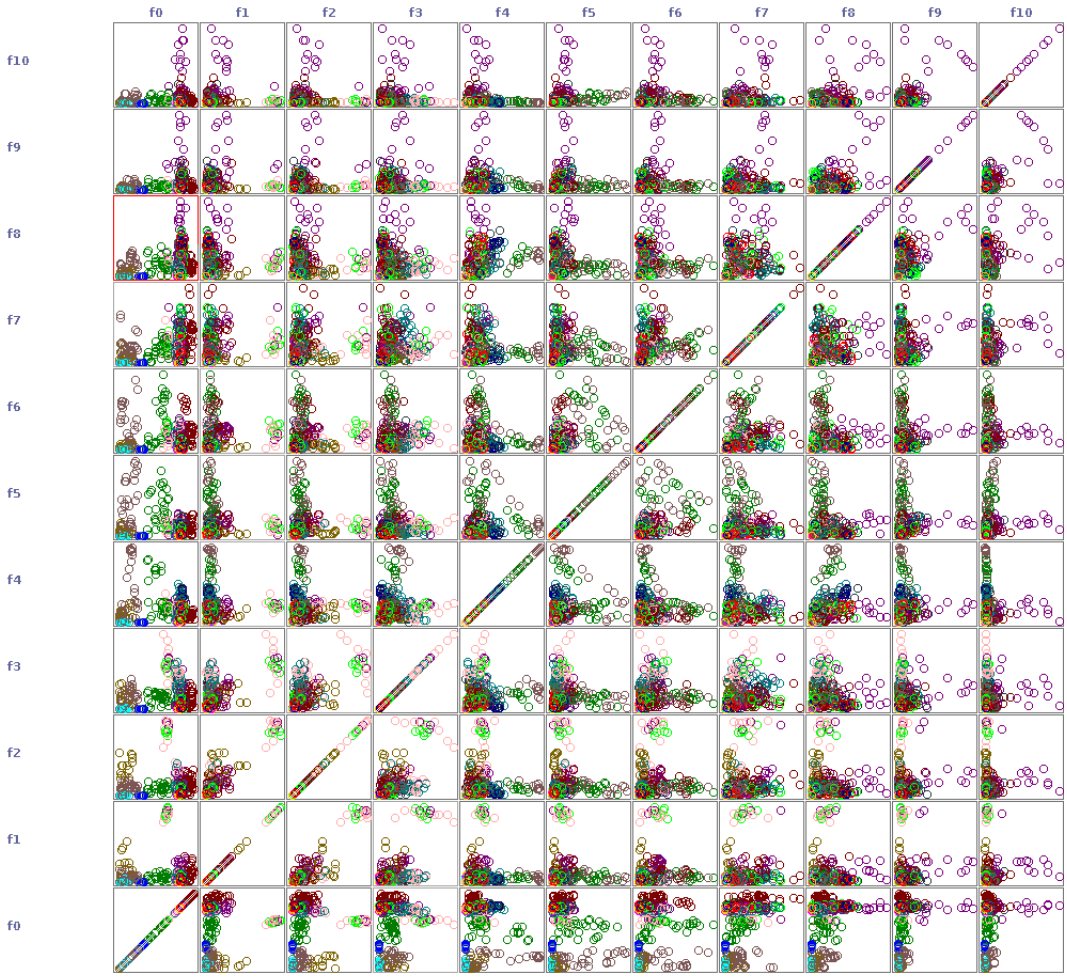


Figure 6.2: Fourier Spectrum Basis Scatter Plot Matrix

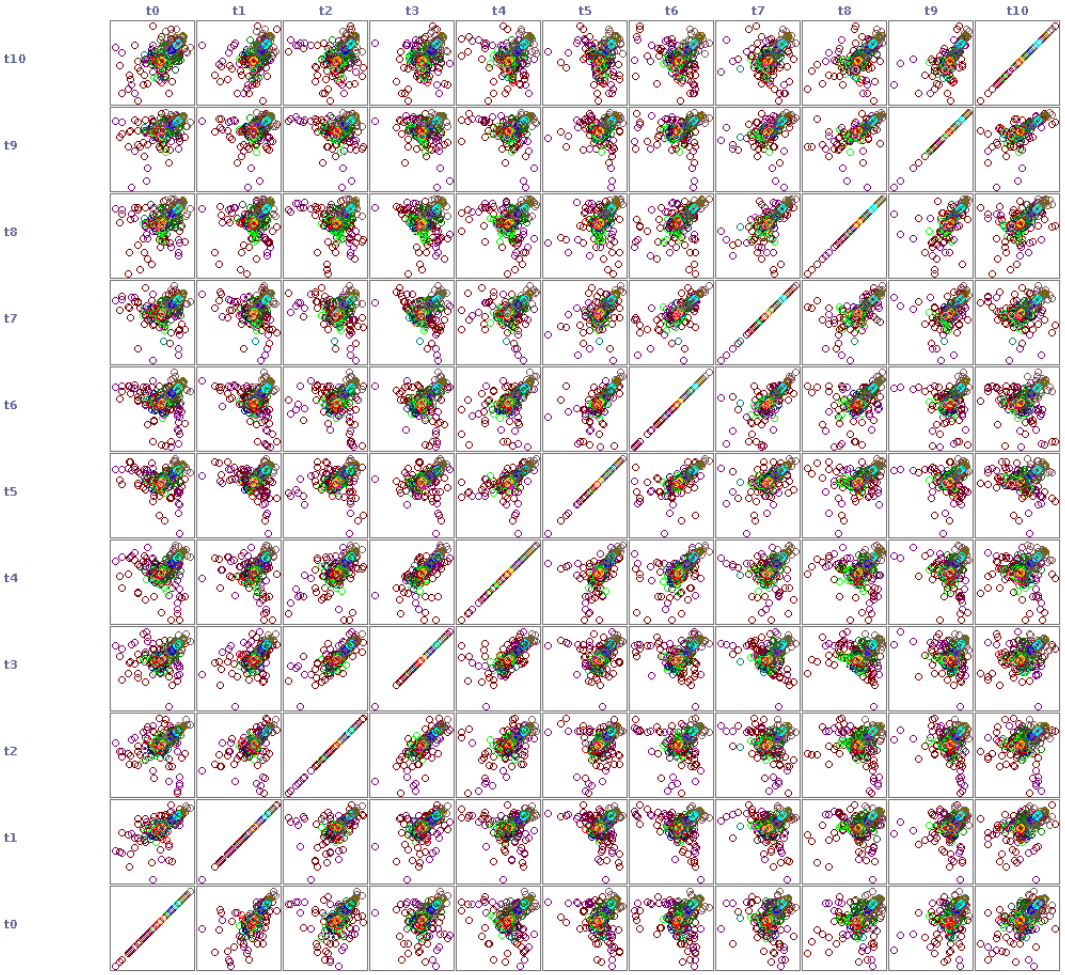


Figure 6.3: Time Basis Scatter Plot Matrix

6.1.3 Kmeans-Clustering

As previously discussed, due to the high dimensionality of the data, it is difficult to assess if there is an improvement in the discrimination power of the data purely via discriminatory visualizations of the data itself. Therefore, in this section an algorithmic approach is used to assess if the new Fourier-Spectrum basis provides an improvement for classification purposes.

A Kmeans-Clustering (setting k to the number of classes 19) algorithm [19] is used on the training set for both original and feature extracted data. After the clustering procedure is over, an intrinsic evaluation is carried out to assess how well the clusters correspond to the labeled classes in the data:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
0	0	0	0	0	0	0	0	0	0	0	0	134	0	226	0	0	0	0	0
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	300	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	3
0	0	0	0	0	28	0	0	1	211	0	0	0	0	0	0	0	0	120	4
8	0	0	0	0	104	0	6	25	97	0	0	0	0	0	0	0	0	120	5
0	0	0	0	0	0	358	0	0	0	0	0	0	0	0	0	0	0	0	6
3	0	0	0	0	4	299	0	18	36	0	0	0	0	0	0	0	0	0	7
165	0	0	0	0	117	0	0	27	51	0	0	0	0	0	0	0	0	0	8
106	0	0	0	0	32	0	0	222	0	0	0	0	0	0	0	0	0	0	9
151	0	0	0	0	5	0	0	150	54	0	0	0	0	0	0	0	0	0	10
0	0	0	0	0	0	0	4	0	0	93	0	0	66	0	0	0	197	0	11
37	0	0	0	0	114	1	2	19	187	0	0	0	0	0	0	0	0	0	12
80	0	0	0	0	81	0	1	64	134	0	0	0	0	0	0	0	0	0	13
0	0	89	0	0	0	0	0	0	0	0	0	0	0	0	118	153	0	0	14
0	1	134	0	0	0	0	0	0	2	0	1	0	0	0	152	70	0	0	15
0	0	0	2	0	0	0	0	0	0	0	358	0	0	0	0	0	0	0	16
7	107	0	0	57	3	0	88	1	0	0	0	0	59	0	0	0	0	38	17
15	3	0	0	0	28	21	142	101	45	0	0	0	3	0	0	0	2	0	18

Table 6.1: Clustering Results on Fourier Spectrum basis (Confusion Matrix)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	300	0	0	2
0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	120	4
0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	120	5
0	0	0	0	0	0	359	0	0	0	0	0	0	0	0	0	0	0	0	6
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	7
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	8
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	9
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	10
0	0	0	0	0	0	32	13	63	42	44	63	0	0	0	54	0	49	0	11
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	12
0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	13
88	0	0	50	27	0	0	0	0	0	0	0	0	0	0	0	195	0	0	14
0	0	177	0	34	0	0	0	0	0	0	0	149	0	0	0	0	0	0	15
0	0	0	218	0	0	0	0	0	0	0	0	0	0	0	0	142	0	0	16
0	27	0	0	0	25	86	75	15	13	0	2	0	53	25	0	0	1	38	17
0	0	0	0	0	0	352	4	2	0	0	2	0	0	0	0	0	0	0	18

Table 6.2: Clustering Results on original time basis (Confusion Matrix)

Fourier spectrum basis	Time basis
Cluster 0 – 8	Cluster 0 – 14
Cluster 1 – 17	Cluster 1 – No class
Cluster 2 – No class	Cluster 2 – 15
Cluster 3 – 2	Cluster 3 – 3
Cluster 4 – No class	Cluster 4 – 0
Cluster 5 – 12	Cluster 5 – No class
Cluster 6 – 1	Cluster 6 – 1
Cluster 7 – 18	Cluster 7 – 17
Cluster 8 – 9	Cluster 8 – 11
Cluster 9 – 4	Cluster 9 – No class
Cluster 10 – No class	Cluster 10 – No class
Cluster 11 – 16	Cluster 11 – 18
Cluster 12 – No class	Cluster 12 – No class
Cluster 13 – No class	Cluster 13 – No class
Cluster 14 – 0	Cluster 14 – No class
Cluster 15 – 15	Cluster 15 – No class
Cluster 16 – 14	Cluster 16 – 2
Cluster 17 – 11	Cluster 17 – No class
Cluster 18 – 5	Cluster 18 – 4

Table 6.3: Contrasting class assignments in between basis

From the previous table it can be observed that the Fourier basis has a higher class assignment to cluster ratio than the time basis. In summary, the missclasified instances per basis are reported:

Fourier spectrum basis	Time basis
3951 (57.7801 %)	4934 (72.1451 %)

Table 6.4: Contrasting misclassified instances in between basis

It can be seen from Table 6.4 that using Kmeans clustering to evaluate the discriminative power of the Fourier Spectrum basis is indeed significantly more powerful than the original time basis.

6.2 Shapley Value

As discussed in the introduction, the problem of determining which device performs the best at classifying can be tackled as a coalition game. A fair way of distributing cost in this setting is the Shapley value. Therefore, a coalition game is set up in order to carry out feature selection using Shapley value analysis [25]. A big downside of using

the Shapely value, is that its computation requires a factorial number of evaluations and an exponential number of trained models:

$$\mathbf{O}(m! \cdot evaluation\_time\_of\_n\_datapoints + 2^m \cdot training\_time\_of\_n\_datapoints)$$

In this case m is remarkably large, 15 devices times axes (agents). The training time of all these models therefore becomes somewhat intractable. Thus, to reduce the number of calculations, all the devices excluding the chest are used and only one axis is used per device. The best axis is determined by picking a device at random, training and evaluating by using a single hidden layer fully connected neural network. The axis x was determined to be the best axis, resulting in the following coalition:

1. $Accelerometers = A_c = \{RLAx, LLAx, RHAx, LHAx\}$, (agents);
2. $\nu : \mathbb{P}(A_c) \rightarrow \mathbb{R}$ Defining ν as the classification accuracy obtained on the validation set for a model trained using a subset of A_c ;

The Shapely value is defined formally as:

$$\phi_{device}(v) = \frac{1}{|m|!} \sum_{R} \left[v(P^R_{device} \cup \{device\}) - v(P^R_{device}) \right] \tag{6.1}$$

Where the sum ranges over all $|m|!$ orders, R of the players, and P^R_i is the set of players in m which precede the device in the order R.

6.2.1 Results

The resulting shapely values calculated per device are:

Device	$\phi_{device}(v)$
RLAx	0.227339
LLAx	0.197222
RHAx	0.14122
LHAx	0.1228

Table 6.5: Shapely Value per device obtained

One of the right limbs contributes the most in terms of discriminative power. This is expected since the majority of people coordinate at best with their right limbs. Additionally, the best devices are located on the legs which again is expected as most of the activities involve the legs as the most prominent actuator.

