Time-series Classification using Convolutional Neural Network with application to Financial Data

Klaudia Ludwisiak

Industry Supervisor: Mattia Fiorentini

Academic Supervisor: Simone Severini

Dissertation submitted in partial fulfillment

of the requirements for the degree of

MSc Computational Statistics and Machine Learning

of

University College London

Departments of Statistics and Computer Science

University College London

September 2017

I, Klaudia Ludwisiak, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Even though the use of Convolutional Neural Networks (CNNs) for time-series classification has not been widely explored, results reported thus far are very promising. In this work the most acclaimed approaches are reviewed and extend to the task of financial time-series classification. The data in question is minute-by-minute real-life crude oil price and volume, recorded over a series of anomalous events starting from 2010.

Particular focus is given to the FCN and ResNet architectures implemented by Weng et al. (2016) [1], currently holding the state-of-art performance in classifying benchmark data-sets. These models are recreated and used as the starting point for further improvements. For ease of comparison, models are validated against benchmark UCR data-sets prior to their application on the finical data. The recent work by Borovykh et al. (2017) [2] paired with inspiration from GoogLeNet [3] is used to develop a augmented FCN model which outperforms Wang's et al.(2016) implementation on the finical data classification task. The improved FCN model includes: a final *conv1x1* layer, adjusted filter sizes, dilation and dropout. In line with literature, despite employing residual connections the ResNet model does not outperform the FCN. Thus, less emphasis is placed on this architecture's development.

The complex, noisy nature of the financial data-set presents challenges not only for model training but also for evaluation. It is concluded that validation results are the most meaningful to judge true model performance and compare between all models. On this basis, the improved FCN architecture achieves a stable validation accuracy of almost 58%, which can be deemed a good result for the financial time-series classification task at hand.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my industry supervisor Mattia Fiorentini for facilitating this project and for his continuous support. His guidance, patience and immense knowledge helped me through the research and writing of this thesis. Additionally, I would like to thank the entire team at Cambridge Quantum Computing for their kind reception, provision of data and access to the firms computational resources. Without this support undertaking the project would not have been possible. Besides my industry supervisor and *CQC*, I would like to thank my academic supervisor Dr. Simone Severini for his insightful comments and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter explains the motivation behind this work based on recent literature and introduces the vast, and longstanding topic of artificial neural networks (sections 1.1 1.2). The key concepts and terminology behind time-series classification are also explained in section 1.4. This is followed by a problem formulation section, where the goals of this work are stated (section 1.5). A summary of the remaining chapters is also provided (section 1.6). The code supporting this thesis can be found on: `https://www.dropbox.com/sh/q6mc4yuhrjxldpr/AAAz8ri9XxcA50B9_qMMrpUPa?dl=0`

1.1 Motivation

Time series classification draws unceasing attention because of its broad applications across different fields, ranging from health and bio-informatics to finance. Therefore, the subject has been widely and continuously studied in the machine learning domain and many approaches have been developed for this task. Historically, the quality of results was improved at the expense of additional computational resources. Thus, state-of-art algorithms were difficult to scale to large data-sets - with more than a few hundred data points (Bagnall at al. 2016)[4]. However, recent breakthroughs in a sub-field of machine learning called Deep Learning (DL) have paved the way for a new generation of time-series classification approaches.

Over the past decades, the application of machine learning to real-world problems has greatly benefited from mining and storing large data-sets. Due to the

advances in cheap GPU processing power, the application of artificial neural networks (ANNs) to analyze ever larger data-sets, i.e. *big data*, across a vast range of domains has become common practice, also in light of recent successes in specific applications [5]. The most recent developments in the fields of ANNs are grouped under the name of *Deep learning*, which is a school of machine learning concerned with building very deep ANN architectures, comprised of many layers. Convolutional Neural Networks (CNNs) are one such type of architecture that has achieved unprecedented results in image and speech classification tasks. CNNs have advantages over other types of ANNs in that they share weights, substantially reducing the amount of parameters required to train the model compared to other types of deep neural networks. For this reason, they can either be cheaply scaled up to tackle more complex, larger data-sets, or applied successfully to relatively small ones [6], without suffering too much from *over-fitting*. CNNs exhibit more desirable properties, explained in further sections, which render this network type promising for application in Time-Series Classification tasks.

Some ANNs architectures have been successfully used to predict and classify sequential data, such as time-series. However, the application of CNNs in this context has been comparatively understudied. Therefore, this work will build on existing research in the area aiming to apply CNNs to financial time-series classification. Unlike many other time-series, financial signals are challenging to classify due to their unstructured and erratic nature. However, a strong motivation for their classification exists as patterns hidden in financial-time series are a source of valuable information about future price fluctuations.

1.2 Artificial Neural Networks (NNs) Overview

1.2.1 How NNs Work

Artificial neural networks, abbreviated to ANNs or NNs, are algorithms which loosely resemble the structure and function of a brain. In its simplest form a artificial neural network is a collection of neurons arranged in layers, with connections in between layers. Depicted in Figure 1.1, a simple feedforward NN consist of an

input layer, receiving raw data, followed by one or several hidden layers and a final output layer. The amount of layers in a NN model is commonly referred to as the depth of the model, and in many applications it has been shown that deeper models learn better [6].

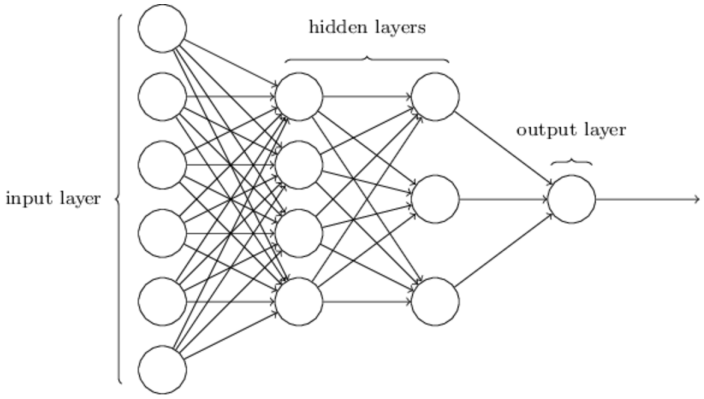


Figure 1.1: Schematic of a Basic Neural Network
Adapted from [7]

Much like a biological neuron, each artificial neuron receives input signals, processes those and outputs a single signal, this is depicted schematically in Figure 1.2. Within each neuron a linear transformation is performed calculating a weighted sum of inputs and a bias term. This is followed by a non-linear activation layer where various choices of the activation function exits and the best choice is problem specific [8]. At present, a popular activation function is the Rectified

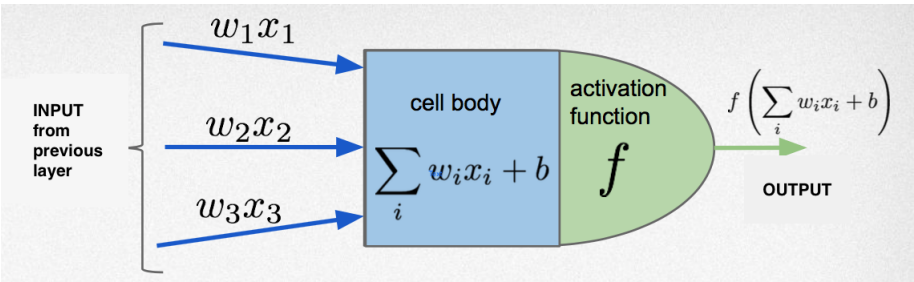


Figure 1.2: Schematic of a Basic Neuron
Adapted from [8]

Linear Unit (ReLU): $f(x) = max(x, 0)$, that is a piece-wise linear function. Typically, ReLU learns much faster than smoother non-linearities which were used in

the past[1].

Another important activation function employed in this work is the softmax: $f(x) = \mathbf{y} = softmax(\mathbf{x})$ such that $y = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ is the normalized exponential probability distribution over the K possible discrete values of y [9], which is the standard case of a classification task.

If neurons belonging to the same layer do not share connections, than the network is called a feedforward NN or Multi Layer Perception (MLP). In this network inputs only travel forward through the layers and no feedback loops exists. A cyclic network on the other hand, allows for connections between neurons of the same layer. A notable example of this is the recurrent neural network (RNN), which has been extensively developed and applied for classifying sequential data [9]. However, among several drawbacks RNNs are too expensive to implement on a large scale, and do not allow very deep architectures. Although cyclic networks will not be a focal part of this project, more detail on the RNNs is given in section 2.2.2.1.

What is commonly called *training* the network or *learning* is the process of finding values for all the model weights which minimize the objective function, yielding the *best* predictions. In a supervised learning setting, such as the one in this work, there exists a training data-set with known classification targets. This can be used to compute the objective, also called the loss function, by repeatedly passing information through the network in a feedforward manner and comparing network output with the target. After each iteration, the model loss is calculated as some measure of the difference between the correct and predicted results. This information is then passed back through the network from the outputs (i.e final layer) to the inputs (i.e 1st layer) in a procedure called backpropagation, which allows weight optimization and adjustment upon each iteration. The backpropagation procedure computes the gradient of the loss function with respect to the weights of consecutive model layers by applying the chain rule of derivatives [6]. One pass of all the training data through the network is referred to as an epoch and typically

[1] ReLU learns much faster than smooth activation functions because its gradient is fixed at either 0 or 1. Upon backpropagation, if a gradient is fixed at zero it remains so, this can interpreted as the neuron being *dead*. Fixing desired gradients at zero increases sparsity, which is desirable to reduce model complexity and prevent gradients from vanishing or exploding.

thousands of epochs are required to identify a satisfactory configuration of all the model weights. For further detail on the backpropagation mechanism please refer to source literature [10].

At this stage it should also be clarified that in supervised learning setting the data-set is typically split into three portions: training, validation and testing. The training set is the largest as this is the potion of the data used to train the model. The validation set is usually much smaller - circa 15% of the entire data set or less. This is used during training to evaluate the model performance and aid learning. The test set is never used for model training and instead it is used for the final model evaluation after training is completed [9].

1.2.2 Brief History

The concept of an artificial neuron dates back to 1958, when Rosenblatt introduced the famous Perceptron model for supervised learning of binary classifiers [11]. Much like the neuron described in Section1.2.1 above, the perceptron calculated a linear-combination of multiple-inputs, optionally followed by a non-liner transformation, to produce a single output. In the decades to follow the neuron building blocks were assembled together into the most basic artificial neural network called the Multi Layer Perceptron (MLP) (schematic in Figure 1.1). However, it was not until the famous backpropagation algorithm was developed that a way to efficiently update weights through the neural network was known. In 1986 Hinton, Rumelhart and Williams [12] successfully employed the backpropagation algorithm to train MLPs and the field of Deep Learning began to take shape.

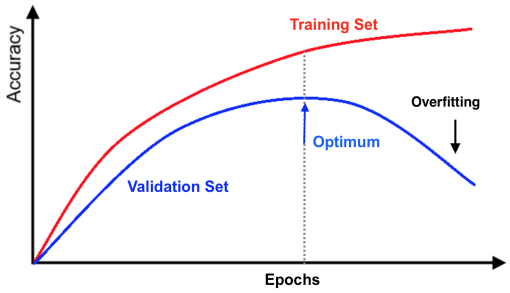
The predecessor to the modern Convolutional Neural Network (CNN) was introduced in 1980 by Fukushima under the name *Necognitron* [13]. The 1986 work by Hinton, Rumelhart and Williams built on the concept of the *Necognitron* requiring that the model share trainable weights. In the years to follow Necognitrons were continually improved [14] [15] and in 2003 their design was simplified [16] cementing the form of the modern CNN architecture.