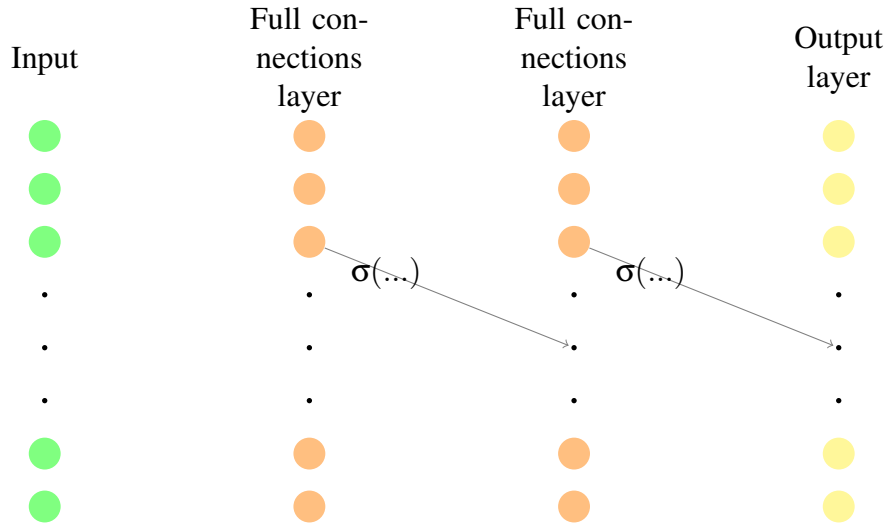
Chapter 7

Base Line Models

7.1 MLP (Fully Connected)

From the datamining it can be observed that both the time and Fourier Spectrum basis feature vectors have a rather complicated topological structure. Multi layer perceptrons are universal approximators [7] (given reasonable limits and an arbitrary error), this property makes the MLP model a standard tool used to tackle the complex topology exhibited by the feature spaces.

The baseline MLP architecture was constructed in the following way:



This network was constructed in this form in order to match the number of layers and units used in the Spectral MLP derived in the next chapter. Additionally, altering this structure (adding more layers and units) did not improve classification accuracy on validation set.

Classification experiments are run with both a time-series basis and a Fourier Spectrum basis.

7.1.1 Time-Series Basis

The raw time-series is used as an input vector for the MLP model specified in this chapter.

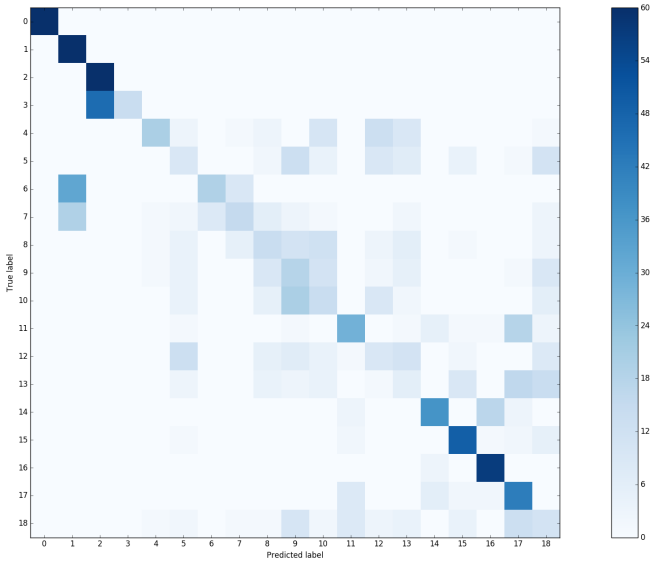


Figure 7.1: Confusion matrix for times-series

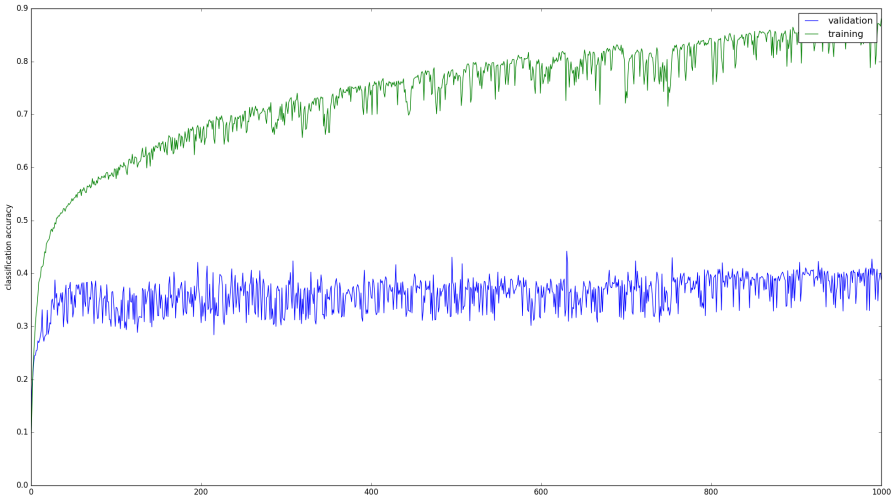


Figure 7.2: Validation and Training accuracies per epoch for the time-series basis

test set accuracy	mean log likelihood
0.476315789474	-0.388061360992

Table 7.1: Test set results for time-series basis.(after 1000 epochs)

Note: all accuracies in this study are reported in the range zero to one, as opposed to zero to one hundred percent

Overall, the confusion matrix shows that the model is making false predictions in a large number of activities (classes), and that no particular class is being highly confused in this experiment. There is a possibility that the model may be overfitting since the classification accuracy on the training set keeps increasing per epoch. Although, the validation accuracy stays at a lower but constant value, this may mean that noise is being fitted or there are irrelevant characteristics in the training set for this particular classification task. The accuracy on the test set is not very high. Although, the mean log likelihood is high, which confirms that the model has too high a confidence on average across its predictions.

7.1.2 Fourier Spectrum Basis

In this experiment the feature space is projected onto a Fourier Spectrum basis and is fed into the MLP structure outlined above.

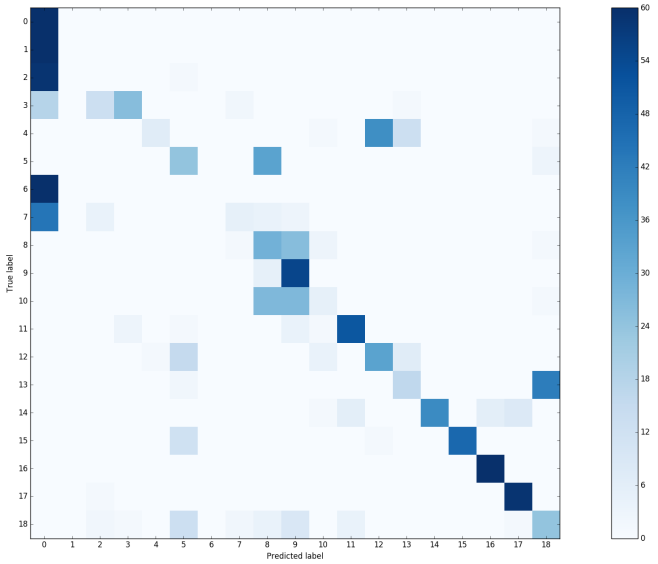


Figure 7.3: Confusion matrix time basis (performance on test set)

test set accuracy	mean log likelihood
0.473684210526	-0.82459059781

Table 7.2: Test set results for Fourier spectrum basis baseline (after 1000 epochs)

From the confusion matrix in Figure 7.3 it is clear that the model constantly predicts class 0 (which is sitting down) with classes 1, 2, 6, and 7 (standing, lying on back, descending stairs and standing still in an elevator). Both the validation and training error across time (per epoch) follow very similar curves in this model as shown in Figure

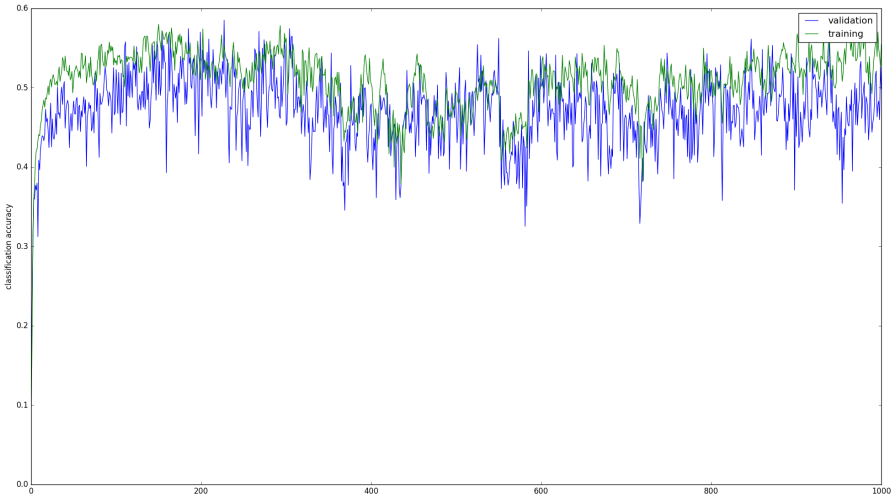


Figure 7.4: The validation and training accuracy per epoch for the Fourier Spectrum basis MLP

7.4 . This may hint that there seems to be no overfitting with this basis representation and model. The final results (Figure 7.3 and table 7.2) show a relatively low classification accuracy, which is accompanied with a similarly low mean log likelihood which is interpreted as the model having a low confidence on its predictions.

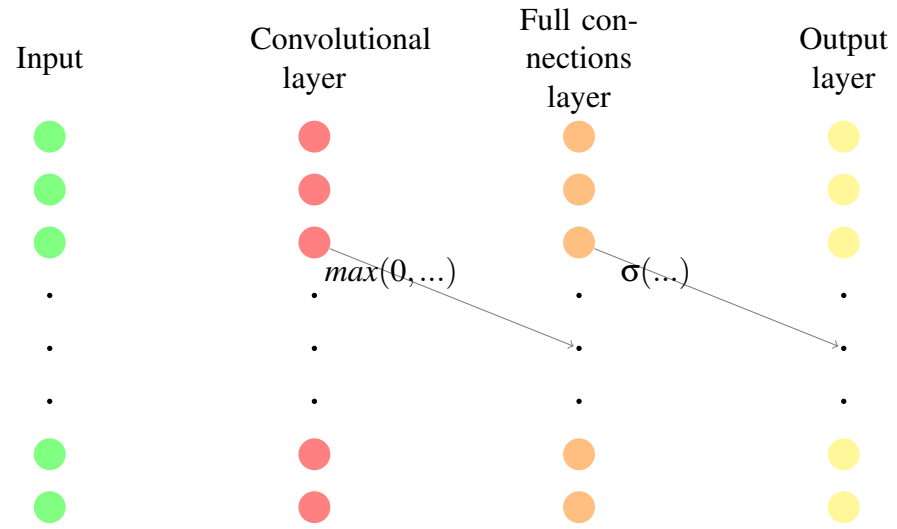
7.2 CNN

One downside of fully connected MLPs is that they do not take into account the structure of the input. For example, time-series have a strong sequential 1D structure. Therefore, removing connections such that only nearby connections are combined into one neuron (receptive field) [18], can reflect how nearby temporal data has a higher correlation than events further away. For this reason convolutional neural networks are a stronger candidate for the classification of time-series based data.

An alternative interpretation used in the digital signal processing/ Fourier analysis domain, is that convolution in time is multiplication in Fourier space. The convolution theorem allows one to interpret the convolution operator as a filtering operator. The impulse response in frequency space (Fourier transform) of the weights in the convolutional layer show how certain frequencies are filtered out in the frequency domain. Thus, a convolutional layer can be interpretated as a denoising layer, which then feeds into an MLP. Given that time-series data tends to be noisy this is another good reason why convolutional neural networks are a suitable candidate for this problem.

The following architecture of the CNN used in this model is derived from the orignal architecture[17]. Then from the original architecture, max-pooling layers were removed and the activation functions of the MLP following the CNN stack, were changed

from rectified linear units to sigmoidal activation units. This choice was made because of an empirical increase in accuracy given by the sigmoidal activation units.



7.2.1 Time-Series Basis

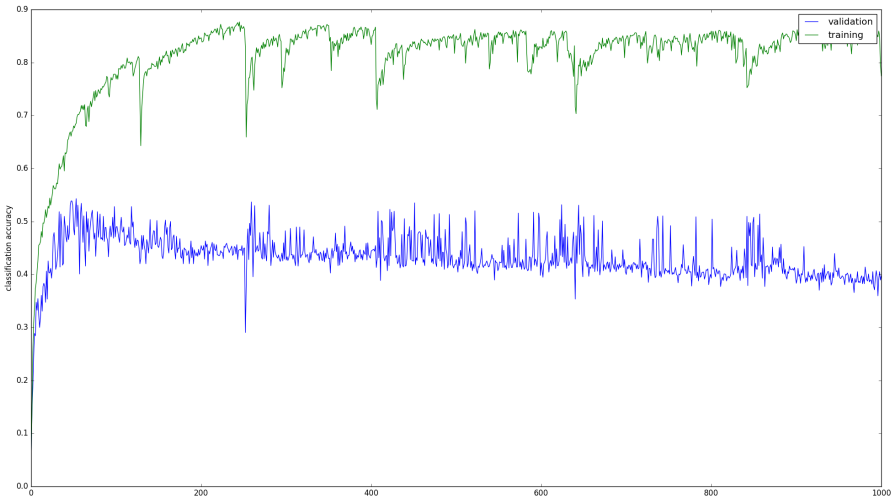


Figure 7.5: Validation and Training Accuracies per epoch

test set accuracy	mean log likelihood
0.493859649123	-0.394923602425

Table 7.3: Results on test set for time basis CNN (after 1000 epochs)

Inspecting Figure 7.5, it is clear that the model is overfitting since the training accuracy is still improving whilst the validation accuracy is decreasing. The justification as to why the networks overfit more in the time basis, is that in this basis there is a lot of high

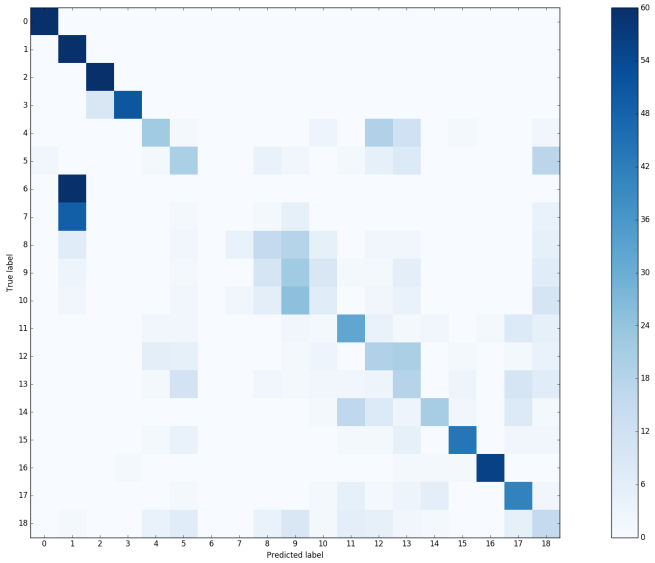


Figure 7.6: Confusion Matrix for baseline CNN time basis (on test set)

frequency noise that can easily seen as part of the signal. This confuses the networks when learning the training set. The model also has a high misguided confidence on its predictions (mean log likelihood) which again can be interpreted as an indicator of ovefitting.

It can be seen from the confusion matrix (Figure 7.6) that the network is constantly predicting class 1 (standing) as classes 6 and 7 (standing and moving around in an elevator respectively). This makes sense intuitively, as all classes require the subject to be standing.

7.2.2 Fourier Spectrum Basis

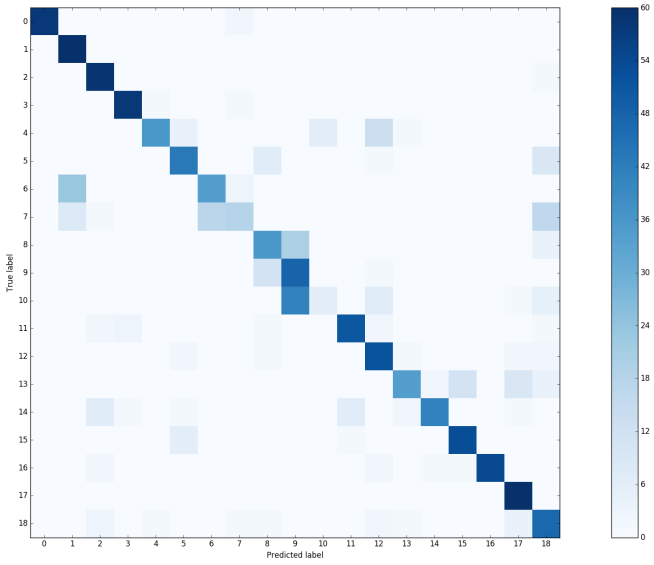


Figure 7.7: Confusion Matrix for baseline Fourier spectrum basis (on test set)

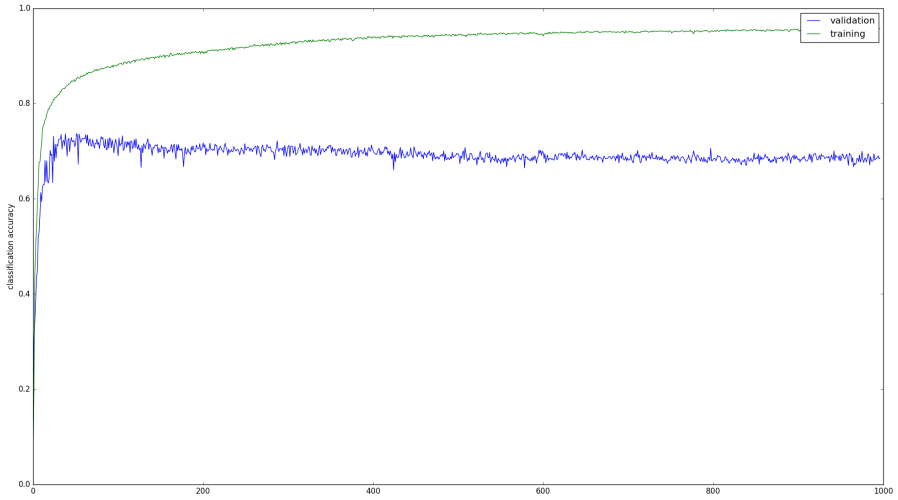


Figure 7.8: Validation and Training Accuracies per epoch

test set accuracy	mean log likelihood
0.767543859649	-0.186666155506

Table 7.4: Results of convolutional network on dft spectrum basis (after 500 epochs)

From the baselines it is clear that this is model is by far the best. The model learns relatively fast and achieves reasonable validation accuracies. In terms of performance

on the test set, the results present a high confidence when predicting and a good classification accuracy relative to the other baseline models introduced in this chapter.

7.2.3 Optimization on Convolution Operator

Taking advantage of the fact that the general setting in DSP is that finite impulse response filters are significantly smaller than the signal they are convolved with. One can carry out some algorithmic optimizations on calculating the convolution between the input vector and the CNN kernels. Note this optimization does not change the computational complexity of carrying out a convlution, but instead takes advantage of broadcasting arithmetical operations in numpy ndarrays vs for loops in python [28].

7.2.3.1 Cross Correlation

Usually, the word convolution is used when referring to convolutional neural networks. Although, the operator in which the input vectors transform by the weights, through the CNN layer is in fact a cross correlation and not a convolution.[16]:

$$(w \star x)[n] \overset{\text{def}}{=} \sum_{m=-\infty}^{\infty} \bar{w}[m]\, x[m+n].$$

Given that both the input vector x and the signal involved are finite the crosscorrelation in the CNN's can be rewritten as:

$$(w \star x)[n] = \sum_{m=0}^{d} \bar{w}[m]\, x[m+n].$$

Where d is the dimension of the kernel. Direct implementation of this formula for all elements of the output signal $(w \star x)$ yields the following algorithm:

Algorithm 1 Naive Crosscorrelation

Input: signal x , kernel w
Output: $(w \star x)$
Define d_x as the dimensionality of the input vector and d_w
Initialize output vector $(w \star x)$ (with zeroes) of dimensionality $d_x - d_w + 1$
for $n \in [0, d_x - d_w + 1)$ **do**
    for $m \in [0, d_w)$ **do**
        $(w \star x)[n] := (w \star x)[n] + \bar{w}[m]\, x[m+n]$
    end for
end for
Return $(w \star x)$

It can be observed that the upper bound complexity on this naive construction is $\mathbf{O}((d_x - d_w + 1) * d_w)$. The algorithm can be optimized by converting the inner for loop into vectorized numpy operations. Numpy ndarrays are dnoted with bold capital letters and broadcasting operations with regular arithmetic symbols in between the ndarrays($\cdot$ inner product, $\circ$ Hadamard product):

Algorithm 2 Partially Vectorized Crosscorrelation

Input: signal $\mathbf{X}$, kernel $\mathbf{W}$
Output: $(\mathbf{W} \star \mathbf{X})$
Define d_x as the dimensionality of the input vector and d_w
Initialize output vector $(w \star x)$ (with zeroes) of dimensionality $d_x - d_w + 1$
for $n \in [0, d_x - d_w + 1)$ **do**
    $(\mathbf{W} \star \mathbf{X})[n] := \bar{\mathbf{W}} \cdot \mathbf{X}[n : n + d_w]$
end for
Return $(\mathbf{W} \star \mathbf{X})$

This version carries out the inner loop over the kernel array in a vectorized form. This is a computational advantage. Further improvements can be made in exploiting numpy's broadcasting abilities due to the fact that the weight vector is smaller than the input signal. One small tweak that is made to the big-OH notation, in order to communicate the different computational models in terms of iteration between python and numpy.

$$\mathbf{O}_{python}, \mathbf{O}_{numpy}$$

Giving $\mathbf{O}_{python}(d_x - d_w + 1) * \mathbf{O}_{numpy}(d_w)$ as the complexity for the partially vectorized crosscorrelation algorithm.

7.2.3.2 Symmetry and Final Optimization

A fundamental and useful property of the cross correlation operator is that it is symmetric:

$$(x \star w)[n] = \sum_{m=-\infty}^{\infty} \bar{w}[m+n]\, x[n] = (w \star x)[n]$$

This means that it is possible to switch the order of the loops in the naive implementation making the outer loop go over the kernel vector and the inner loop goes over the input vector:

Algorithm 3 Naive Crosscorrelation 2

Input: signal x , kernel w
Output: $(w \star x)$
Define d_x as the dimensionality of the input vector and d_w
Initialize output vector $(w \star x)$ (with zeroes) of dimensionality $d_x - d_w + 1$
for $m \in [0, d_w)$ **do**
  for $n \in [0, d_x - d_w + 1)$ **do**
    $(w \star x)[n] := (w \star x)[n] + \bar{w}[m]\ x[m+n]$
  end for
end for
Return $(w \star x)$

Now the inner loop can be vectorized in terms of numpy ndarrays:

Algorithm 4 Partially Vectorized Crosscorrelation 2

Input: signal $\mathbf{X}$, kernel $\mathbf{W}$
Output: $(\mathbf{W} \star \mathbf{X})$
Define d_x as the dimensionality of the input vector and d_w
Initialize output vector $(w \star x)$ (with zeroes) of dimensionality $d_x - d_w + 1$
for $n \in [0, d_w)$ **do**
  $(\mathbf{W} \star \mathbf{X}) := (\mathbf{W} \star \mathbf{X}) + \bar{\mathbf{W}}[n] \circ \mathbf{X}[n : n + d_x - d_w + 1]$
end for
Return $(\mathbf{W} \star \mathbf{X})$

Doing this switch the computational complexity $\mathbf{O}_{numpy}(d_x - d_w + 1) * \mathbf{O}_{python}(d_w)$ is obtained. Given that d_w in this case is 5 and $d_x - d_w + 1$ is 122 the for loop in python is going over a very small number (5) and the one in numpy going over the large number (122).

7.2.3.3 Results

In order to check how fast the implementation of crosscorrelation is, the author set up an experiment in which the weight vector size is kept constant at 5. Also, increasing the size of the input signal from 1000 up to 8990, and comparing both the algorithm 4 and the scipy implementation. This experiment was run on a set of 100 input signals simultaneously (to simulate batches in training). Where, both the algorithm and scipy can handle calculating the correlation of all these signals in parallel via broadcasting:

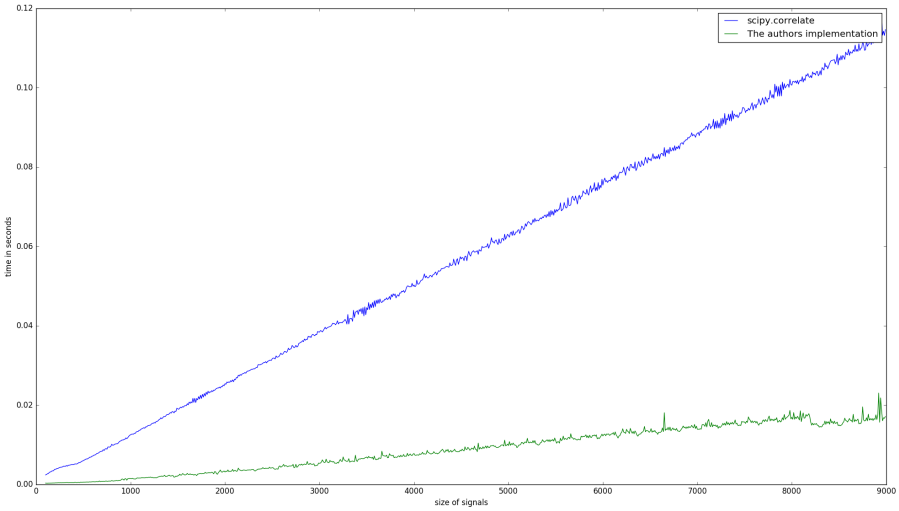


Figure 7.9: Running time of my correlation vs scipy.correlate

Chapter 8

Spectral Layer

This chapter will explore the idea of a spectral like layer for an ANN. This layer has a structure and initialization such that a forward pass across this layer, yields the Fourier spectrum of an input signal. Additionally, this layer allows the error signal from layers above, to backpropagate along the connections of this layer. This results in relaxing the weights that carry out the spectral decomposition.