1.2.3 Obstacles to Learning and Solutions

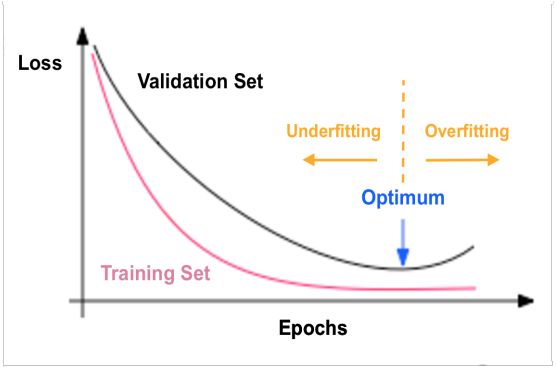
1.2.3.1 Overfitting, Regularization and Dropout

Deep Neural Networks can be thought of as a very effective universal function approximator; capable of capturing complex relationships between inputs and outputs. The existence of many parameters guarantees robustness but also increases the nets propensity for overfitting, especially on relatively small data-sets. In machine learning, overfitting describes the phenomenon of learning the noise in the data instead of the desired underlying dependencies between variables. In the context of NNs, overfitting means that consecutive hidden layers have redundant connections, which instead of learning meaningful patterns learn the noise in the data. This noise is unique to the training portion of the data-set and thus detracts from the solution upon validation and testing. Overfitting depends on the size of the model (i.e. model capacity) in relation to size of the data-set. If the model is too big for the data-set at hand this will exacerbate the tendency to learn noise. Thus to train a NN model, it is crucial to find the optimal model capacity. Figure 1.3 shows the typical accuracy and loss learning curves, illustrating the phenomenon of overfitting beyond the optimum point. It can be seen that during initial training stages (epochs) the discrepancy between the validation and training sets reduces, until it reaches a minimum, which is the optimum. Beyond this point the model begins to overfit and the difference between validation and training results begins to increase. Similar learning curves will be produced to visualize results in further chapters.

Various approaches to prevent NNs from overfitting exist; early-stopping, pruning, soft weight-sharing etc. However, the most popular are L2 regularization and dropout, which will be employed in this body of work and described in more detail in this section. The first group of approaches to prevent overfitting concentrates on altering model weights; rewarding weights of those neural connections with positive contributions to the overall solution, while penalizing other weights . This can be achieved through *regularization*, whereby a regularization term R(f(x)) is added to the objective function: $\min_f \sum_{i=1}^{n} L(f(\hat{x}_i), \hat{y}_i) + \lambda R(f)$, where $L(f(\hat{x}_i), \hat{y}_i)$ is the loss function describing the cost of predicting $f(x)$ when the true label is y for each



(a) Accuracy Learning Curve



(b) Loss Learning Curve

Figure 1.3: Typical NN Learning Curves and Overfitting

data point *i*. The regularization term can be chosen as the *L1-norm* or the Euclidean-norm, commonly referred to as *L1* and *L2* regularization respectively. Another approach to prevent overfitting does not attempt to interfere with model weights but instead it randomly *drops* certain units from the NN during training. When a unit is dropped it is temporally removed from the network along with all its connections. This technique was first introduced in 2014 and is known as Dropout [17]. It effectively prevents neurons from excessive co-adaption and often achieves superior results to other regularization approaches. Applying dropout to an NN is equivalent to sampling a smaller, *thinned* network at each training instance. Thus, an NN with n units can give rise to 2^n different thinned networks, which all share weights so that the total number of parameters remains unchanged at or below $O(n^n)$. At each training step a new thinned NN is sampled and this can be thought of as the bagging of all possible NN configurations. At test time dropout is not applied and a *full* NN with all its original units is used to evaluate the model. Results converge because the weights of the full network are scaled-down version of all the weights of the thinned networks learned during training. The use of dropout and approximate weight averaging during training is found to significantly reduce generalization error [17].

1.2.3.2 Exploding and Vanishing Gradients

The problem of vanishing and exploding gradients has first been pointed out in 1994 by Bengio et al. [18] and until today it presents one of the major obstacles

to training neural networks. Although, different types of networks suffer from this problem to varying degrees. The vanishing gradient problem arises when employing gradient-based learning methods to learn the optimal model weights. Upon every training iteration, each weight receives an update proportional to the gradient of the loss function with respect to the current weigh, computed using the chain rule. The numerical value of this derivative depends on the activation function, for instance the traditional $tanh$ function ranges between -1 and 1. Thus, for a network with n layers, backpropagation has the effect of multiplying n small numbers, in the range $(-1,1)$, to compute the gradient. For deep networks, the product of this gradients can quickly reach values very close to zero and the activation function is said to be *saturated*. In effect, the error signal carried by the gradients is lost and the model cannot learn. A suitable choice of the activation function can alleviate the problem of vanishing gradients, the ReLU unit is one such choice although a family of modern activation functions with improved differentiation properties have now been developed and will be introduced in the sections to follow.

Aside from using a ReLU activation function or similar, other architecture-specific solutions to prevent vanishing gradients have been developed. For RNNs special memory cells (LSTMs, GRUs) reduce the problem. Whereas, CNNs are inherently less susceptible to vanishing gradients, which will be explained in more detail in the sections to follow. Additionally, CNNs can now be implemented with recurrent connections, which further prevent vanishing gradients (detail in Section 2.1.3).

The problem of exploding gradients is very similar to that of vanishing gardens, with the difference that upon backpropagation using the chain rule of differentiation, products of gradients tend to a very large number instead of zero. As introduced by Pascanu et al. (2013) [19] gradient norm clipping is an effective way for dealing with exploding gradients.

Furthermore, a suitable weights initialization of the model weights at the start of trading can also hugely impact the overall solution and help control the vanishing or exploding gradient problem. It is not advisable to initialize model weights at

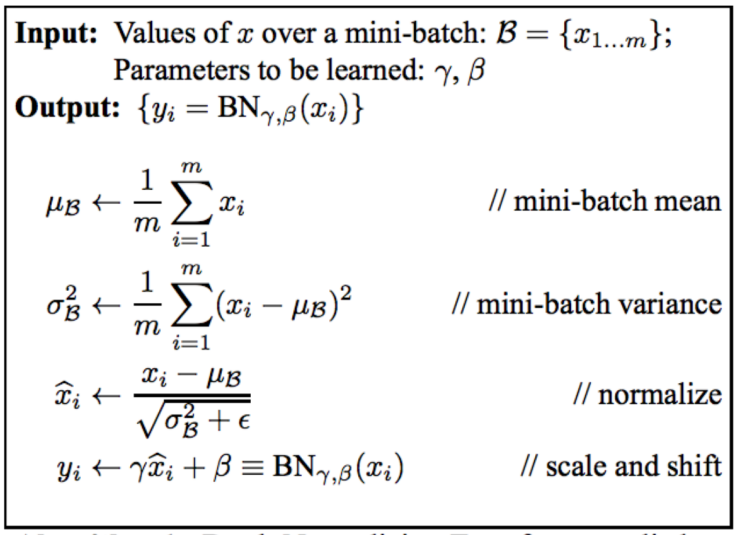


Figure 1.4: Batch Normalization Transform
Adapted from [20]

random, instead a statistical approach should be employed. One such approach is the *Xavier* weights initialization, whereby weights are sampled from the Gaussian distribution and a near constant variance of the inputs to each layer is preserved. This prevents the signal from exploding to a high value or vanishing to zero [9].

1.2.3.3 Internal Covariate Shift and Batch Normalization

In 2015 Ioffe and Szegedy uncovered another obstacle to NNs trading and called this the *Internal Covariate Shift* [20]. The authors define this as *the change in the distribution of network activations due to the change in network parameters during training*. This slows down training since small learning rates and careful parameter initialization are required. Furthermore, this phenomenon significantly inhibits learning in networks with saturating activation functions. To improve training, the authors suggest a method to reduce the internal covariate shift called *Batch Normalization*, which fixes the mean and variance of the input layers over min-batches of data during training. To make sure each layer has the same representation as prior to Batch Normalization (BN) the normalized inputs are scaled and shifted by learned parameters γ and β as per the algorithm in Figure 1.4. BN allows for the use of larger learning rates and less careful weights initialization, thus greatly reducing training time and achieving results competitive with those of Dropout. BN can be

thought of as a stand-alone regularization technique, however it can also be used alongside other regularization methods.

1.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a type of feedforward artificial neural network, biologically inspired by human visual neurons. The name stems form the mathematical convolution operation they employ, which simplifies the learning process by extracting local information in an affine and transformation-invariant fashion. CNNs are characterized by: local connections, wight sharing, pooling and multiple layers ,which will be explained in this section.

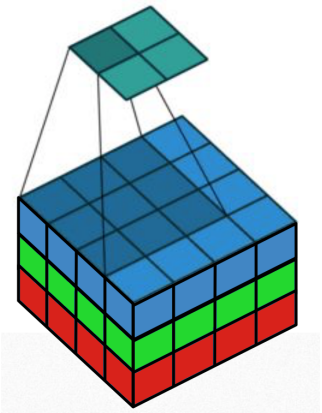


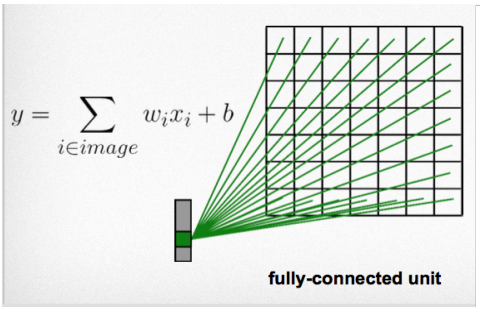
Figure 1.5: Basic 2D CNN input and Receptive Field

Adapted from [21]

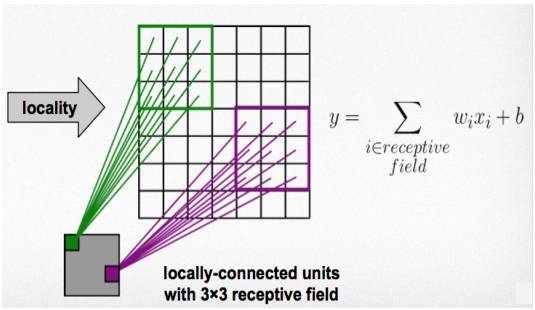
CNNs have been originally conceived to process data in the form of multiple arrays; for example, a colour image composed of three 2D pixel arrays, each array corresponding to a colour channel (RGB). This typical input is depicted in Figure 1.5, and the dimensions are widely referred to as: width x height x channels, also called features. However, input arrays of other size are also possible, such as 3D arrays for video or 1D arrays for sequential data. It should be noted that, since multiple input arrays are permissible, the actual input shape to the CNN has one more dimension than the CNNs dimension (i.e. a 2D-CNN takes 3D inputs and a 1D-CNN takes 2D inputs).