8.1 Theoretical Motivation

The maximum likelihood estimate is a good starting point to help motivate the construction of the spectral layer.

$$\mathcal{L}(\mathcal{D}|\theta,\mathcal{M}) = \prod_{k=1}^{N} P(\vec{x}_k|\mathcal{M},\theta)$$

Where $\mathcal{D}$ is the data set, $\mathcal{M}$ is the model, and θ is the parameter being estimated by the MLE procedure:

$$\hat{\theta}_{MLE} = argmax_{\theta}\{\mathcal{L}(\mathcal{D}|\theta,\mathcal{M})\}$$

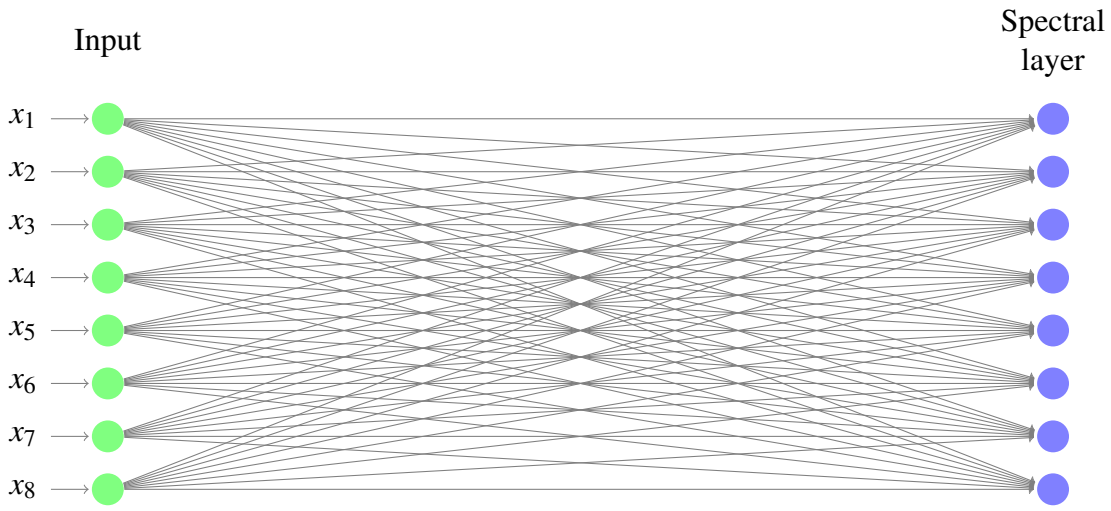
In this particular scenario, the pair $(\mathcal{M},\theta)$ define a particular ANN $\mathcal{M}$ and its corresponding weights θ. The idea of a spectral layer is to initialize a subset of θ. Namely θ_{spec}, which corresponds to the weights in the initial layer as a transformation that performs a spectral decomposition (in this case the DFT matrix due to simplicity). Furthermore, the activation function of the spectral layer should compute a non linear activation function, such as the amplitude or phase of the spectrum. This prompts the model to work in a spectral space that is convenient to the periodicity that is encoded by the signals in this study.

However, initializing and modifying subsets of both $\mathcal{M}$ and θ is a rather restrictive

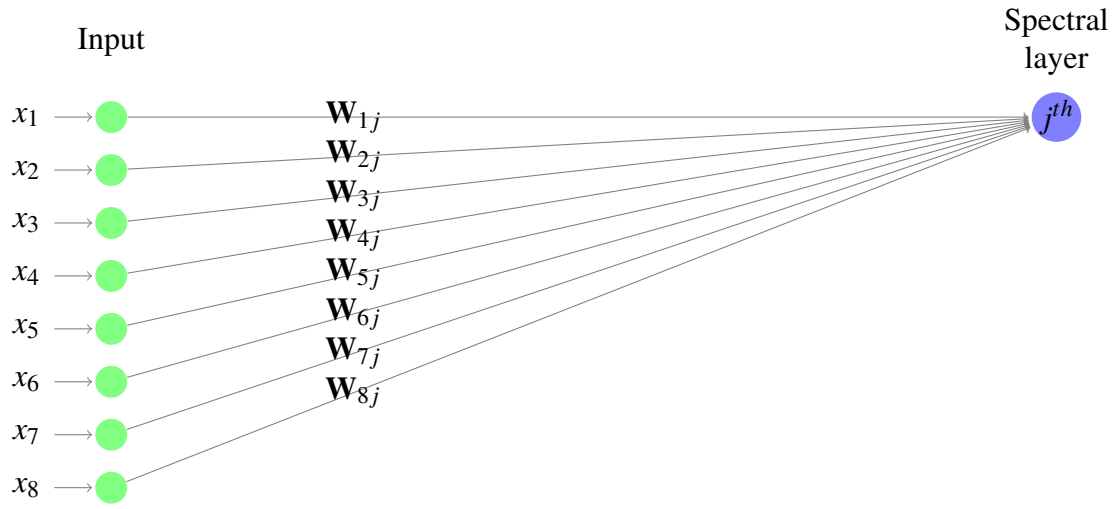
procedure, therefore it is a good idea to allow the MLE procedure interact with these initializations. This ensures that the initializations fit the data and the classification task.

8.2 Architecture of the Spectral Layer

The spectral layer is introduced as a complete (forwards-backwards pass), fully connected layer for an ANN. The initial set up is taken from [29], where this paper processes the DFT of an input sequence by initializing the weight matrix as the DFT matrix. This is done for purely computational purposes, and therefore the backpropagation and gradients of the layer are not dealt with.



As shown above the spectral layer has a fully connected architecture, which is equivalent to that of a regular fully connected layer. Here the author has chosen to showcase the connections and initializations by using an input vector of size eight since the original sizes will not be visible. There is only initialization difference from a fully connected layer. The difference is that the weights in the spectral layer are initialized (the weight matrix) to be the elements of the DFT matrix. This was discussed in the previous section. The initialization is demonstrated by picking the j^{th} hidden unit :



Here the DFT matrix has its standard definition of :

$$\mathbf{W}_{kj} = \frac{e^{\frac{2\pi kji}{N}}}{\sqrt{N}}$$

This initialization procedure is then carried out for every unit in the hidden layer. This ensures that all the connections are initialized such that a Fourier transform is computed when run across this layer.

8.3 Activation Function

Each of the Fourier coefficients are numbers in $\mathbb{C}$ or $\mathbb{R}^2$.

The $\mathbb{R}^2$ representation of complex numbers, requires twice the number of hidden units when representing these complex numbers in a structure that is suitable for a NN layer. The information encoded by these complex numbers is as follows:

$|\mathcal{F}\{x\}(\omega)| = (\mathcal{R}e\{\mathcal{F}\{x\}(\omega)\}^2 + Im\{\mathcal{F}\{x\}(\omega)\}^2)^{1/2}$ The amplitude of a coefficient.

$$\Phi\{\mathcal{F}\{x\}(\omega)\} = tan^{-1}\left(\frac{Im\{\mathcal{F}\{x\}(\omega)\}}{\mathcal{R}e\{\mathcal{F}\{x\}(\omega)\}}\right) \quad \text{The phase of a coefficient.}$$

The amplitude is normally referred to as the Fourier spectrum. It encodes *w.l.o.g.* the strength of a particular harmonic in the original signal. The phase of the spectrum encodes how a particular harmonic is shifted within the original signal. In a lot of applications, phase is discarded. Even though the phase contains important information, it is outside the scope of this project to interpret results coming from phase and is therefore discarded [8].

After discarding the phase one is left with the activation function as the amplitude of the Fourier coefficients. This function is applied to the hidden nodes in the Fourier Spectrum layer. Since the activation function is the amplitude of the Fourier coefficients, the doubling of hidden units no longer takes place.

It is also important to mention that when using inputs (which is the case here), there exists a symmetry such that approximately half of the coefficients are equal. This redundancy can be addressed by limiting the columns of the DFT matrix between $[1, \lceil N/2 \rceil - 1]$. This study uses the full DFT matrix since this is a more general formulation. Even though duplicates create redundancies, they should not have a major affect on the learning process of the NN.

8.4 Gradients

As described in the chapter above it is possible to express the DFT of an input signal (vector) $\vec{x}$ as :

$$\mathcal{F}\{\vec{x}\} = \mathbf{W}\vec{x} \quad s.t. \, \mathbf{W}_{kl} = \frac{e^{\frac{2\pi kli}{N}}}{\sqrt{N}}$$

In addition to this orthogonal transformation performed by the spectral layer, each of the Fourier components experiences the non linearity described in the previous chapter. This non-linearity calculates the amplitude of the component:

$$h(\mathcal{F}\{\vec{x}\}_l) = |\mathcal{F}\{\vec{x}\}_l|, \quad h : \mathbb{C} \to \mathbb{R}$$

Furthermore, to ease the calculation of derivatives. One can map the vectors in the complex plane to vectors in $\mathbb{R}^2$. To motivate this mapping one can show how the fundamental operations in $\mathbb{C}$ hold in $\mathbb{R}^2$.

Define $\phi : \mathbb{C} \to \mathbb{R}^2, \phi(x+iy) = (x,y)^T$.

Addition:

$$z_1 + z_2 = (x_1 + x_2) + (y_1 + y_2)i$$
$$\phi(z_1) + \phi(z_2) = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix} = \phi(z_1 + z_2)$$

Conjugation:

$$\overline{x+iy} = x - iy$$

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \phi(z_1) = \phi(\overline{z_1})$$

Multiplication:

$$z_1 z_2 = x_1 x_2 - y_1 y_2 + (y_2 x_1 + y_1 x_2) i$$

$$\phi(z_1 z_2) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \{\phi(z_2) \otimes \phi(z_1)\} \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \{\phi(z_2) \otimes \phi(z_1)\} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\phi(z_1 z_2) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \begin{pmatrix} x_1 & y_1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \begin{pmatrix} x_1 & y_1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\phi(z_1 z_2) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 x_2 & y_1 x_2 \\ y_2 x_1 & y_1 y_2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} x_1 x_2 & y_1 x_2 \\ y_2 x_1 & y_1 y_2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\phi(z_1 z_2) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} y_1 x_2 \\ y_1 y_2 \end{pmatrix} + \begin{pmatrix} x_1 x_2 \\ y_2 x_1 \end{pmatrix}$$

$$\phi(z_1 z_2) = \begin{pmatrix} -y_1 y_2 \\ y_1 x_2 \end{pmatrix} + \begin{pmatrix} x_1 x_2 \\ y_2 x_1 \end{pmatrix}$$

$$\phi(z_1 z_2) = \begin{pmatrix} x_1 x_2 - y_1 y_2 \\ y_2 x_1 + y_1 x_2 \end{pmatrix}$$

Using this mapping $\phi : \mathbb{C} \to \mathbb{R}^2$ it is possible to split the weight matrix into $\mathbf{W}_{\mathbb{R}} = \mathcal{R}e\{\mathbf{W}\}, \mathbf{W}_{\mathbb{I}} = Im\{\mathbf{W}\}$ which allows the activation function to be re-expressed:

$$h(\mathcal{F}\{\vec{x}\}) = ((\mathbf{W}_{\mathbb{R}}\vec{x})^2 + (\mathbf{W}_{\mathbb{I}}\vec{x})^2)^{1/2}$$

Where the powers are applied element wise. This formulation is equivalent to that of the absolute value of the DFT. Here the derivatives are expressed in terms of the new weight matrices. For a given activation function j:

$$\frac{\partial h(a_j)}{\partial \mathbf{W}_{\mathbb{R}jl}} = \frac{\partial (\overbrace{(\mathbf{W}_{\mathbb{R}l}\vec{x})^2 + (\mathbf{W}_{\mathbb{I}l}\vec{x})^2}^{a_j})^{1/2}}{\partial \mathbf{W}_{\mathbb{R}jl}}$$

Applying the chain rule:

$$\frac{\partial h(a_j)}{\partial \mathbf{W}_{\mathbb{R}jl}} = \frac{\partial a_j^{1/2}}{\partial a_j} \cdot \frac{\partial a_j}{\partial \mathbf{W}_{\mathbb{R}jl}} = \frac{1}{2} a_j^{-1/2} \cdot \cancel{2} x_l(\mathbf{W}_{\mathbb{R}l} \cdot \vec{x}) = h(a_j)^{-1} x_l(\mathbf{W}_{\mathbb{R}l} \cdot \vec{x})$$

And equally for the imaginary counterpart:

$$\frac{\partial h(a_j)}{\partial \mathbf{W}_{\mathbb{I}jl}} = h(a_j)^{-1} x_l(\mathbf{W}_{\mathbb{I}l} \cdot \vec{x})$$

Both of these scalar values are constituents of the gradients. They will be used to update the elements of each of the weight matrices in the spectral layer when back-propagating the error from the layers above.

8.5 Vectorization

This section will show how the gradients derived in the previous section can be efficiently compacted into a matrix form (vectorized [26]). This allows for an efficient implementation in tensor based libraries (in this case numpy).

For example, using the real component from the gradients it is possible to calculate :

$$\frac{\partial h(a_j)}{\partial \mathbf{W}_{\mathbb{R}jl}} = h(a_j)^{-1} x_l (\mathbf{W}_{\mathbb{R}l} \cdot \vec{x})$$

This is for a given component jl in the weight matrix. It is possible to obtain the expression to update column l of the weight matrix by removing the index j :

$$x_l (\mathbf{W}_{\mathbb{R}} \vec{x} \circ h(\vec{a})^{-1})$$

Furthermore, it is also possible to remove the index l which corresponds to the input dimensions of the layer :

$$(\mathbf{W}_{\mathbb{R}} \vec{x} \circ h(\vec{a})^{-1}) \otimes \vec{x}$$

The rows of the matrix produced by the above outer product, contain single elements which are the derivatives obtained in section four of this chapter.

From the maximum likelihood estimate, updating the gradients for the whole data set $\mathcal{X}$ (every row is a data point) consists of summing the gradient contributions from each data point (as shown in the equation above).

$$\sum_{n=1}^{N} (\mathbf{W}_{\mathbb{R}} \vec{x}_n \circ h(\vec{a}_n)^{-1}) \otimes \vec{x}_n$$

Matrix multiplication can be reformulated as the sum of outerproduct of the columns in the first matrix vs the rows in the second. Using A_n as notation for specifying the nth row of matrix A, two n by n matrices can be multiplied in the following manner:

$$AB = \sum_{n=1}^{N} (A^T)_n \otimes B_n$$

Using this reformulation of matrix multiplication it is possible to rewrite the gradient updates in a compact matrix form:

$$(\mathbf{W}_{\mathbb{R}}X^T \circ h(\mathcal{A})^{-1})X^T, \quad \mathcal{A} = (\mathbf{W}_{\mathbb{R}}X^T)^2 + (\mathbf{W}_{\mathbb{I}}X^T)^2$$

and for the complex matrix:

$$(\mathbf{W}_{\mathbb{I}}X^T \circ h(\mathcal{A})^{-1})X^T, \quad \mathcal{A} = (\mathbf{W}_{\mathbb{R}}X^T)^2 + (\mathbf{W}_{\mathbb{I}}X^T)^2$$

This representation provides a speed boost when implementing the gradient update. This increase in speed occurs because numpy Hadamard and matrix multiplications are much faster than for loops written directly in python.

8.6 Backpropagation

Backpropagation is implemented so that this layer can be used after another hidden layer (experiments like this are beyond the scope of this project).

In order to back propagate $\delta^{layer\_above} \circ h(\mathcal{A})^{-1}$ is projected onto the dimension of the lower layer [15]. Since there are two weight matrices, this means that the number of neurons is doubled. Therefore, the real and imaginary components of the gradient add up at the neuron in the layer below. This yields the following expression for the backpropagation:

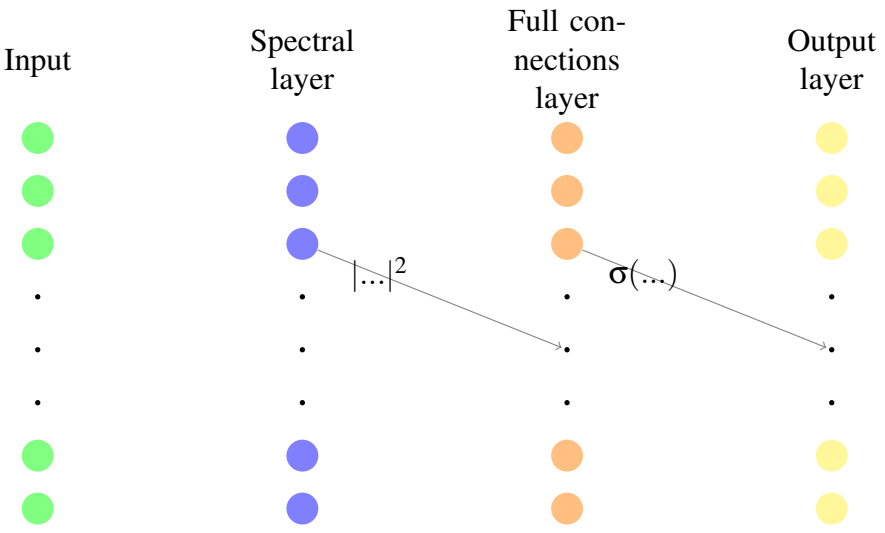
$$\delta^{layer\_above} \circ h(\mathcal{A})^{-1}(\mathbf{W}_{\mathbb{I}} + \mathbf{W}_{\mathbb{R}}) \tag{8.1}$$

Where δ^{layer_above} is the error signal from the layer above the spectral layer.

8.7 Final Models Architecture and Results

8.7.1 Final MLP (Spectral MLP)

The structure of the final MLP in which the spectral layer was tested was the following:



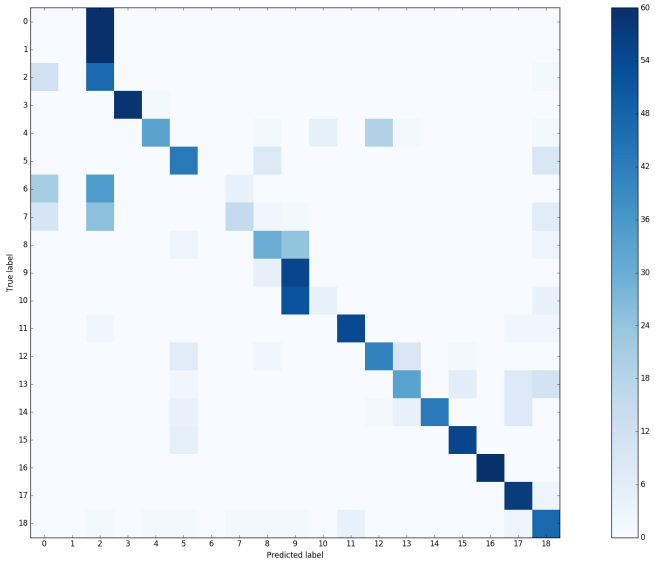


Figure 8.1: Confusion Matrix for final ConvNet

A high miss-classification occurs at class 2 (lying on back). Other than this, the confusion matrix has a good proportion of its density lying along the diagonal. This proportion along the diagonal is greater than the proportion in the baseline MLP with the Fourier Spectrum inputs.

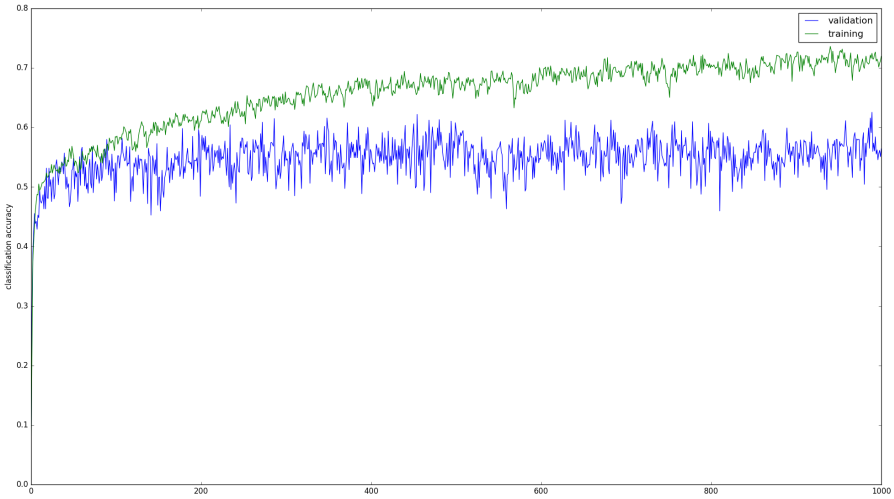


Figure 8.2: Training and Validation Accuracy per Epoch

The Spectral MLPs training and validation accuracies vary less erratically than its corresponding Fourier Spectrum baseline. It also achieves a better performance and does not overfit like the time-basis baseline MLP.

accuracy on test set	mean log likelihood
0.59298245614	-0.519292860747

Table 8.1: Results for Spectral MLP on Test Set (after 1000 epochs)

The final results show a significant improvement in accuracy going up by 10% when classifying on the test set.

A final experiment was carried out to account for the stochastic element of the gradient descent procedure. This was done by training and testing the model over ≈ 200 different random initialization points (random seeds). A histogram of the test accuracies obtained were plotted per model [10]

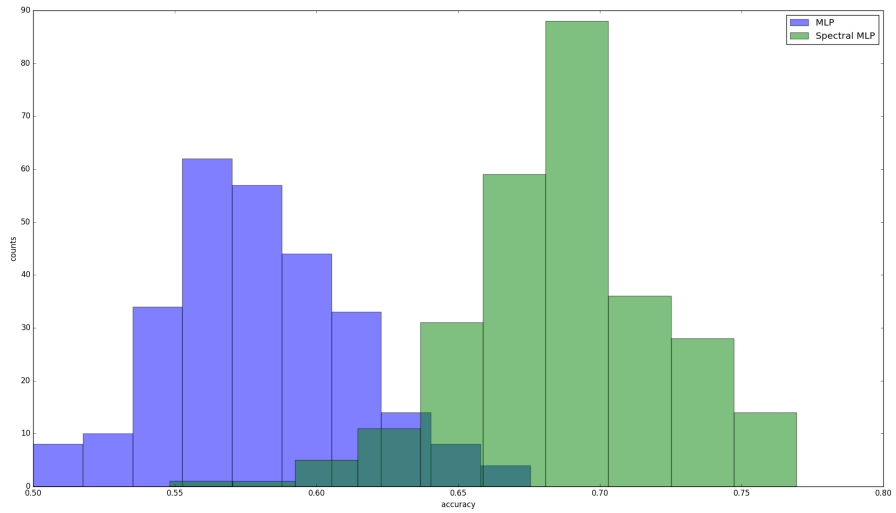


Figure 8.3: 200 different initlialization points (500 epochs)

In Figure 8.3 it can be observed that the mode of the Spectral MLP is about 15% (0.15) higher in terms of accuracy than the mode of the baseline MLP with Fourier Spectrum inputs. The mass of the Spectral distribution lies in the range 65% to 77% meanwhile the mass of the baseline MLP with Fourier Spectrum inputs ranges between 50% and 65%. From these results it is clear that the Spectral MLP outperforms significantly its baseline standard MLP with Fourier Spectrum inputs.

Additionally this experiment allowed to place error bars on the validation for epoch j plots:

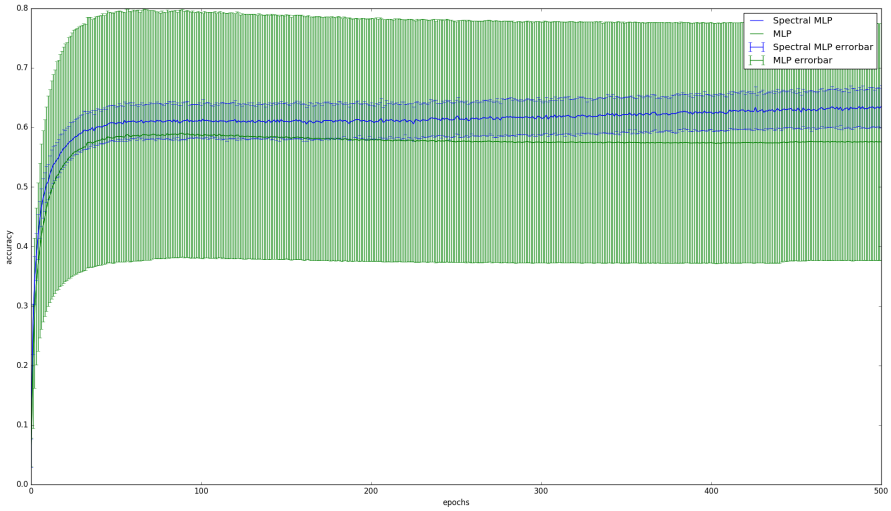


Figure 8.4: 200 different initialization points validation accuracy

Let X_a define a random variable that represents the accuracy of a particular model over different initializations.

Errors bars in Figure 8.4 for epoch j are computed in the following manner:

$$\sqrt{Var[X_a|Epoch = j]}$$

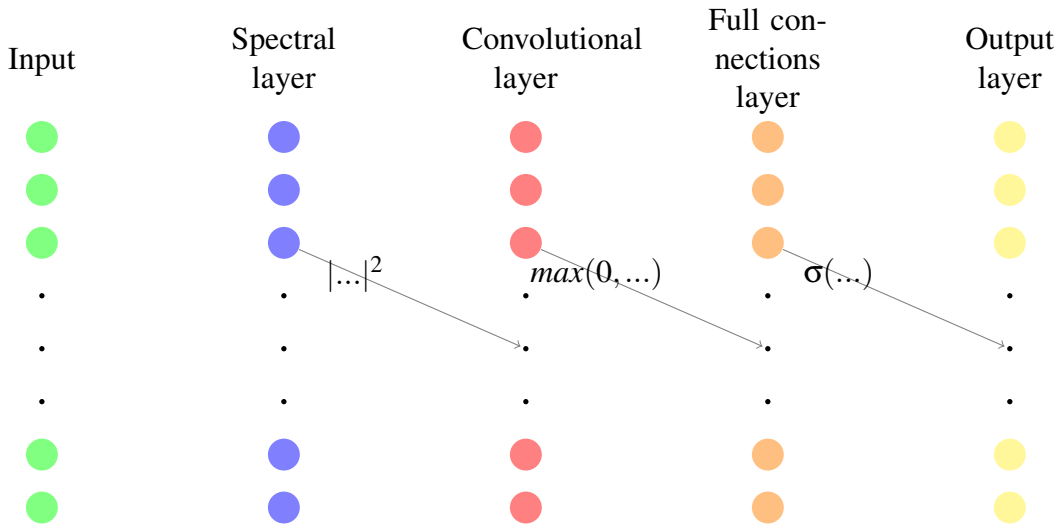
And each data point in Figure 8.4 for epoch j:

$$E[X_a|Epoch = j]$$

From the results in Figure 8.4 it can be observed that the spectral classifier's validation accuracy increases steadily and with a very low variance across different iniializations in time. The standard MLP architecture stops increasing at approximately 200 epochs and has a very high variance across different initializations.