Figure 1.6 illustrates the properties of locality and weight sharing in the context of 2D images. These properties enable the transition from an ordinary fully connected layer (figure 1.6a) to a convolutional layer i.e one employing the convolution operation (figure 1.6c). The property of locality refers to the fact that objects tend to have local spatial support and depend on neighbouring pixels. Thus, a fully connected layer can be transformed to a locally-

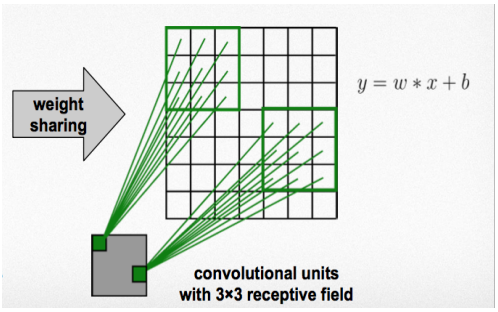
connected layer, where each neuron is connected to a group of pixels instead of one pixel. This grouping is depicted by the green and purple areas in Figure 1.6b. The entire input to the layer is called the receptive filed and sliding over this is the filter, also called kernel, which maps the input onto the feature map (output). The property of weight sharing, also called translation invariance, refers to the fact that objects have the same appearance regardless of their position on the image. Thus, similar objects will share weights, regardless of where in the pixel array they are encountered (Figure 1.6c). Stacking multiple convolution blocks together allows for the learning of increasingly complex data dependencies and has been the key to CNNs success. For example, applied to images, the initial layers would learn to distinguish edges whilst the deeper layers would specialize in linking these edges together into objects. Depicted on Figure 1.6, as the network learns the filter slides along the receptive field applying the convolution operation to each group of pixels. In addition to the filter number, parameters such as sliding, padding and dilation control how the filter is applied to the receptive field and what is the output size of the layer. Stride controls the step of the kernel and thus the convolutional operation. Stride is typically set to one, but if increased this will reduce the spatial resolution of the input and enable faster pro-



(a) Fully Connected Unit



(b) Locally Connected Units

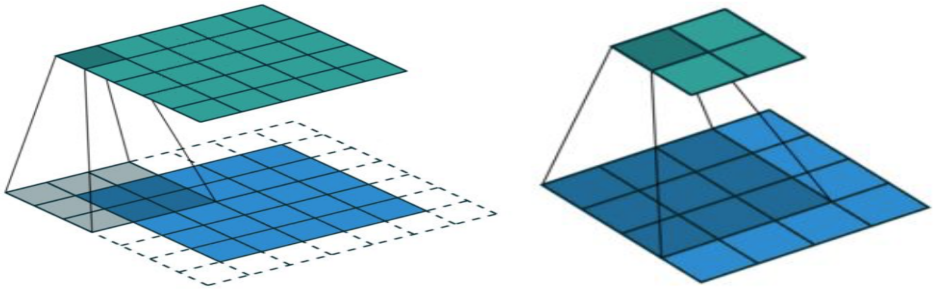


(c) Weight Sharing

Figure 1.6: Workings of a Convolution Layer

Adapted from [21]

cessing. Dilation achieves a similar result however instead of controlling where the sliding filter is applied it introduces a step between the filter elements themselves, as depicted on Figure 1.7. Dilation has the notable advantage of largely increasing the effective receptive filed without increasing the kernel size and the amount of network parameters. Padding is another control parameter for the convolutional layer, it dictates the shape of the output layer with respect to the original input layer. If padding is chosen to be *Valid*, then the output is reduced by a certain margin depending on the kernel size, as depicted in Figure 1.8b. If padding is selected as *Same* then the input and output dimensions are the same, this is achieved by padding the input with a suitable rim of zeros (Figure 1.8a).



(a) 'Same' padding **(b)** 'Valid' padding

Figure 1.8: Padding Variants
Adapted from [21]

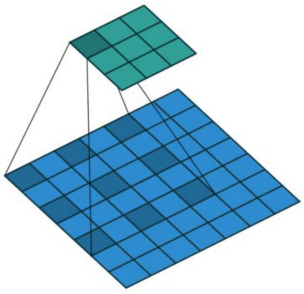


Figure 1.7: Dilation
Adapted from [21]

A pooling or downsampling layer is another building block of a typical CNN, it reduces output dimensionality and prevents overfitting, thus it is typically employed after the convolution layer. The downsampling can be performed in various ways, the most popular being taking the maximum or average values of the input to the layer, where the input size is dictated by a parameter. In contrast, a global maximum or average pooling layer will use the entire receptive filed as an input, producing a single output value.

Summarizing, a typical CNN architecture consist of a stack of convolutional layers followed by pooling layers. The final layer of this network is a fully con-

nected (MLP) to enable the output of desired regression or classification targets (Figure 1.9). For the purpose of this work it is also important to introduce a special type of CNN called a Fully Convolutional Network (FCN). In contrast to the conventional CNNs, which produce non-spatial outputs, the FCN can output a spatial map. This is achieved through the elimination of the final MLP layer of the network in favour of other building blocks. The reasoning behind this is to preserve information stored in spatial data coordinates, which would be thrown away by the MLP [22].

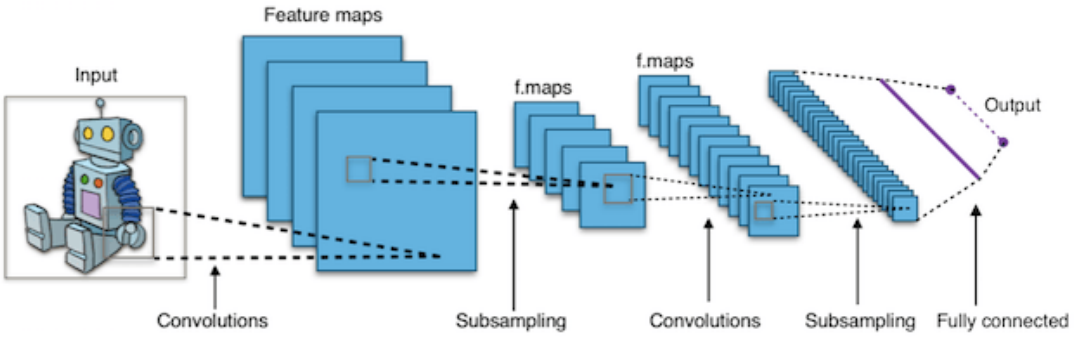


Figure 1.9: Schematic of a Typical CNN Architecture
Image adapted from [23]

1.4 Time-series Classification: Background and Terminology

This section introduces time-series classification and highlights some of the key issues of working with times-series data.

CLASSIFICATION

Supervised learning tasks can be divided into classification and regression problems. The aim of classification is to ascribe a feature vector to one of a discrete number of classes. This classes are also called labels and are in a categorical form. In binary-classification there are only two classes, whereas in multi-class-classification there are more than two classes. Regression on the other hand, focuses on predicting continuous valued output given the feature vector. Deep learning models can be applied to both types of problems albeit certain aspects of the methodology and

evaluation metrics differ [9].

TIME-SERIES

A univariate time series is any sequence of data points listed in time order, typically measured at uniform time intervals. A multivariate time series is a collection of more than one univariate time series with the same time-step and of equal length (Zheng et al 2014)[24]. In the context of time-series classification, time is the independent variable and other features recorded in time are dependent variables. In this work both univariate and multivariate settings are explored, where the features are price (univariate) or price an volume (multivariate).

A stationary time-series is generated from a stationary process, which in mathematics is defined as one whose joint probability distribution does not change when shifted in time. Statistical parameters such as mean and variance, also do not change over time. If a time-series is not generated by a stationary process it is called non-stationary and generally financial time-series exhibit global non-stationarity [25]. In deep learning non-stationarity is linked to the issue of *covariate shift*; where the training and test sets follow different distributions but the relationships learned by the ANN do not change between training and test phases. In other words, the distributions of inputs changes but the conditional distribution of outputs does not. Covariate shift can impede training of NNs, thus for time-series data it is useful to model the conditional probability of the output given the input vector as a chained conditional probability [26]. Additionally, it should be noted that, unlike typical classification problems, time-series classification uses ordered data (Bengall et al 2016)[4]. This means that discriminatory features dependent on data ordering can be learned. Thus, precaution should be taken when handling the data and shuffling should be avoided.

PROPERTIES OF FINANCIAL TIME-SERIES

Time-series originating form financial markets do not exhibit regular patterns, such as can be encountered in other forms of signals i.e. a heart beat or sound wave. Albeit financial data carries a lot of noise, it is not purely random and empirical intuitions about its properties have been developed by practitioners. Non-trivial,

statistical properties of price-series are referred to as Styled Empirical Facts and their existence in crude oil price series was shown, among others, by He et al. (2008) [27]. The existence of patterns in oil data makes it possible for deep learning networks to extract information and learn meaningful representations [2].

1.5 Problem Formulation

In this body of work, the application of selected CNN architectures to financial time-series classification was explored. The data used was minute-by-minute crude oil prices extracted during periods of anomalous market events. The overall aim of this investigation is to classify each financial time-series anomaly as a Buy or Sell case, using binary classification. Anomalies have been defend by Cambridge Quantum Computing (CQC) and further detail about their nature is proprietary. Supported by literature, the hypothesis posed is that it is possible to achieve better than random results in this classification task employing a variant of state-of-art CNN architectures. Thus, the goal of this work was to test some of the most notable result encountered in literature and apply these to the specific crude oil data-set at hand.

1.6 Report structure

Chapter 2 is the literature review, presenting key developments in CNN architectures as well as their application to time-series classification. Chapter 3 sets out the experiment design and the two baseline models used to evaluate performance. Chapter 4 describes the journey taken to improve baseline model performance and summarizes the results. Chapter 5 gives the final conclusions. Appendix A contains evidence of baseline model validation. Appendix B supplements the results reported in chapter 4.

Chapter 2

Literature Review

This chapter reviews relevant literature on CNN's and their application to time-series classification tasks. Section 2.1 summaries the most notable developments of CNN architectures in chronological order. The information provided is a birds-eye view with emphasis placed on concepts employed in this body of work. For further detail refer to the original literature. Section 2.2 describes various approaches to time-series classification but focuses on the recent applications of ANNs to this task. A detailed literature review of the use of CNNs in time-series classification is also presented.

2.1 Notable CNN Architectures

2.1.1 Form InceptionNet to VGG-Net

Using CNNs for Machine Learning tasks was first put on the map in 2012 when ImageNet [28], also called AlexNet, won the ILSVRC (Large-Scale Visual Recognition Challenge). This is a difficult and widely acclaimed annual competition in the field of Computer Vision, where new models compete in a range of tasks such as classification, localization and detection. ImageNet employed ReLU activation functions, dropout and data augmentation to achieve outstanding results. This first breakthrough is still arguably the most influential as it has paved the way for a series of improved CNN architectures emerging in the years after. The focus has been predominantly on tasks involving 2D and 3D spatial inputs, such as images or video. However, the outstanding results in Computer Vision tasks prompted the success-

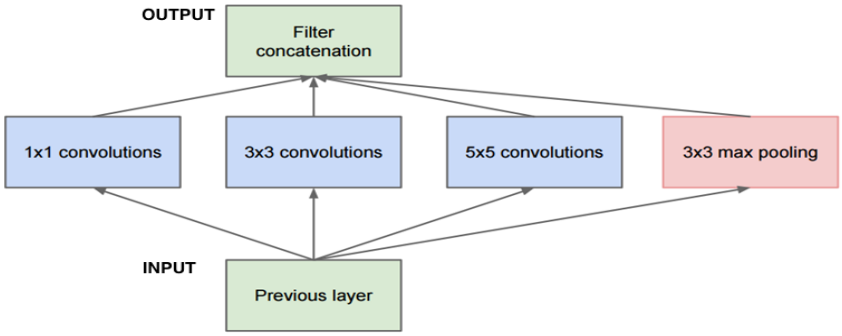
ful application of CNNs to other fields such as Natural Language Processing and Speech Recognition.

In 2013 the ILSVRC was won by another CNN called the ZF-Net [29], which was largely a tuned versions of the ImageNet. The authors delved deep into the inner workings of CNNs and developed ways to visualize these. In 2014 the VGG-Net [30] was introduced, it did not win the ILSVRC but it was groundbreaking in emphasizing the power of simplicity and depth. Unlike ImageNets 11x11 or ZF-Net's 7x7 filters, this architecture only employed 3x3 sized filters. However, these smaller filters were stacked together, effectively representing larger receptive fields but with significantly less weights. To illustrate; a stack of two 3x3 convolutional layers will have a receptive field of 5x5 and a stack of three 3x3 convolutions will have a receptive field of 7x7. This stacking 'trick' substantially reduces the number of model parameters and enables the construction of deeper networks. The authors also employed MaxPooling between stacks of convolutions. It is also worth noting that after each pooling operation the filter depth was doubled.