8.7.2 Final CNN (Spectral CNN)

It is possible to illustrate the architecture of the best performing CNN derived empirically in the following diagram:



The functions drawn on top of the one connection displayed in the diagram represent the activation function applied to the layer. Traditionally a CNN is composed of a set of convolutional layers folllowed by an optional maxpool layer. The convolutional stack is usually fed into a standard MLP architecture, with rectified linear activation (ReLu) units [17]. However in this study it was empirically determined that for the purpose of this experiment sigmoidal activation units outperformed ReLu ones. Additionally, the networks performed better without intermediate max-pooling layers after the convolutional layers.

Convolution is a blurring (smoothing) like operation. Therefore, applying a convolutional layer to the Fourier Spectrum may be a form of noise removal from the spectrum itself.

8.7.2.1 Results

The density of the confusion matrix lies in the diagonal more than any other model in this study. There are some randomly dispersed small miss-classifications outwith the diagonal. Although, these miss-classifications are much less common than the corresponding Baseline CNN with Fourier Spectrum basis.

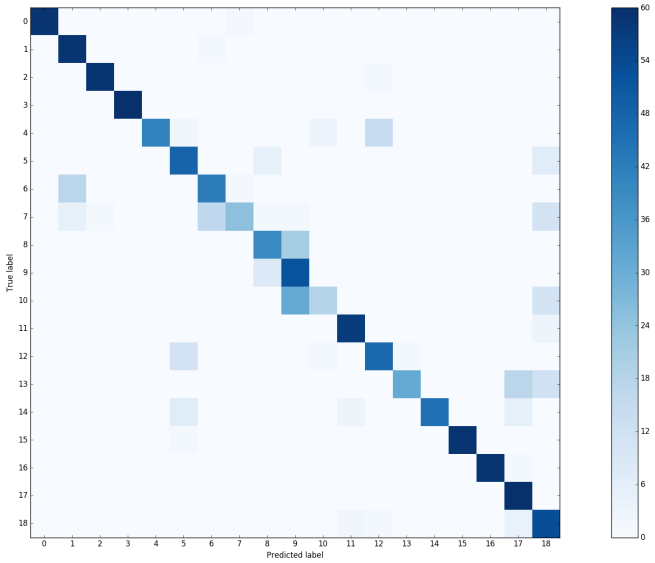


Figure 8.5: Confusion Matrix for final CNN (166 epochs)

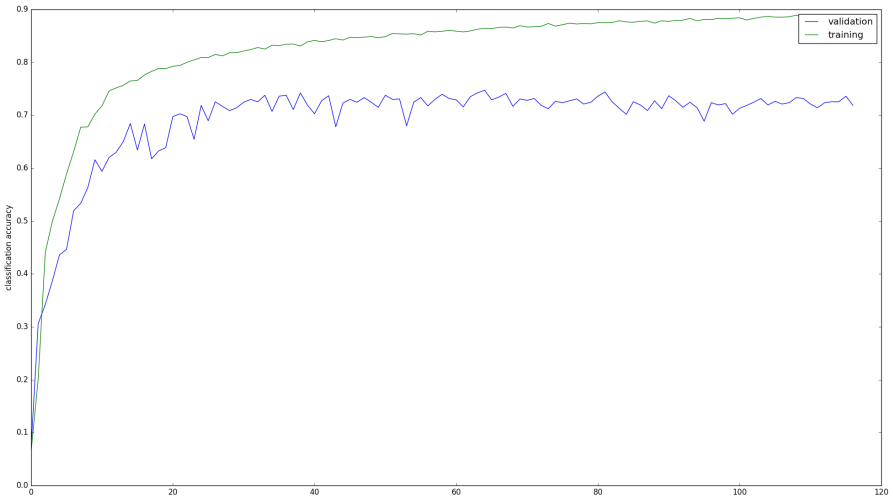


Figure 8.6: Training and Validation Accuracy per Epoch

From analysing this plot, it is clear that this model overfits much less than its counterpart baseline. The baseline has a significantly lower performance over the validation set and higher in the training set.

accuracy on test set	mean log likelihood
0.800877192982	-0.39038107006

Table 8.2: Test set results on spectral CNN (after 116 epochs)

Final results shows that the Spectral CNN model beats its baseline in terms of accuracy on the test by 3.3%. Although, it must be noted that this model trained on 116 epochs to reach this accuracy. Whereas, its baseline is trained on 300 epochs. Therefore, the baseline not only performs more poorly, but it also learns at a much slower rate.

Furthermore, a final experiment was carried out to account for the stochastic element of the gradient descent procedure. This was done by training and testing the model over ≈ 200 different random initialization points (random seeds). A histogram of the test accuracies obtained were plotted per model [10]

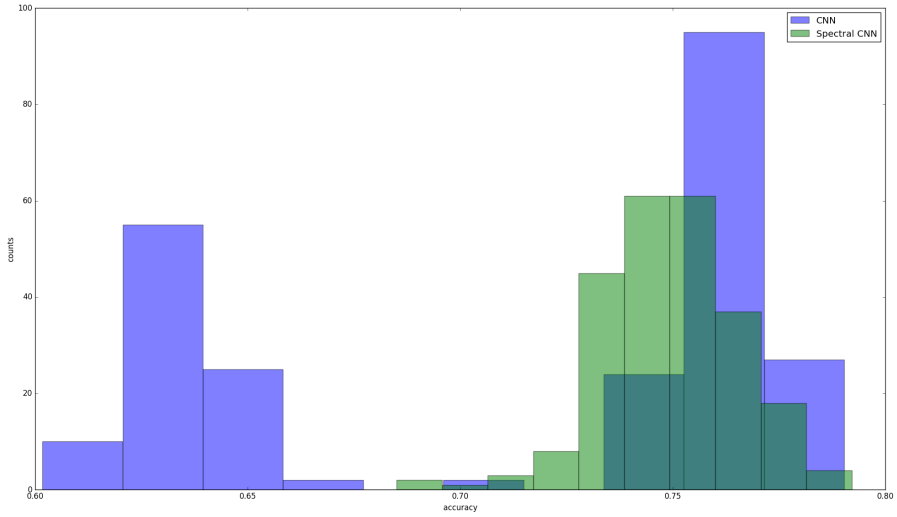


Figure 8.7: 200 different initialization points (116 epochs)

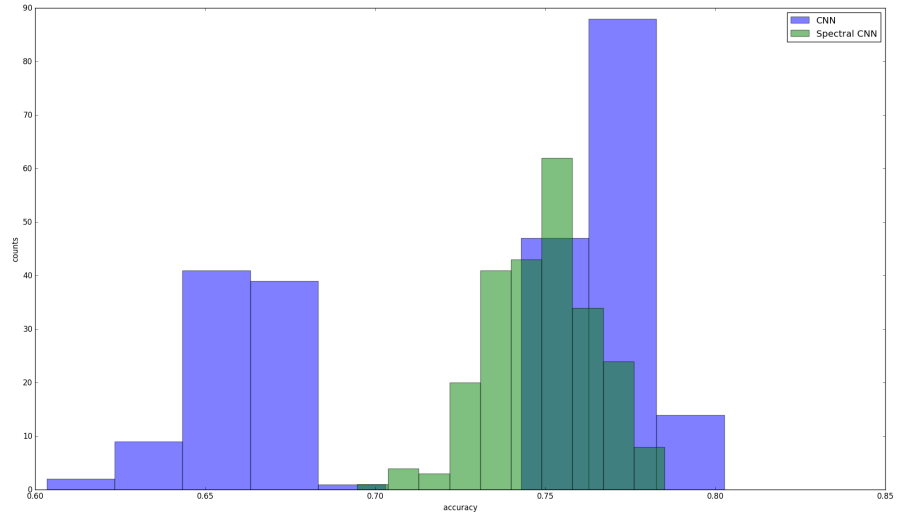


Figure 8.8: 200 different initlialization points (500 epochs)

The regular CNN with Fourier Spectrum inputs exhibits a bimodal distribution over its performance accuracy given different starting points. The author's interpretation of this is that the regular CNN with Fourier Spectrum inputs converges almost equally to two different local optimas. One with a good performance and the other with a poor performance as can be seen from Figures 8.7 and 8.8.

The Spectral CNN model seems to only land in a basin of attraction which corresponds to one local minima which has high accuracies. The variance of the distribution of the Spectral CNN is slightly higher than that of the well performing mode of its baseline. The mode of the spectral CNN histogram is slightly lower than the high mode of the baseline histogram (blue histogram). Although, the Spectral CNN is much more reliable since its overall center of mass and variance are much higher. Additionally, from examining the two plots it is clear that the spectral CNN learns faster than its baseline. The Spectral Layer CNN performs better on 116 than 500 epochs as can be observed from the histograms.

8.8 Visualizing the Fourier Operator (DFT Matrix)

This section will illustrate how the MLE affects the weights of the spectral layer when training the two final models.

8.8.1 Spectral MLP

8.8.1.1 DFT Matrix

The following plots display the DFT matrix after the MLE procedure:

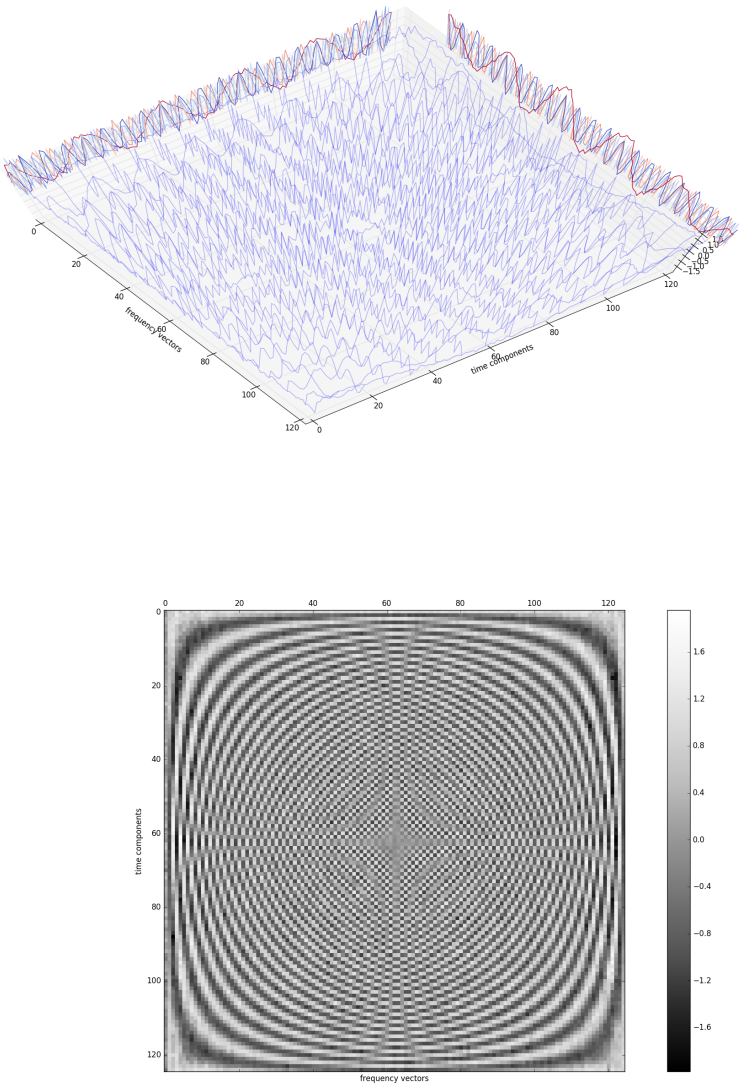


Figure 8.9: Post MLE Spectral MLP weight matrices in 3d and contour views

It can be seen from the projections in the 3D plot that the originally smooth sinusoid's, which were outlined in the background reading, have become distorted due to the MLE procedure when training the network. As was argued before in the theoretical motivation section of this chapter, propagating these errors in the DFT weights via the MLE, may make the DFT fit both the data and the classification task more accurately.

8.8.2 Difference from Original DFT

This section aims to quantify how much the MLE has changed the DFT matrix.

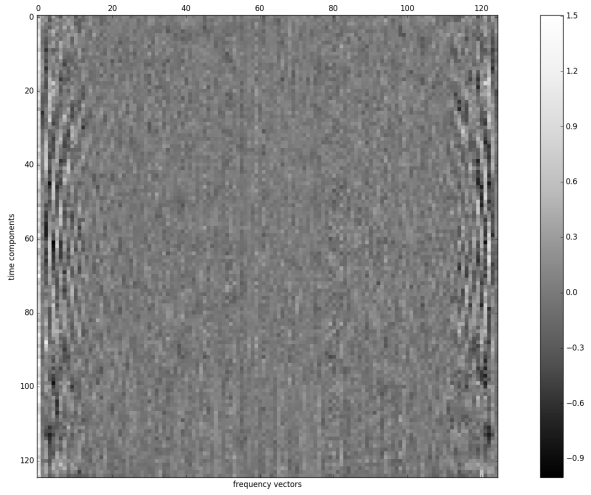


Figure 8.10: $\mathcal{R}e\{\mathbf{W}_{original} - \mathbf{W}_{MLE}\}$

The difference between the original DFT matrix and the post MLE matrix is that it exhibits some of the sinusoidal patterns shown by the original DFT matrix in some regions. This could mean that some sections of the sinusoidal vectors are transformed by the MLE by shifts in their amplitudes.

8.8.3 Unitary Preservation

The DFT matrix was multiplied by its complex conjugate transpose in order to assess if its unitary property is preserved. This can be determined from Figure 8.11, where the visualization shows it is not preserved. Nonetheless, most values that are not on the diagonal are on average 0, and the diagonal itself remains close to 125. Thus, the unitary property is not heavily changed by the MLE procedure.

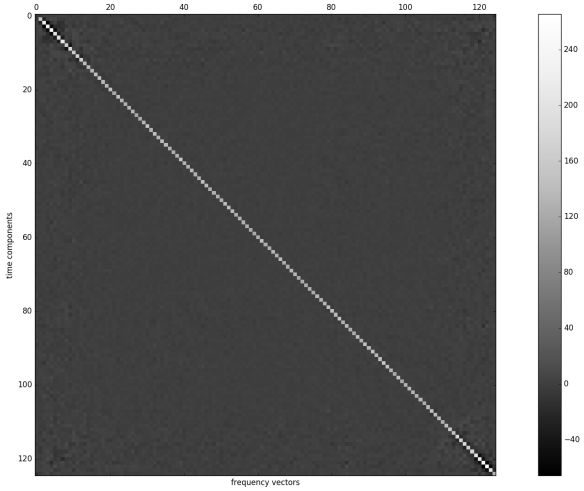


Figure 8.11: Unitary transformation check $\mathcal{R}e\{\mathbf{W}_{MLE}\mathbf{W}_{MLE}^{\dagger}\} \approx N\mathbb{I}$

8.8.4 Spectral CNN

8.8.4.1 DFT Matrix

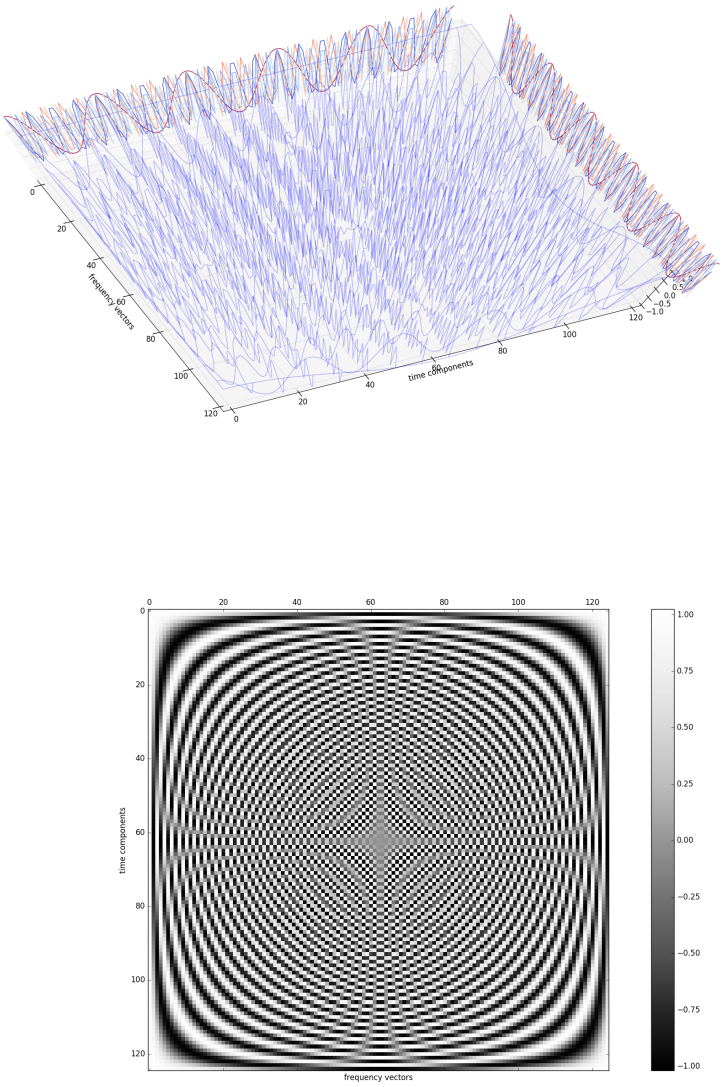


Figure 8.12: Post MLE Spectral CNN weight matrices in 3d and contour views

8.8.5 Difference from Original DFT

It can be noted that some of the differences happen along vertical lines. Vertical lines represent DFT frequency basis vectors. It can be seen that the biggest differences happen along bands of vectors. Setting this in context to the MLE, this implies that the negative log likelihood (or the error) is amplified the most along certain DFT components (each hidden unit in the spectral layer constitutes a DFT component). This shows

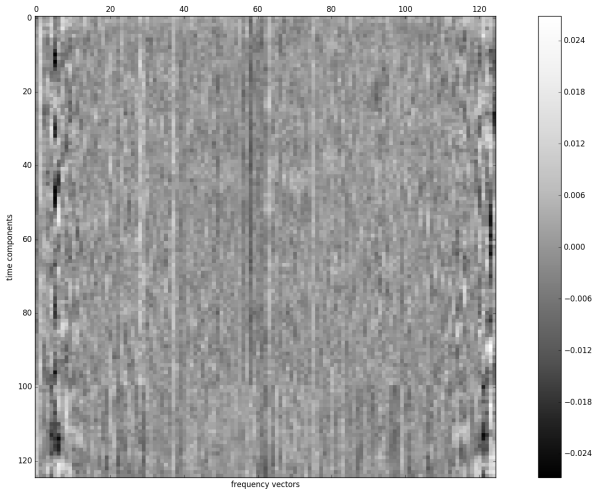


Figure 8.13: $\mathcal{R}e\{\mathbf{W}_{original} - \mathbf{W}_{MLE}\}$

that most of the change is happening along certain frequency bands. This means that certain spectral hidden units were required to slightly adapt in the optimization process, this may mean that some particular frequencies were more important than others in the classification process.

8.8.6 Unitary Preservation

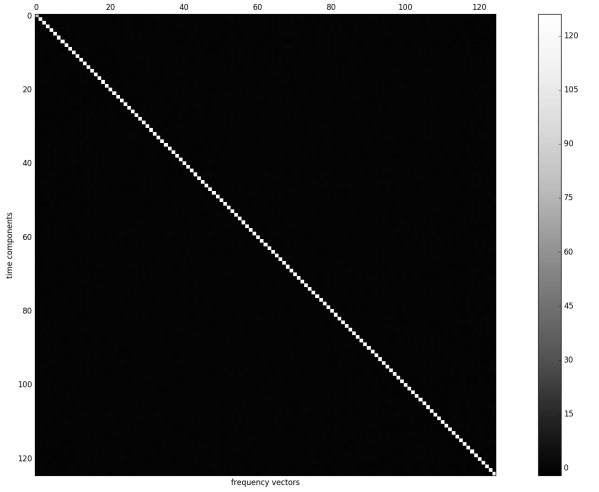


Figure 8.14: Unitary transformation check $\mathcal{R}e\{\mathbf{W}_{MLE}\mathbf{W}_{MLE}^{\dagger}\} \approx N\mathbb{I}$

The Unitary property of the DFT matrix seems to be intact after the MLE procedure

8.9 Parallelization of DFT

Paper [29] shows that the representation can be implemented into neurohardware. This allows one to take advantage of the parallel architecture of the spectral layer. Therefore, both computation of spectral coefficients and the updating of elements in the weight matrix can be done in parallel.

8.10 Previous Relevant Work

The most relevant works in terms of Fourier analysis based neural networks can be split into two categories, one being the use of sinusoidal activation functions and Fourier transform based weightings. The other is more closely related to this study which presents an adaptive Fourier transform similar to the spectral layer in this study.

8.10.1 ADFT and Fourier Spectrum Activation Units

In paper [20] the adaptive discrete time Fourier transform is introduced by embedding a Fourier transform in a feed-forward neural network in the following diagram [20]:

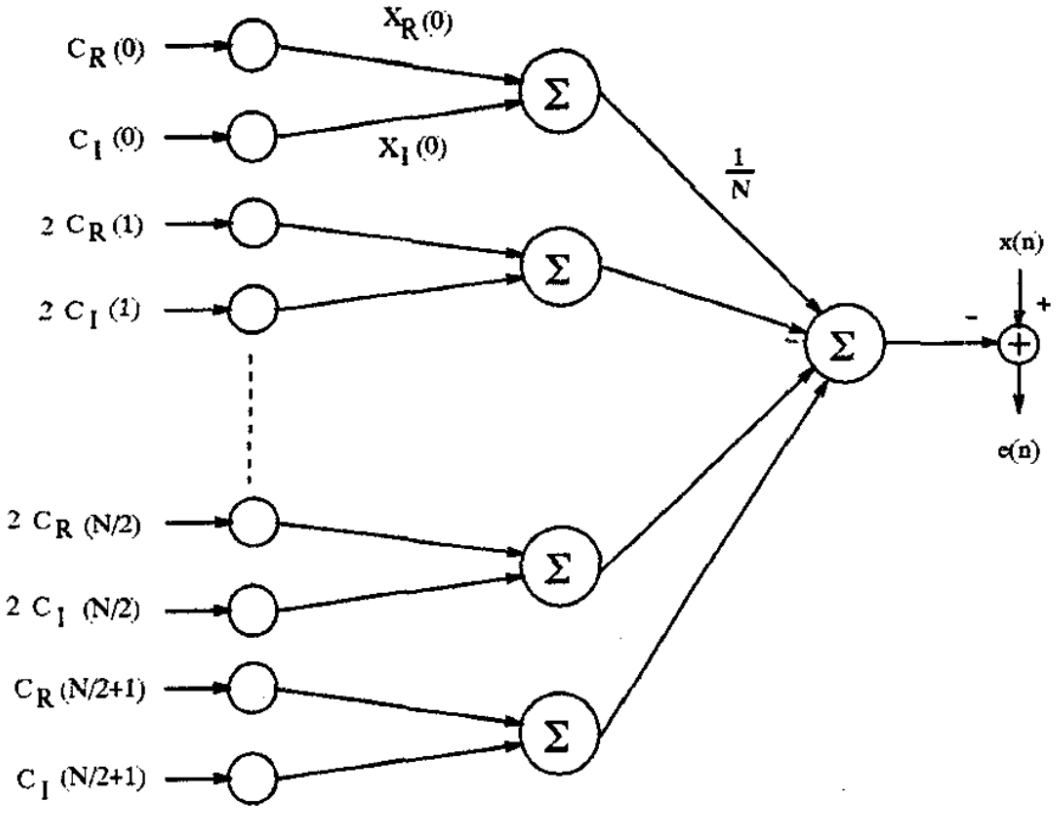


Fig. 1: Structure of ADFT using neural networks

The propogated error arises from a least means square based cost function. This occurs because this paper is carrying out a regression based task for analysing the pitch of speech signals. The main differences between the spectral layer presented in this study and the one used in paper [20] are:

1. The task in this study is a classification and the network is propagating a cross-entropy error.
2. The activation functions for the Spectral layer are non linear absolute value functions. These functions take the magnitude of the Fourier coefficients.

Another relevant paper for this section is [21]. In this paper the author uses absolute value activation functions on a filtered Fourier transform input. Unlike the ADFT paper [20] it does not back-propagate the errors from the training of the network to the DFT operator.

The spectral layer presented in this chapter is a combination of the ADFT in [20] and the absolute value activation units in [21].

A more mathematically motivated paper that also develops the ADFT-like transform is [27]. In this paper the forward computation of the neuron is modelled as special case of an integral transform. These are traditionally used to solve dynamic systems such as differential equations [13]:

$$f(\vec{x}) = \int_D c(\vec{y})\phi(\vec{x},\vec{y})d\vec{y}$$

Note the original paper has a repeated typo where $c(\vec{y})$ is written as $c(\vec{x})$.

The interpretation of this integral is that it maps $\vec{y}$ from the input layer space (or output of a layer below) to a new basis, namely $\vec{x}$. In this representation $\phi(\vec{x},\vec{y})$ is known as the kernel and it defines how the change of basis is carried out. For example kernel of the form e^{-iyx} yields a continuous Fourier transform. Discretizing this integral models how neural networks forward propagate from one layer to another. As mentioned earlier a Fourier transform can be carried out by this integral given the appropriate choice of kernel. In the study of differential equations this is a common choice since it provides an algebraic basis in which to solve differential equations. In paper [27] an ADFT [20] like transform is devoloped by taking only the real components of the DFT produced by each neuron. This choice potentially creates a loss of information.

8.10.2 Sinusoidal Activation Functions

Some of the work presented in [11] shows the use of sinusoidal (Fourier operator like) activation functions which is a common used when looking into the literature available for Fourier Neural Networks. In addition to the sinusoidal activation functions, the algorithm used to set up the Fourier Neural Networks in [11], initializes a subset of the weights to be the DFT of the input vector.