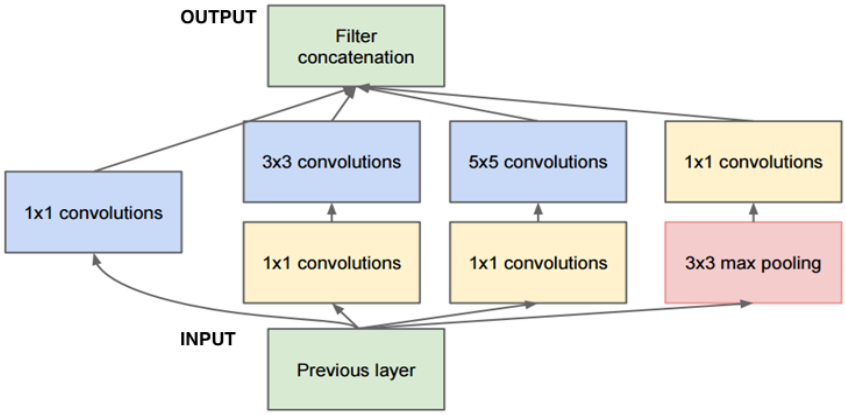
2.1.2 GoogLeNet

The 2014 ILSVR Challenge was won by GoogLeNet (LeNet), introducing the Inception module [3]. This network was groundbreaking in its departure from the notion that CNN layers needed to be stacked sequentially. LeNet has over a 100 layers arranged into 9 Inception modules, which allow for multiple calculations to happen in parallel. Figure 2.1 depicts a schematic respiration of the naive and full inception modules. Visualized with the naive schematic (figure 2.1a), the idea behind Inception module is to enable both pooling and convolution operations to occur in parallel. This is in contrast to traditional CNNs where a choice between either of those operations and the convolutional filter number has to be made. However, the naive implementation produces too many outputs; if applied directly it would result in a extremely large depth channel. Thus, the authors introduce further 1x1 convolution operations as depicted in Figure 2.1b. The added 1x1 convolutions provide a way of dimensionallity reduction along the channel dimension. Inspired by the Inception module, the application of 1x1 convolution with a pooling layer was

further explored in this project.



(a) Naive Inception Module



(b) Full Inception Module

Figure 2.1: Naive and Full Inception Module from GoogLeNet
Adapted from [3]

2.1.3 Microsoft ResNet

In 2015 the ILSVRC was taken by storm with the deepest CNN architecture yet, this was designed by Microsoft Research Asia [31] and became known as ResNet. Although, increased CNN depth was shown to improve results, in practice adding layers exacerbates the issue of vanishing gradients, hindering backpropagation and the discovery of optimal weights. In ResNet this effective limitation on CNN depth was overcome through the introduction of a novel architectural component called the residual connection, or skip connection. Figure 2.2 depicts a *mini* CNN module with the addition of a residual connection, as adapted from [31].

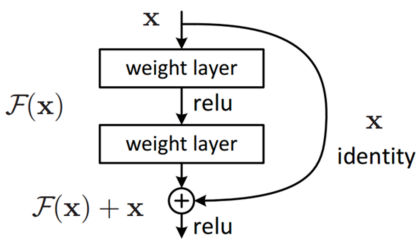


Figure 2.2: Residual Connection

Adapted from [31]

As can be seen, the input x is passed through a block (convolution + ReLU + convolution) to produce an output $F(x)$, this depicts the traditional workings of a CNN. Moving from x to $F(x)$ the model learns a completely new representation (i.e mapping) that does not keep any original information about x. The novelty comes in introducing a residual connection, which bypasses the networks block and adds the raw input x to the new representation $F(x)$. This sum: $H(x) = F(x) + x$ can be though of as learning a residual mapping, which also carries information about the original input x. Employing skip connections enables the learning of the delta of a slight change in x, which the authors believe to be easier to optimize than the unreferenced mapping ($F(x)$). Furthermore, addition operations introduced into the network (through the skip connections) have the advantage of distributing gradients when these are back-propagated through the network, substantially reducing the issue of vanishing gradients.

2.1.4 WaveNet

In 2016 another groundbreaking CNN architecture was developed under Google DeepMind, this time the application was not to machine vision tasks but to speech generation. This architecture, known as WaveNet [32], not only achieved state-of-art results in generating speech from text but also other sound-waves, such as music. It should be noted that unlike images, sound waves are a form of time-series and have a sequential nature. Thus, the triumph of WaveNet has opened the window of possibility for CNNs to be employed to other problems with sequential data, such as financial time-series classification.

The key component behind the success of the WaveNet model is the causal dilated convolution. *Causal* refers to the fact that the convolution operation can only be applied to that portion of the sequence of data, which has already happened prior to the prediction being made. Thus, the model cannot *look* into the future which would violate the data ordering. The use of dilation allows the receptive field

of the network to grow exponentially with depth and span thousands of time-steps without significantly increasing computational expense. The causal convolution with dilutions increasing by a factor of two in each layer, is schematically shown in Figure 2.3.

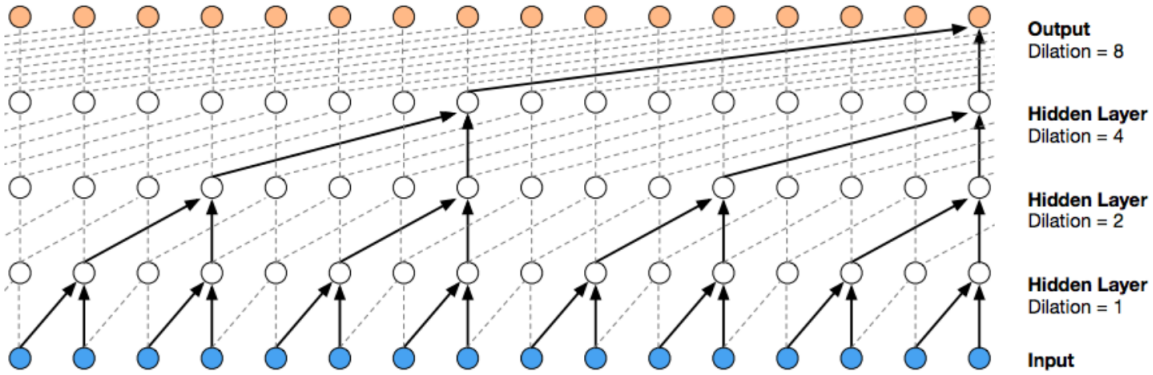


Figure 2.3: Schematic Net with Causal Dilated Convolution
Adapted from[32]

2.2 Time-series Classification

2.2.1 Classical Approaches

Traditionally, time-series classification approaches have been evaluated against a reference database known as the UCR data-sets. These database contains 47 time-series data-sets with varying origins and properties. Bengall et al. (2016) [4] have tested and summarized the performance of various time-series classification approaches using the UCR data-sets. Three types of methods can be distinguished for time series classification: distance-based methods, feature-based methods and Ensemble based approaches. Distance-based methods, such as work Dynamic Time Warping (DTW) or Euclidean Distance (ED), work on raw data to calculate a similarity measure and perform classification. For over a decade, state-of-the-art performance on UCR time-series classification has been achieved by combining k-nearest-neighbour classification (k-NN) with Dynamic Time Warping (DTW). This approach is is still used as a benchmark in recent works for instance that by Wang et al. 2016. Feature-based methods rely on extracting, from each time series, a set

of feature vectors that describe it locally and performing classification based on this [33]. Ensembling techniques that outperform kNN-DTW on reference data-sets, have been proposed in recent years. The most notable of these is the Collective of Transform-Based Ensembles (COTE)[1]. However, all the above methods require heavy data pre-processing, are computationally expensive and cannot be feasibly extended to large-scale data analysis [24]. This presents motivation to find more efficient algorithms to match or beat the performance of those outlined above.

2.2.2 Deep Learning Approaches

2.2.2.1 Recurrent Neural Network (RNN) Approach

Thus far, the preferred NN architecture for working with sequential data is the Recurrent Neural Network (RNN). Illustrated in figure 2.4, this net has a chain-like structure enabling the persistence of information from one input (x_t) instance to the next, which is in stark contrast with feedforward NNs.

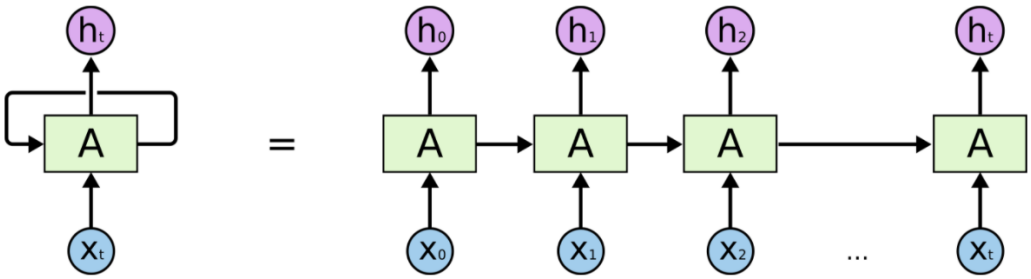


Figure 2.4: Schematic Recurrent Network (RNN)
Adapted from[34]

Vanilla RNNs are very susceptible to the vanishing and exploding gradients problem, which is overcome by the addition of Long-Short Term Memory (LSTM) or Gated Recurrent Units (GRU) to the network. This units can decide which steps of backpropagation through time are important and only use those to update model weights. A schematic LSTM unit is shown on figure 2.5, where each black line carries a vector from the output of one cell to the next.

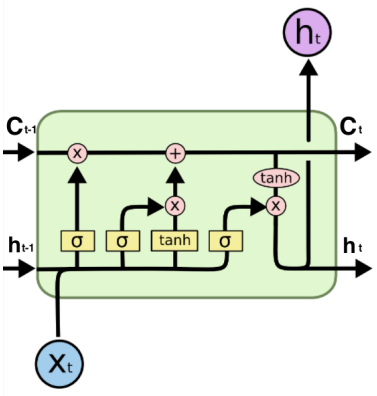


Figure 2.5: Schematic LSTM Unit Adapted from[34]

Circles represent point-wise operations and squares represent a NN layer with its activation function. C is the cell state passed from one time-step to another carrying information, h is the hidden state and x is the data input. C_t is generated at the current time-step by altering the previous cell state C_{t-1} passed form the depending on the observed input X_t and the hidden state h_{t-1}. There are three gates within the LSTM cell, each altering the form of the output C_t and h_t enabling the preservation or removal of information. This cyclical nature of RNNs, akin to having *memory*, renders them the first-choice for time-series classification [34]. However, RNNs applied to real-life financial data remain difficult to train and regularize, especially with increasing depth and model complexity. The use of RNNs for time-series classification has been relatively widely studied and will not be the focal point of this body of work. Instead, the use of a Convolutional Neural Network (CNN) will be examined.

2.2.2.2 Convolutional Neural Network (CNN) Approach

Despite their huge success in other fields, the use of CNN for time series classification has not been widely explored, particularly in the context of relatively noisy and unstructured signals such as the ones observed in financial market data. A trivial approach is to use 2D images of financial time-series as the input data for a CNN, this has been attempted by Siripurapu (2015) [35] with underwhelming results. This is unsurprising given the nature of the data and its graphical representation, which when analyzed on a pixel level carries little information. Intuitively, it seems that a 1D convolution on time-series data is a more direct and simpler way to employ CNNs for classification, and indeed this has been more widely explored in literature.

Zheng et al. (2014) were first to apply CNNs to time-series classification. The authors propose a Multi-Channel CNNs (MC-CNN) to study weakly and well aligned multivariate time-series arising from health-care research. In the approach

multivariate time series are split into univariate series and feature learning is done separately on each of these. This is done by feeding each series through a channel with two blocks of convolution with a sigmoid activation, followed by average pooling. The output of all the channels is then flattened, concatenated and fed into the final fully-connected layer, also known as the Multiple Layer Perceptron (MLP), to perform classification. The authors succeed in outperforming their chosen benchmark of 1-NN DWT, thus paving the way for further research onto the filed [24]. However, they only evaluate their approach on two proprietary data-sets and there is no benchmark for evaluation rendering the results hard to reproduce or compare.

Meanwhile, Mittleman (2015) [36] had implemented a undecimated fully convolutional neural network (UFCNN) to an univariate financial time-series specifically. In his model, the upsampling and pooling operations typically used in FCNs are replaced by upsampling of the filters with a factor that depends on the resolution level. He argues this approach, combined with maintaining the input and output layers of the CNN at the same rate, is critical for many applications involving time-series data.

Cui et al. (2016) implemented a multi-scale CNN approach (MCNN) for univariate time series classification. They heavily employ data preprocessing techniques: down sampling, skip sampling and window sliding to prepare for a multi-scale setting[37]. Although evaluation against the UCR data has been provided, the heavy amount of preprocessing and the use of univariate time series make it less applicable to this body of work.

Wang et al.(2016) [1] carried out a review of the time-series classification work applied to UCR data-sets. The authors also evaluated the performance of three different architectures on the UCR data-sets; Multi Layer Perceptron, (MLP), Fully Connected Convolutional Network (FCN) and a Residual Network (ResNet). The MLP was used as a baseline and proven to yield results very similar to that of the widely acclaimed golden baseline 1-NN and DTW. The FCN and ResNet architectures are adapted and simplified versions of the original architectures baring this names and described in the previous section. The authors also compared their mod-

els against Cui et al. (2016)[37] and other ensemble models, generally demonstrating the superiority of FCN and ResNet in terms of classification performance and ability to work with raw-data inputs. The authors argue that replacing the MLPs fully connected layers by a fully convolutional architecture, with global pooling and softmax, achieves the same classification results but is significantly faster and computationally cheaper than the nave approach. The minimal data preprocessing requirement and good results yield this architectures very promising for this body of work. Refer to chapter 3 for further detail of the simplified FCN an ResNet architectures.

Le Guennec et al. (2016)[33] implemented a simpler CNN with two convolutions layers, each followed by a sub-sampling step performed through max pooling. The focus of their work was on data augmentation techniques in cases where there is insufficient data to train a CNN. For evaluation the authors used univariate-time-series from seven 'small' UCR benchmark data-sets. In way of data augmentation, the authors experiment with window slicing (WS) and window warping (WW). WS relies on extracting slices from time series and performing classification at the slice level, where the slice size is a parameter. WW wraps a randomly selected slice of a time series by speeding it up or down. Techniques employed by Le Guennec et al. (2016) only slightly outperform those by Wang et al. (2016)[1] in the case of two data-sets. Since data augmentation is not at the core of this body of work, those techniques are note explored further. However, finding effective techniques to perform time-series data augmentation and produce data-sets of comparable size to those used for image classification, is very interesting topic in its own right.

Most recently, Borovykh et al. (2017) developed an Augmented WaveNet model capable of learning from conditional time series. They employed stacks of layered convolutions with ReLU activations and dilation growing exponentially within each stack. Additionally, they included batch normalisation and residual connections between layers. Their focus was on multivariate time-series modelled as conditioned on one another, this was applied to financial data such as commodity and stock index prices. Their network can effectively learn dependencies between

time-series without the need for long historical data and is shown to outperform a modified FCN architecture akin to the one by Wang et al. (2016). To compare between models the authors used data sets from a different source than the UCR benchmark. Five data-sets from the Time Series Data Library [38] were used and the evaluation was framed as a regression problem rather than a classification problem. Even though the focus is on conditional times series, this paper is very relevant as it draws on the work of Wang et al. (2016) and successfully applies a modified FCN architecture to financial data.

Chapter 3

Investigation Design and Baseline Models

This chapter outlines the experimental set-up for this investigation as well as the baseline models adopted for comparison and results improvement, undertaken in Chapter 4. Section 3.1 describes the data-sets used in the experiments, placing particular focus on the to proprietary financial data made available by *Cambridge Quantum Computing* (CQC) and its preprocessing. Section 3.2 outlines the two model architectures used as the starting point of this investigation, adopted after Wang et al.(2016). Section 3.3 outlines some crucial hyperparameter choices and the general procedure for hyperparameter tuning. Section 3.4 describes the set-up for model evaluation and visualization. Section 3.5 summarizes the best results obtained by applying the baseline models to the financial data set at hand.

3.1 Data-Sets Used

3.1.1 UCR Data

The UCR data-set repository has become a popular benchmark for evaluating the performance of new algorithms in time series classification tasks. The UCR Adiac data-set was used to reproduce results obtained by Wang et al. (2016) as wells to evaluate other proposed net architectures. The Adiac data concerns the classification of diatoms based on their images, the time series in generated as a set of distances to a reference point on the image. The data-set consists of 390 test points and 391

training points, each of length 176. This is a multi-class problem with 37 classes. It should be noted that the data exhibits clear sinusoidal patterns, rendering it easier to infer a signal as compared with noisy financial data.

3.1.2 Financial Data

The financial data-set used for the purpose of this work is one of Brent Crude Oil prices and volume generated by a proprietary anomaly detector. The data is multivariate, and contains two features: close price and volume traded. The data is discrete: recorded every minute, for a period of 301 minutes. To enable classification, each anomaly is accompanied by a binary label. The data-set contains samples from February 2010 onward. Data recorded in 2017 was reserved as test data. 85% of the remaining data was used for training and 15% was set aside for validating the model. The validation portion of the data was taken as the latest 15% of the remaining data i.e. data from the end of 2016 going back in time.

Two variants of the data-set were experimented with, the small data-set consist of 19873 samples, with 1371 reserved for testing and 18502 used for training and validation. The augmented data-set was constructed on the training portion of the data, by sampling at different time intervals as compared with the original small data-set. In effect, creating a 'big' data-set with approximately 13 times more samples i.e. 238524 samples. However, limited improvement of results was found when switching from the *small* to the *big* data-set.

3.1.2.1 Data Preprocessing

It is known that NNs training converges faster if its inputs are linearly transformed to have zero means and unit variances [39]. This form of normalization, commonly referred to as the *Z-Norm*, is given by $z = \frac{x-\mu}{\sigma}$, where: μ is the mean and σ is the standard deviation of the population [40]. Under this normalization regime data points can assume positive or negative values, which may be detrimental to training with a ReLU activation function, this is explored in more depth in chapter 4. Other methods of normalization are also possible, one popular choice for deep learning contexts is the use the *Min-Max* normalization, also called feature scal-

ing.This normalization technique does not return data points with negative values. In its simplest form the normalized data is in the range $(0,1)$, given by equation: $X_1' = \frac{X_i - X_{\min}}{X_{\max} - X_{\min}}$, where for each data point (X_i) the minimum value is subtracted from this point and divided over the difference between the maximum and minimum values[40].

To aid training and remove scale effects, each anomaly instance was normalized with respect to itself using Z-Norm. This method preserves range and carries information about the mean and standard deviation of the series [40]. However, in the later stages of results fine-tuning, Min-Max normalization was also trailed to examine its impact on prediction quality.

The data pipeline was also constructed so that the look-back window could be controlled, this determines the length of the time-series to be used for learning. The maximum look-back period is the duration of the anomaly itself (301min) but shorter time frames can also be analyzed. When shorter time windows were considered then the latest data points were included i.e. counting from the anomaly end.

3.2 Baseline Models

Two baseline models were adopted after Wang et all.(2016) implementation of a Fully Connected Network (FCN) and a simplified Residual Network (SResNet). Generally, the FCN was show to outperform other approaches including the SResNet when applied to the UCR data-sets. However, the SResNet architecture is deeper and more robust as it employs residual connections[31]. Thus, there is ground to believe it could beat the FCN on classification tasks with nosier, complex data.

All models, including those in Chapter 4, were implemented using Keras - a high-level deep learning API, written in Python. Model prototypes were trailed on an ordinary PC and tested on GPUs made available by CQC.

3.2.1 Fully Connected Network (FCN) Architecture

This baseline model replicates Wang's et all.(2016) implementation of a Fully Connected Network (FCN), which was shown to outperform other approaches on classification of the UCR data-sets. For the classification problem at hand, the FCN was implemented as a feature extractor with a final softmax layer. Figure 3.1 illustrates this architecture with 3 convolutional layers, batch normalization (BN) and ReLU activations. The basic building block is summarized by equations 3.1.

$$\begin{aligned} y &= W \otimes x + b, \text{where} \otimes \text{is the convolution operation,} \\ s &= BN(y), \\ h &= ReLU(s) \end{aligned} \tag{3.1}$$

In each convolution layer, the filter numbers are: 128, 256 and 128 respectively, kernel sizes are: 8,5 and 3 respectively, padding is set to *Same*, there is no striding nor dilation. Unlike most CNN architectures, there was no down-sampling between stacks. Instead a GlobalAveragePooling layer was introduced before the final softmax activation. As argued by the authors, this effectively replaces the fully connected layer, significantly reducing the number of weights in the model.

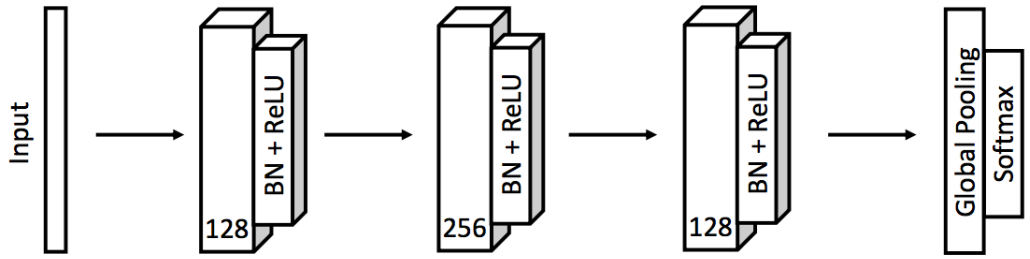


Figure 3.1: Baseline: Fully Connected Network by Wang et al.(2016)
Adapted from [1]

3.2.2 Simplified Residual Network (SResNet) Architecture

The *ResNet* architecture created by Wang et al.(2016), inspired by the original Microsoft ResNet (Section 2.1.3), is referred to as the Simplified ResNet (SResNet) in this body of work. Like the original ResNet, SResNet also employs residual con-

nections between blocks of convolution units, but it is significantly shallower. The architecture is depicted in Figure 3.2; each convolution block, or stack, consists of 3 convolutional layers, with ReLU activations followed by batch normalization (BN). Within each stack the filter number f is fixed: 64 in the first stack and 128 in the remaining stacks: $f_s = (64, 128, 128)$, where s indexes the stack. Within each stack the kernel sizes vary in the same way: $k_i = (8, 5, 3)$, where i indexes the convolutional layer number. The architecture is summarized by equations 3.2, where *Stack* refers to a convolutional block with f filters, as described by equation 3.1. Similarly to the baseline FCN, the 3 stacks of convolution layers are followed by a Global Average Pooling layer and a final label is produced with a softmax layer. Padding is set to *Same* and there is no dilation nor striding. The parallels between the FCN and SResNet architectures should be noted; each block of the SResNet effectively repeats the hidden part of the FCN, with skip connections added in between.

$$\begin{aligned} h_1 &= Stack_{f_1}(x) \\ h_2 &= Stack_{f_2}(x) \\ h_3 &= Stack_{f_3}(x) \\ y &= h_3 + x \\ \hat{h} &= ReLu(y) \end{aligned} \tag{3.2}$$

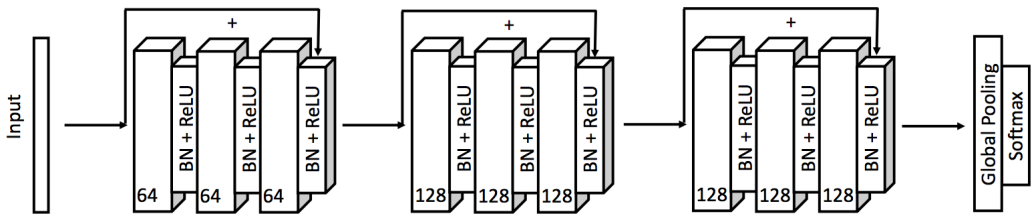


Figure 3.2: Baseline: Simplified ResNet by Wang et al.(2016)
Adapted from [1]

Even though, Wang's et al.(2016) FCN beats the SResNet on classifying UCR data-sets, this is possibly due to the insufficient complexity and small size of this data-sets. Thus, the authors believe that there is promise in applying deeper and more complex CNN architectures for time-series classification.

3.3 Hyperparameters

Both baseline models employed Adam optimizer with default parameters. Additionally, the ReduceLROnPlateau Keras callback was used, enabling the reduction of the learning rate when learning stalls over a pre-specified number of epochs. In their vanilla implementation, the models were geared at UCR data classification thus batch sizes were kept low and the maximum look-back period was used. For the implementation on financial data the batch size, learning rate and look-back period were adjusted.