Chapter 9

Model Compression

The distillation of knowledge method is discussed in the background chapter [2].In order to compress the knowledge from two leg devices into a model that uses one leg device, the model compression was implemented in the following manner:

Algorithm 5 Distill Knowledge

Input: augmented training set with both devices $\{\vec{x}_k^{leg1}, \vec{x}_k^{leg2} | y_k\}_{k=1}^N$, Network topology $\mathbf{N}_{aug}$

Output: Network trained for smaller set $\{\vec{x}_k^{leg1}\}_{k=1}^N$

1.Train neural network on augmented training set giving N_{aug}

2.Predict Targets of augmented training set $\{\vec{x}_k^{leg1}, \vec{x}_k^{leg2}\}_{k=1}^N$ with $\mathbf{N}_{aug}$ (argmax of softmax outputs) namely $\{y_k^{pred}\}_{k=1}^N$

3. Create new training set with lower dimensionality $\{\vec{x}_k^{leg1} | y_k^{pred}\}_{k=1}^N$ and predicted output

4. Compress the knowledge into a new model $\mathbf{N}_{comp}$ (with same topology as N other than the input layer) by training on $\{\vec{x}_k^{leg1} | y_k^{pred}\}_{k=1}^N$

Return $\mathbf{N}_{comp}$

This procedure was performed on the two Fourier Spectrum basis baseline models on both legs, since the shapely value analysis implied that the leg placements were the top two devices.

9.1 MLP Distilled Model Results

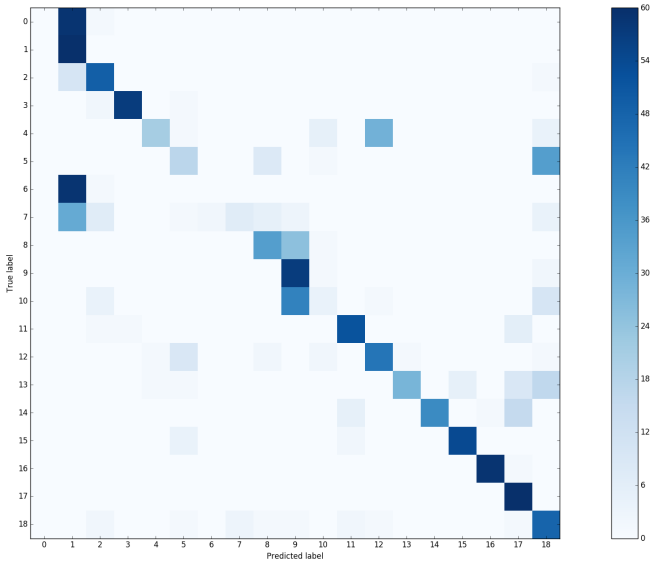


Figure 9.1: Confusion Matrix for MLP distilled model

The confusion matrix contains the majority of its density along the diagonal, with a relatively heavy miss-prediction rate for class one (standing).

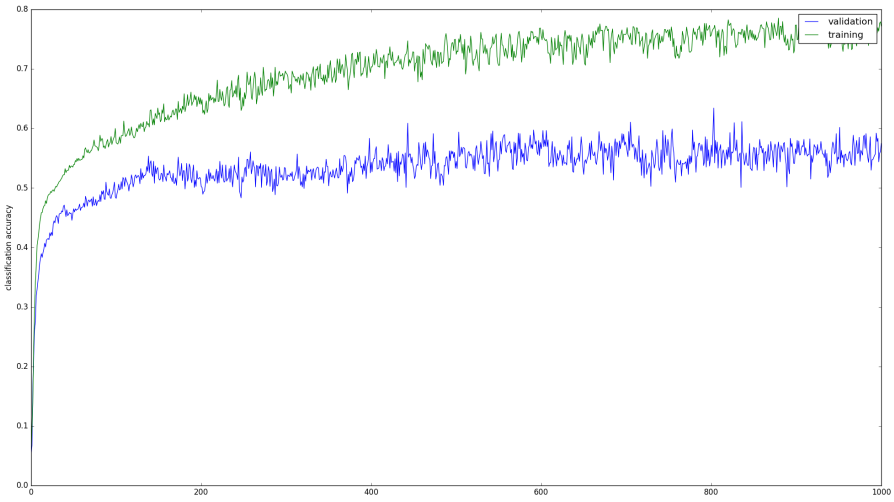


Figure 9.2: Training and validation per epoch for MLP distilled model

Both training and validation accuracy grow together in a rather nice and very similar looking pattern in the MLP distilled model.

Accuracy	Mean log likelihood
0.605263157895	-0.459008427654

Table 9.1: Distilled MLP results on test set (after 1000 epochs)

The final results show that this model outperforms significantly (10.5 %) its baseline (MLP with Fourier Spectrum inputs) with one device nonetheless its performance is quite close to the Spectral MLP with one device. A combination of Spectral MLP's and distillation of knowledge could possibly yield a very good predictor.

9.2 CNN Distilled Model Results

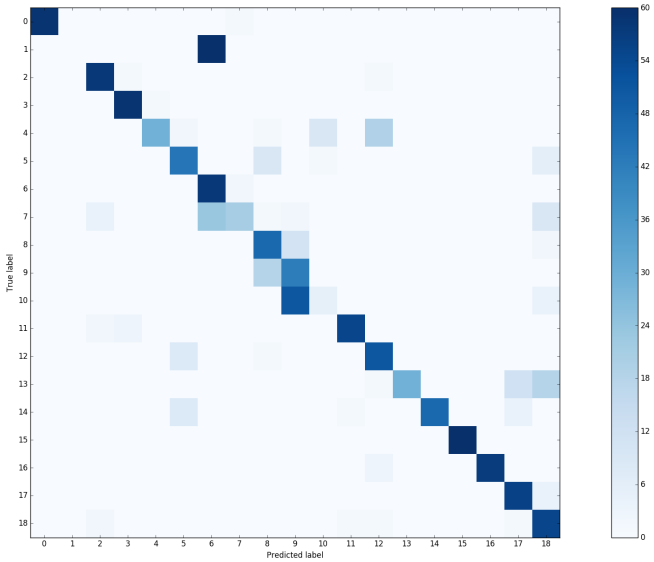


Figure 9.3: Confusion Matrix for CNN distilled model

The results in Figure 9.3 and table 9.4 show that the model has a reasonable accuracy and possibly too high a confidence. However, it does not outperform the baseline (CNN with Fourier Spectrum inputs) which only uses the knowledge of one device and therefore, there must be some sort of miss-information (communication) taking place when the knowledge from two accelerometers are compressed into one model. This implies that the method of distillation used here is flawed. This flaw may occur since the hard targets (original targets) were not mixed when distiling the knowledge in to the smaller model.

Accuracy	Mean log likelihood
0.729824561404	-0.148600612566

Table 9.2: Distilled CNN results on test set (after 500 epochs)

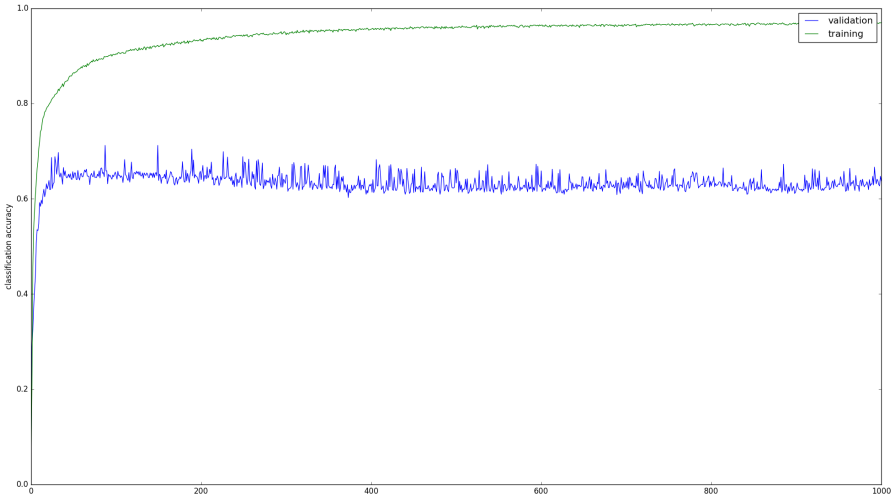


Figure 9.4: Training and validation per epoch for CNN distilled model

The model seems to heavily over-fit on the training set when examining the evolution of training and validation accuracy over time.

Chapter 10

Conclusion

10.1 Further Work

As mentioned in this study, phase was not used when constructing the spectral layer. As some of the tasks used in this data set are very similar (i.e. standing vs standing in a lift), information such as the phase may be very helpful when discriminating such classes. An alternate layer architecture can be derived, so that both the Fourier Spectrum and the phase are computed when forward propagating across this layer. This alternate method using the new layer architecture could be used as the basis of further research.

There exists a wide variety of spectral decompositions (i.e wavelet transform, short time Fourier transform, Hilbert transform). One could study how to decompose some of these transformations into a network layer architecture. Then one could also use them in a similar way to how the spectral layer was used in this study.

The distillation of knowledge method could be enhanced both by adjusting the temperature parameter and mixing both the soft and hard target with some mixing parameter α [14].

In order to achieve a better performance, a distillation of knowledge based method can be applied to ANN's that use a spectral layer as a preprocessing layer. The combination of these two methodologies (Distillation of knowledge and the Spectral layer) could yield a much stronger classifier.

10.2 Conclusion

This study introduced a spectral layer as a preprocessing layer to a neural network. This was done in order to classify periodic time-series based data using ANN's. The spectral layer initially computes a Fourier Spectrum, but is allowed to relax via the MLE when learning the training set. The spectral layer was compared with the standard MLP and CNN baseline models (with Fourier Spectrum inputs).

The performance of the spectral layer based ANN's significantly surpassed the performance of its corresponding baselines. This performance comparison can be seen by the results both classifiers exhibited over a number of different experiments. Therefore, this study concludes that the spectral layer based ANN's perform not only at higher accuracies, but are also more reliable when achieving these accuracies (given different initializations of the weight matrices in the subsequent layers of the ANN's).

A distillation of knowledge method was explored to address the compression of knowledge from a model that used two accelerometers into a mimic model which used one accelerometer. Distillation of knowledge applied to a the baseline MLP with Fourier Spectrum inputs yielded a well performing model relative to the baselines in this study. CNN based models overfitted when compressing the knowledge into the mimic model. The overfitting of the CNN models was possibly due to only using soft labels instead of the recommended weighting of both soft and hard labels [14].

Bibliography

[1] Kerem Altun, Billur Barshan, and Orkun Tunçel. Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recognition*, 43(10):3605–3620, 2010.

[2] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep?, 2014.

[3] Yoshua Bengio. Introduction to multi-layer perceptrons (feedforward neural networks). introduction to multi-layer perceptrons (feedforward neural networks) notes de cours ift6266 hiver 2010. 2 apr. 2010. web. 4 oct. 2014.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[6] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[7] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[8] A Dieckmann. Amplitude and phase of a discrete fourier spectrum. elsa, physikalisches institut der universitat bonn. 2011.

[9] Niklas Elmqvist, Pierre Dragicevic, and Jean-Daniel Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1539–1148, 2008.

[10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[11] Michael S Gashler and Stephen C Ashmore. Training deep fourier neural networks to fit time-series data. In *Intelligent Computing in Bioinformatics*, pages 48–55. Springer, 2014.

[12] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH*, pages 1756–1760, 2013.

[13] Sadri Hassani. *Mathematical physics: a modern introduction to its foundations*. Springer Science & Business Media, 2013.

[14] GE Hinton, Oriol Vinyals, and Jeff Dean. Dark knowledge. *Presented as the keynote in BayLearn*, 2014.

[15] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[16] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[17] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[18] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[19] Zdravko Markov and Ingrid Russell. An introduction to the weka data mining system. In *ACM SIGCSE Bulletin*, volume 38, pages 367–368. ACM, 2006.

[20] Munehiro Namba and Yoshihisa Ishida. Pitch synchronous fourier transform using neural networks. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 5, pages 2896–2899. IEEE, 1995.

[21] T Nguyen, J Kim, S Yang, and YONGGWAN Won. A multilayer neural network for classification of frequency information dominant patterns. In *Proceeding of the 15th International Conference on Advances in Neural Networks, Gdansk, Poland*, pages 62–68, 2014.

[22] Prof. John A. Peacock. School of physics and astronomy, fourier analysis.

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] R. Robu and V. Stoicu-Tivadar. Arff convertor tool for weka data mining software. In *Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on*, pages 247–251, May 2010.

[25] S. Sasikala, S. A. alias Balamurugan, and S. Geetha. An efficient feature selection paradigm using pca-cfs-shapley values ensemble applied to small medical data sets. In *Computing, Communications and Networking Technologies (ICCCNT),2013 Fourth International Conference on*, pages 1–5, July 2013.

[26] Mark Shabahi. A guide to auto-vectorization with intel c++ compilers.

[27] Adrian Silvescu. Fourier neural networks. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 1, pages 488–491. IEEE, 1999.

[28] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[29] Rosemarie Velik. Discrete fourier transform computation using neural networks. In *Computational Intelligence and Security, 2008. CIS'08. International Conference on*, volume 1, pages 120–123. IEEE, 2008.

[30] Eric W Weisstein. Unitary matrix. from mathworld–a wolfram web resource. *Acessado em*.

[31] Eric W Weisstein. Fourier matrix. from mathworlda wolfram web resource. *Acessado em*, pages 01–06, 2014.