Hyperparameters tuning was undertaken in two ways: using grid search as well as a specialized package called *Hyperas*. Hyperas is an easy to use Keras wrapper to python's *Hyperopt* library, which performs Bayesian Optimization in the search space. This is substantially more efficient than a full grid search. The following hyperparameters were tuned: learning rate, look-back period, batch size, filter number (in each convolutional block), kernel size, striding, padding, dilation-rate, l2-regularization rate, dropout rate.

3.4 Model Evaluation and Visualization

To maintain consistency, all models developed in this project shared the same time seed, ensuring results reproducibility through fixing random initializations across models. Additionally, Xavier weights initializer with default settings was used throughout.

All neural network models developed in this work were evaluated in the same way. Since, the problem is one of binary classification, the model loss was measured with categorical cross-entropy (equation 3.3). This measures the distance between two probability distributions; the binary distribution of one-hot encoded true labels (p(x)) and the distribution returned by the final softmax layer of the NN (q(x)) [9].

$$CE(p,q) = -\sum_{x} p(x)\log(q(x)) \tag{3.3}$$

The model accuracy, measuring the fraction of examples predicted correctly, was

calculated using Keras *categorical accuracy*. This metric takes the highest value to be the prediction and matches its against the true label.

Basic results, such as plots of accuracy and loss were visualized in-code using the python matplotlib package and displayed in the working console. However, the model logs were also saved to be compatible with TensorBoard, which is a sophisticated TensorFlow visualization platform. This enabled interactive and real-time display of not only the learning curves, but also the distributions over model weights and the NN compute graph. Figures aiding the presentation of results in the sections to follow were generated using both of these approaches.

3.5 Validation and Results

The baseline models described above were recreated and tested against selected UCR data-sets to verify their correct working. Detailed evidence of the models testing can be found in appendix A. It should be noted that recently Keras added the one dimensional convolution operation (*Conv1D*), however the original FCN implementation had to have been undertaken using the established two dimensional convolution layer (*Conv2D*), setting the redundant dimension to 1. The impact of this implementation difference was tested, and it was concluded that the new Conv1D operation achieves results very similar to those achieved by the *Conv2D* command. Additionally *Conv1D* is faster and requires less parameters, therefore it was chosen as the preferred building block for all models going forward.

Following successful validation, the baseline models were applied to financial crude oil data. At the onset, hyperparameters yielding the best results when applied to the new data-set needed to be determined. This was done by a mixture of using grid search, exploration and the hyperas package. Initially, focus was placed on determining the optimal look-back period (also called window size), batch size and learning rate. It was found that using the maximum look-back period gave the best results. This is intuitive since the NN was presented with the most complete information about the anomaly. Batch sizes ranging from 16 to 2000 were trailed and it was established that sizes in the range of 200-500 performed best. When it

comes to learning rates, a reduction from the Adam Optimizer default of 1e-4 aided training and the starting learning rate was set to 1e-5. The learning rate was additionally controlled by the *ReduceOnPlateau* Keras callback, which automatically reduced the learning rate when training stalled for a specified number of epochs. The callback was set to monitor validation accuracy and halve the learning rate if no learning occurred for 300 epochs; the minimum learning rate was capped at 5e-7. The filter numbers and kernel sizes were preserved as per the original architecture. Additionally, in line with Wang's et al.(2016) implementation, data was normalized using *Znorm*, no regularization other than batch normalization was employed and the convolutional layers had a ReLU activation.

Various data variants were tested: training on one-feature data i.e price only as well as two-feature data i.e price and volume. This was done on both the small and large data-sets. Results obtained using the small data-set and two features (price, volume) were the best and are presented in Table 3.1 below. Generally, results obtained by training on the small data set and using two feature data were much better, thus this set up was employed for all further experiments presented in this work.

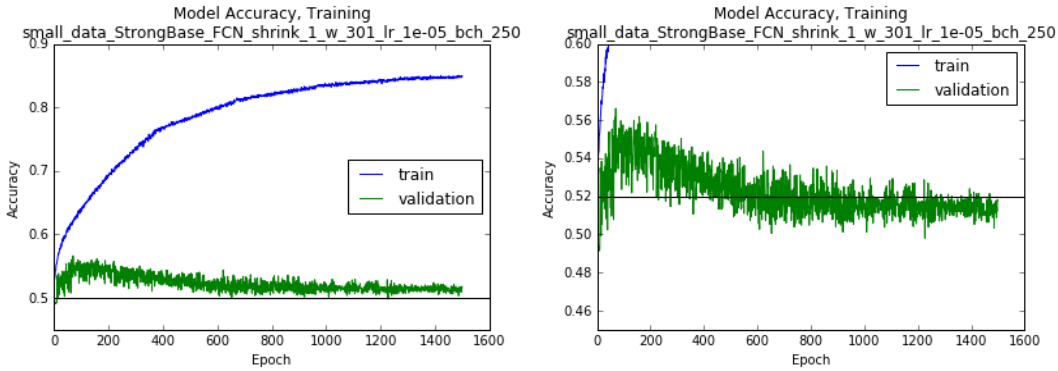
Table 3.1: Baseline Models Results

Best refers to Max Accuracy and corresponding Loss

	Baseline FCN		*Baseline SResNet*	
	Accuracy	Loss	Accuracy	Loss
Training Best	0.850	0.387	0.991	0.072
Validation Best	0.566	0.688	0.547	0.693
Testing Best	0.485	0.711	0.504	0.72

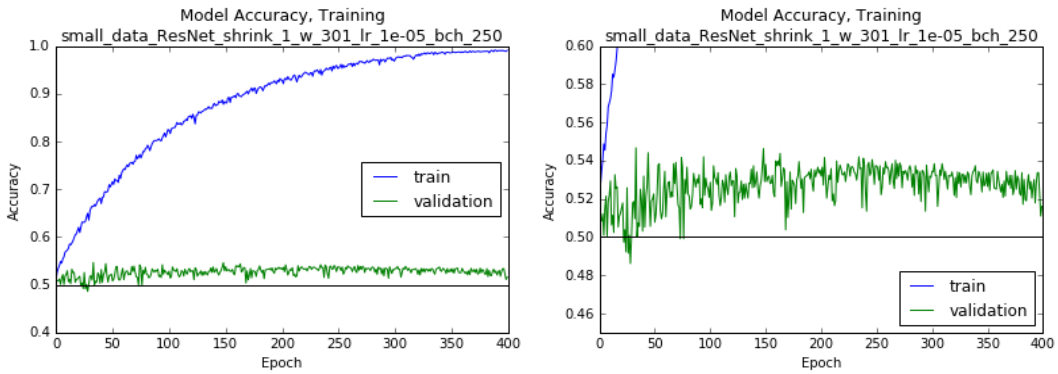
As can be seen form figures 3.3, 3.4 and 3.5, the Baseline Models yielded underwhelming, mostly close to random results. Despite reaching a relatively high best validation accuracy score of circa 56% for the FCN and 55% for SResNet, the test accuracy was below or near random. This is possibly due to the substantial noise and complexity of the financial data, as compared with UCR data-sets, where clear patterns are identifiable. The Baseline FCN model was trained for 1500 epochs, this was deemed too long as overfitting occurred, therefore the Baseline SResNet

was trained for 400 epochs only. However, since the SResNet model has roughly twice as many parameters as the FCN, even with 400 epochs severe overfitting was observed - test accuracy reaching close to unity while validation results remained poor. Even though SResNet performed marginally better on the test set the difference between FCN and SResNet out-of-sample results is too small to be conclusive. Unlike SResNet, the FCN model was learning promisingly well in the initial 400 epochs thus there is more ground to believe the model can be improved to yield stable above random results. focus on improving the FCN architecture going forward. Thus, in line with findings by Wang et al. (2016), the FCN architecture appears to outperform the SResNet and holds more promise for further development.



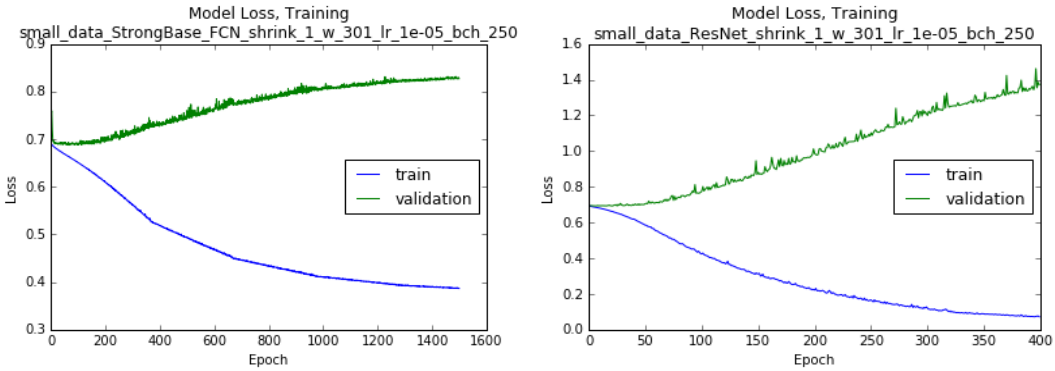
(a) Test vs. Validation Accuracy **(b)** Detailed Validation Accuracy

Figure 3.3: Baseline FCN Accuracy Plots
Learning rate: 1e-5, look-back period: 301 min, batch size: 250, parameters: 265,986



(a) Test vs. Validation Accuracy **(b)** Detailed Validation Accuracy

Figure 3.4: Baseline SResNet Accuracy Plots
Learning rate: 1e-5, look-back period: 301 min, batch size: 250, parameters: 524,848



(a) Baseline FCN Accuracy **(b)** Baseline SResNet Loss

Figure 3.5: Baseline FCN and SResNet Loss Plots
Learning rate: 1e-5, look-back period: 301 min, batch size: 250

Chapter 4

Improved Architectures

Following the initial poor results of applying Baseline Models from chapter 3 to financial data, focus was placed on developing various improvements to this architectures in an attempt to boost their performance.

Sections 4.1 and 4.2 present the journey taken to improve the original baseline FCN model and arrive at *My Better FCN* model. This was the main focus of this work due to the FCN's strong performance in other time-series classification tasks as well as results obtained in chapter 3. Section 4.3 gives further detail on some of he crucial ideas explored to reach the final FCN architecture.

Section 4.4 presents results obtained with a reduced SResNet architecture. Less emphasis was placed on the development of this network given that it performed worse than the FCN model in its baseline implementation.

Section 4.5 contains concluding remarks; a comparison between the improved FCN and SResNet is made. This section also provides a reflection on model evaluation metrics, justifying why validation accuracy and loss are the most indicative of model performance.

4.1 Exploration of Avenues for FCN Improvement

The scope of possible modifications to the baseline FCN is vast. As a starting point the batch size, look-back period and learning rate were tuned and fixed to enable the exploration of the impact of other architecture alterations. This included the modification of filter and kernel sizes, trailing the use of different dilation rates, padding

and pooling variants as well as addition of convolutional layers. The findings from this initial experiments are briefly summarized below and supplemented by results reported in Appendix B. **Filter and kernel size**

It was immediately noted that reducing the FCN's number of channels did not detract from results. For the initial experiments reduced filter numbers of: $f_i = (12, 25, 12)$ were used where i indexes the convolutional layer. In line with literature, taking inspiration from VGG-Net and later notable CNN architectures, kernel sizes were also decreased in various combinations. This reduced the amount of parameters which needed learning but did not improve results. Therefore, the baseline FCN kernel sizes were kept going forward.

Dilation, padding and pooling

Taking inspiration from the original WaveNet architecture [32] as well as its application to financial-time series [2] described in chapter 2, dilation was added to the modified FCN architecture. In literature, for every stack the dilation rate starts at 1 and is doubled in each successive convolutional layer. Various dilation configurations were tested, and it was determined, that dilation set to: $d_i = (1, 2, 2)$,where i indexes the convolutional layer, produced the best results. Alongside trailing different dilation rates, the impact of the padding setting was also examined. The baseline FCN adopted *Same* padding, however with financial time-series data *Valid* padding produced improved results. Thus, further modifications to the baseline FCN consisted of including dilation and changing padding to *Valid*. Supporting evidence can be found in appendix B. Finally, the difference between employing Global Max Pooling versus Global Average Pooling was explored, while keeping all other parameters constant. Contrary to results reproduced in Appendix A and literature [2], it was determined that *GlobalMaxPooling* outperformed *GlobalAveragePooling* and this was adopted in the model going forward. However, developments in further sections will mandate switching back to *GlobalAveragePooling*, in line with the general consensus.

Choice of Optimizer and Learning Rate

A thorough trail of possible optimizer choices was also undertaken, this included popular optimizer choices such as: Adagrad, RMSprop, Adam, SGD. It was determined that Adam Optimizer performed best, thus it was employed in all further models. The impact of including the ReduceOnPlateau callback, which automatically reduces the learning rate when validation accuracy ceases to improve was also examined in more depth. It was concluded that maintaining a fixed learning rate enhanced results. The callback was thus dropped going forward.

Additional Convolutional Layers

Although Borovykh et al. (2017) focused on conditional time-series forecasting using an adapted WaveNet architecture, their research paper commences with an exploration of the FCN proposed by Wang et al.(2016). The authors claim that improved results can be obtained using a convolution layer with kernel size set to 1 as the final layer (i.e. *conv1x1*), instead of the Global Average Pooling layer employed in the Baseline FCN model. This claim was extensively tested and validation of results was attempted. However, this solution did not yield satisfactory results when employed on the financial data-set. Never the less, the idea of including *conv1x1* was further explored, as this has been successful in GoogleLeNet's Inception module. In this module various layers are run in parallel and combined with a 1x1 convolution, which reduces the output to a 1 by 1 stack of depth equal to the number of filters. One of the branches of the Inception module links a pooling layer with a *conv1x1* layer. Various combination of convolutional layers followed by a *conv1x1* were attempted, but it was found that the combination of pooling with *conv1x1* yielded the best validation results.

4.2 Summary of the FCN Development Process

4.2.1 First Improved Architecture: *My Baseline FCN*

Following the initial exploration described in the section above and appendix B, the first modified FCN model employed: filter numbers: $f_i = (12, 25, 12)$, kernel sizes: $k_i = (8, 5, 3)$ dilation $d_i = (1, 2, 2)$ where i indexes the convolutional layer. Padding

was set to *Valid* and *GlobalMaxPool* was used in the final layer. A schematic of this architecture is given in figure 4.3. The optimal parameters were found to be: leaning rate: 3e-6, batch size between 250 and 400, look-back period: 301 minutes. It should also be noted that applying *L2* regularization had no impact on the solution. This model was trained for a 1000 epochs, figure 4.1 presents the learning curves and a summary of results. It can be seen that the model begun to overfit between epochs 400 and 500, thus warranting further reductions in training length going forward. When it comes to the test results, accuracy had been improved to above average, which is a promising result. However, this occurred at the expense of an increase in test loss, which needed reducing going forward.

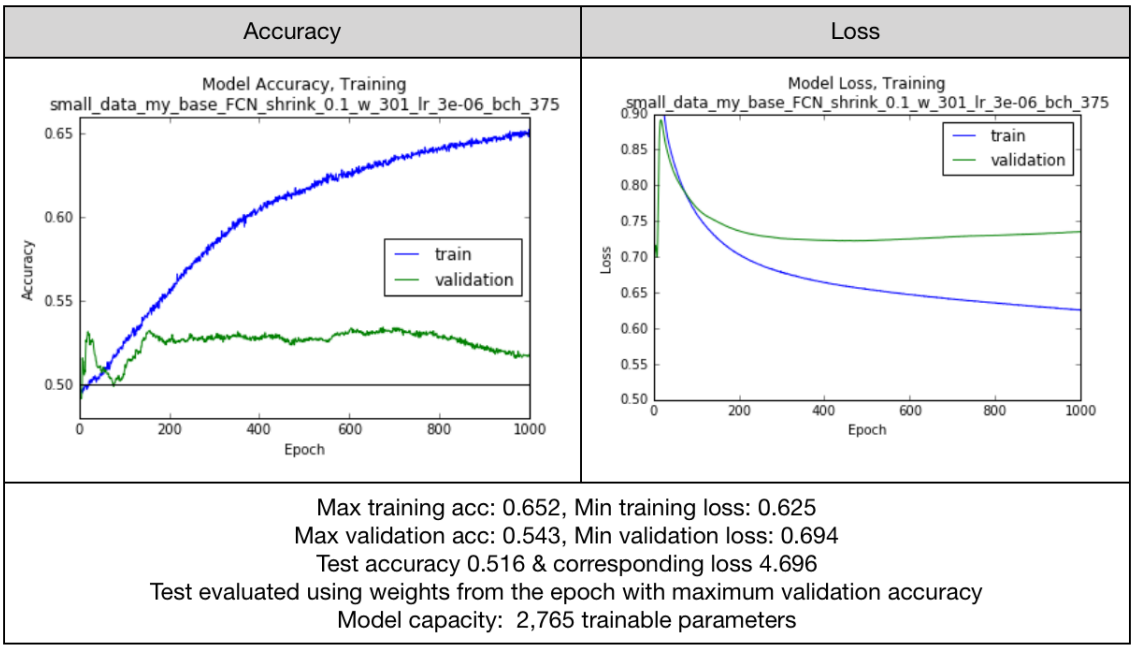


Figure 4.1: Results Summary for *My Baseline FCN*
Learning rate fixed at 3e-6, look-back period is 301 min, batch size 375

4.2.2 Second Improved Architecture: *My Better FCN*

Following on from *My Base FCN*, *My Better FCN* was developed through the addition of a 4th convolutional layer. This was a *conv1x1* with dilation kept at 1 and the number of channels same as in the previous (3rd) convolutional layer. This effectively created a bottleneck output vector of size: $1 \times 1 \times$ *num. filters*, which was followed by a pooling layer. In this architectural variant Global Average Pooling

substantially improved results and was thus put forward. It was also found that adding a moderate amount of depth to the network and doubling the filter size in the 3rd convolutional layer improved validation results. In general all these findings were in agreement with literature, reaffirming that the architecture was being shaped in the right direction. It should also be noted that, this architectural variant increased the model parameters roughly by a factor of ten: from 2,765 in *My Baseline FCN* to 27,938 in My Better FCN. The substantial increase in model capacity, without a matching increase in data-set size could potentially exacerbate overfitting and detract from learning. However validation results presented in figure 4.2 below show the contrary.

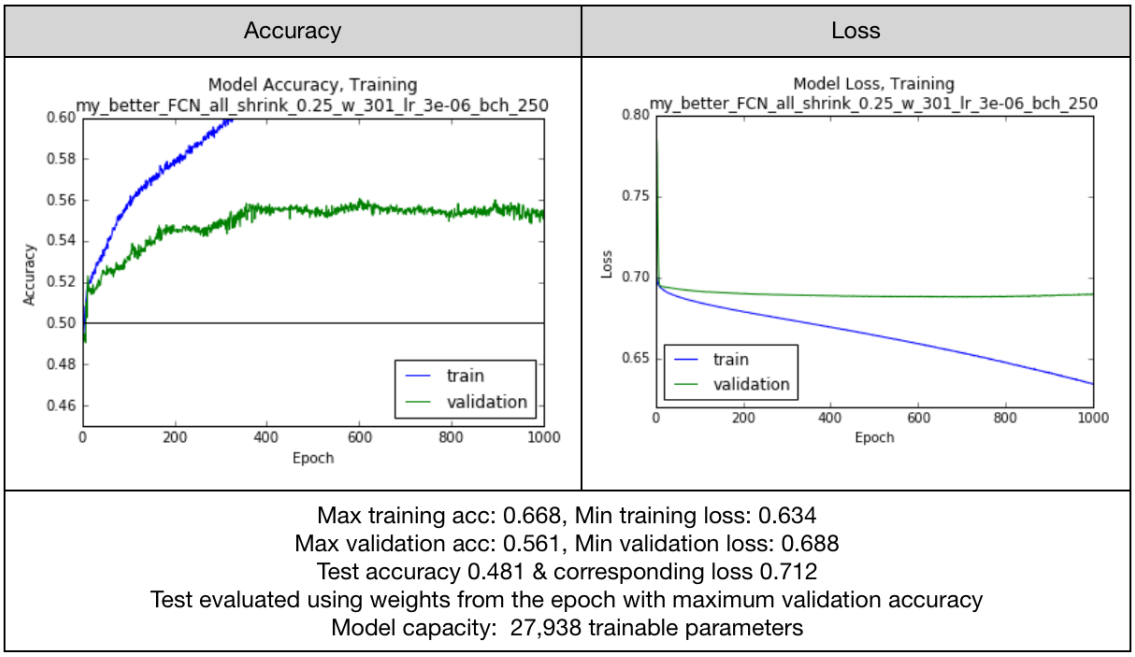


Figure 4.2: Results Summary for *My Better FCN* without Dropout
Learning rate fixed at 3e-6, look-back period is 301 min, batch size 250

From the training curves in figure 4.2 an improvement over *My Baseline FCN* and the baseline FCN model can be seen; validation accuracy reached a higher average value and did not decrease with the amount of epochs. However, the test accuracy remained close to random. In an attempt to further boost performance dropout was added to *My Better FCN* (detail in section 4.3.3). This, improved validation results and was therefore kept going forward. The change of activation

function was also attempted from ReLU to Leaky ReLU (detail in section 4.3.4).

Summarizing, the final FCN model put forward consists of 4 convolutional layers each with a ReLU or LeakyReLU activation, followed by Batch Normalization (BN) and Dropout. Parameters: filter numbers: $f_i = (32, 64, 64, 64)$, kernel sizes: $k_i = (8, 5, 3, 1)$ dilation $d_i = (1, 2, 2, 1)$ where i indexes the convolutional layer. The final layer is a Global Average Pooling layer followed by a softmax for classification. *My Better FCN* is depicted in figure 4.3.

4.2.3 Progression of FCN Architectures and Results

Figure 4.4 and table 4.1 give an overview of the results obtained training all the FCN models. Whereas figure 4.3 illustrates a progression of the FCN model architectures. From the training and validation learning curves it can be seen that the variants of *My Better FCN* achieve superior performance to the other FCN architectures. However, on the out-of-sample *My Baseline FCN* marginally outperforms the other architectures, this could be a chance outcome or it could be attributed to the significantly lower model capacity. Nevertheless, all testing results are near random and the differences between test scores are too small to be informative. What can be said with more certainty is that improvement in the validation scores is achieved by variants of *My Better FCN* model. Albeit the maximum validation accuracy is similar to the baseline FCN, from the learning curves it can be seen that *My Better FCN* models achieve stable learning and desired shapes of the validation loss curves.

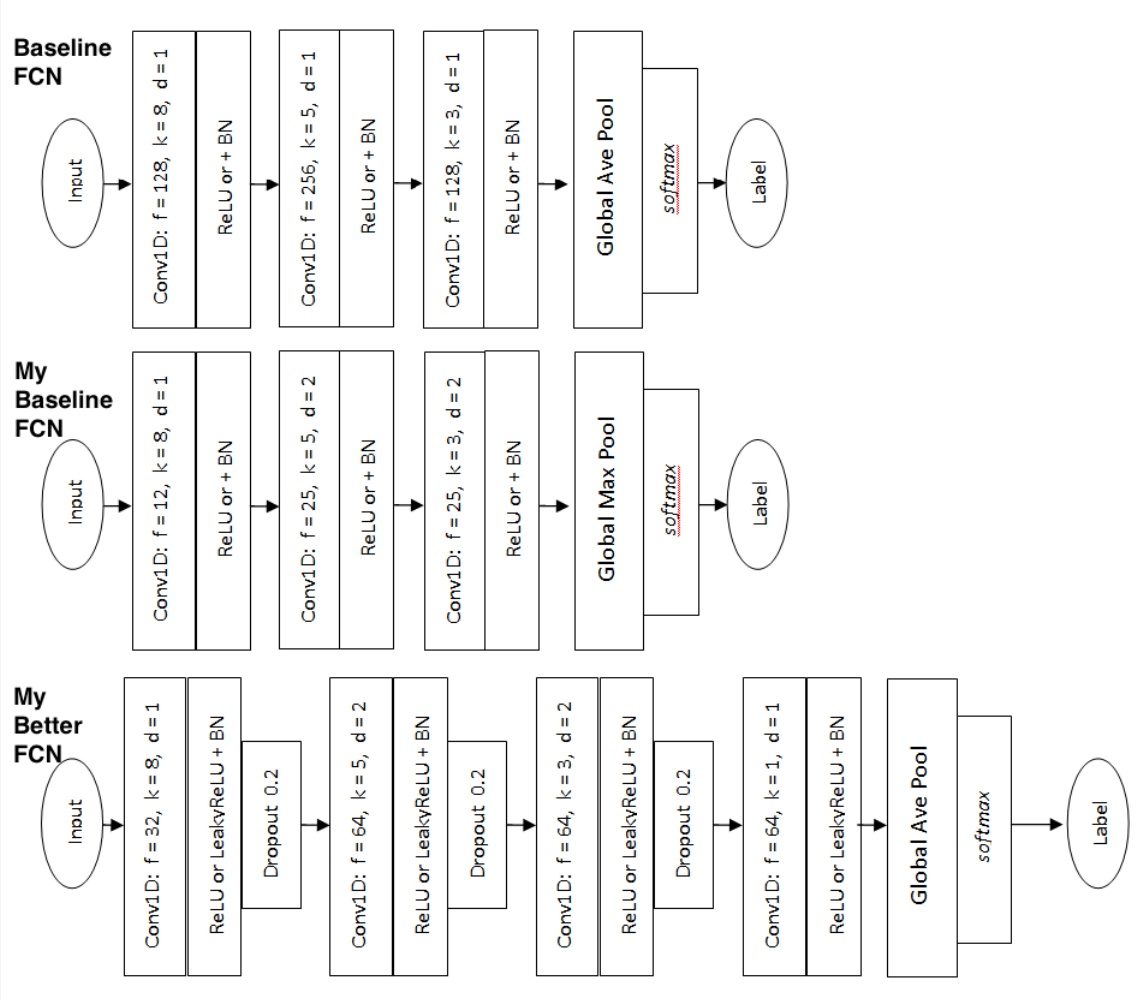


Figure 4.3: Schematic of All FCN Architectures
f is the filter number, k is the kernel size, d is the dilation rate,

Table 4.1: Comparison of FCN Models

Best refers to Max Accuracy and Min Loss
Both *My Better FCN* architectures include Dropout

	Baseline FCN		*My baseline FCN*	
	Accuracy	Loss	Accuracy	Loss
Training Best	0.850	0.387	0.652	0.625
Validation Best	0.566	0.688	0.534	0.694
Testing Best	0.485	0.711	0.516	4.696

	My Better FCN		*My Better FCN + LeakyReLU*	
	Accuracy	Loss	Accuracy	Loss
Training Best	0.594	0.669	0.568	0.681
Validation Best	0.577	0.683	0.560	0.687
Testing Best	0.471	0.731	0.490	0.715

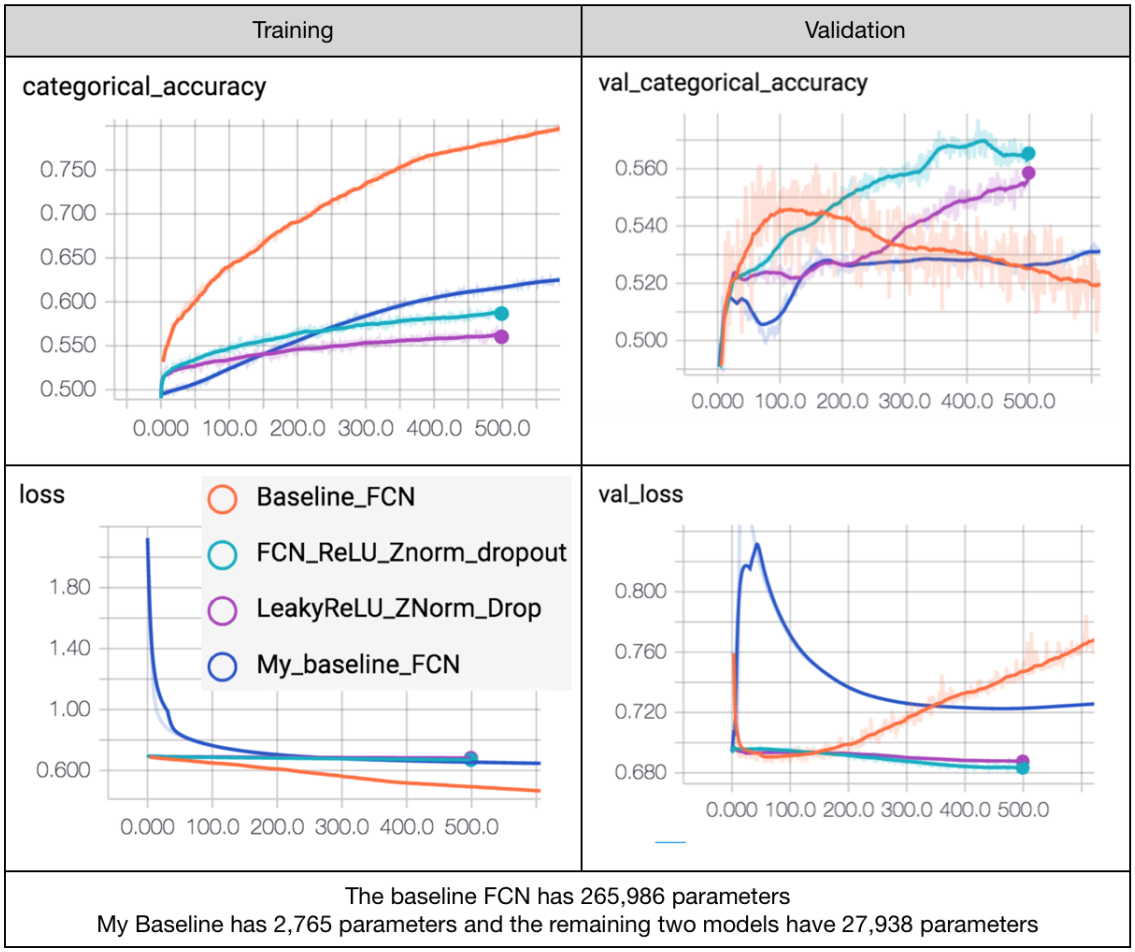


Figure 4.4: Comparison of All FCN Learning Curves vs. Epoch

4.3 FCN Discussion

4.3.1 Impact of Intermediate Pooling layers

Additionally to setting the final layer back to Global Average Pooling, a sanity check was performed to verify Wangs et al. (2016) claim that intermediate pooling layers are redundant in the FCN architecture. Typically, pooling is employed between stacks of convolutional layers, such has been the case in VGG-Net [30], as well as architectures developed specifically for time-series classification: the MCNN [37] and MC-CNN [24]. Therefore, another avenue for results improvement was to re-instigate these layers into the baseline FCN. This has been done and both max and average pooling layers were tested, with and without the inclusion of batch normalization layers, their presence did not improve results. Thus, as far as intermediate pooling layers are concerned this work cedes with Wang et al. (2016) in concluding that their removal form the FCN architecture is beneficial.

4.3.2 Impact of Changing the Data Normalization technique

Despite achieving widespread success in modern CNN architectures, the ReLU activation function has a potential drawback when employed together with *Znorm* data normalization. *Znorm* produces data points with zero mean and unit standard deviation, with negative values present. While ReLU sets gradients for all negative values to zero and permanently 'switches off'. Thus, it could be that in the initial stages of training a disproportionately large portion of the network is set to zero and made redundant before any meaningful learning can occur. Even though *Znorm* was used with ReLU in Wang et al. (2016) FCN implementation switching to *Min-Max* normalization (described in section 3.1.2.1) was attempted to examine the impact on the solution. Figure 4.5 summarizes the results obtained; it can be seen that *Min-Max* normalization does not improve the solution. Both training and validation results are worse than in *My Better FCN*. The learning curves exhibit more variability and the tests scores remain below random. This can be partially attributed to the lack of careful weights initialization, despite employing Xavier Initializer. In light of this findings, *Znorm* was be kept going forward and another

avenue for improving the activation layer performance was explored, as described in the section 4.3.4 below.

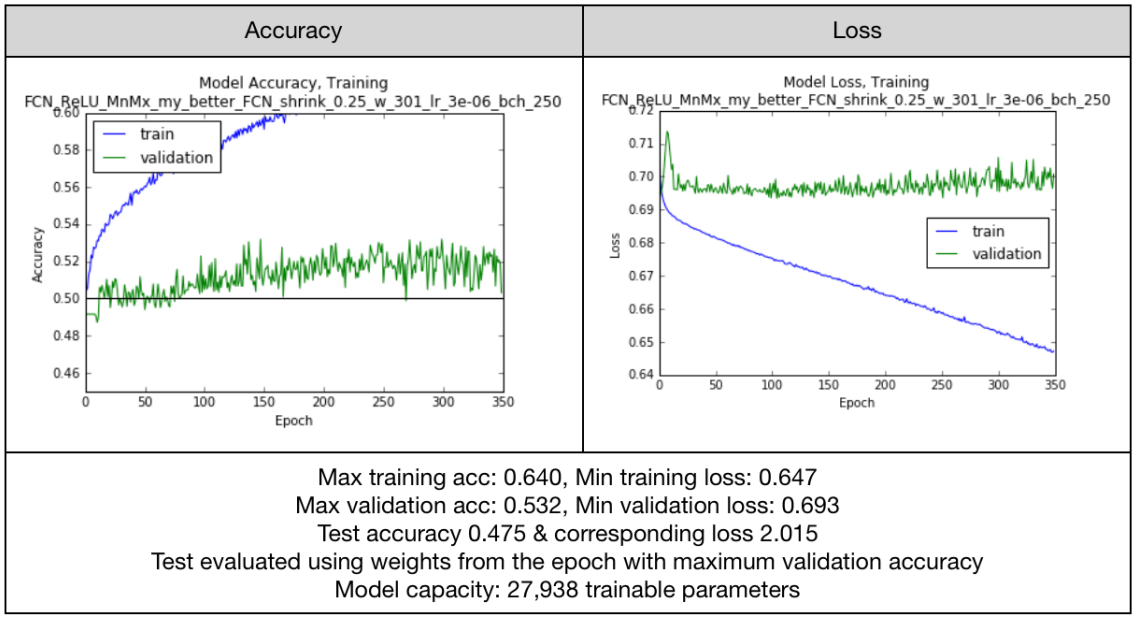


Figure 4.5: Results Summary for *My Better FCN* with *Min-Max* Normalization Learning rate fixed at 3e-6, look-back period is 301 min, batch size 250

4.3.3 Impact of Regularization and Dropout

Even though, *My Better FCN* did not suffer from overfitting to a similar degree as the previous architectures, learning stalled and the out-of-sample (i.e test) accuracy remained comparable to random. Therefore, adding regularization was attempted in an effort to further stabilize the solution, boost learning and the test performance. Re-applying *L2* regularization was attempted but it had no impact of the solution. Different lambda parameters controlling the 'strength' of regularization were tested, ranging from 0.1 to 1e-5.

Having achieved breakthrough results in notable CNN architectures, dropout was attempted instead. Dropout was added before the initial convolutional layer, after each consecutive hidden layer (i.e. *conv* + *ReLU* + *BN*) and before the final classification layer. Various configurations were tested and optimized using hyperas. It was found that for optimal results the initial and final dropout layers should be omitted i.e set to zero and the intermediate layers should be kept with a dropout

of circa 0.2. Figure 4.6 illustrates the results, it can be seen that during training this model preformed better than the variant without dropout. As expected, overfitting was reduced and the training and validation learning curves remained closer together. However the out-of-sample accuracy remained around 50%. Therefore, it cannot be concluded whether dropout improved the generalization of the model. On the other hand, dropout certainly did not detract from the solution, thus it was carried forward to the final model improvement stage.

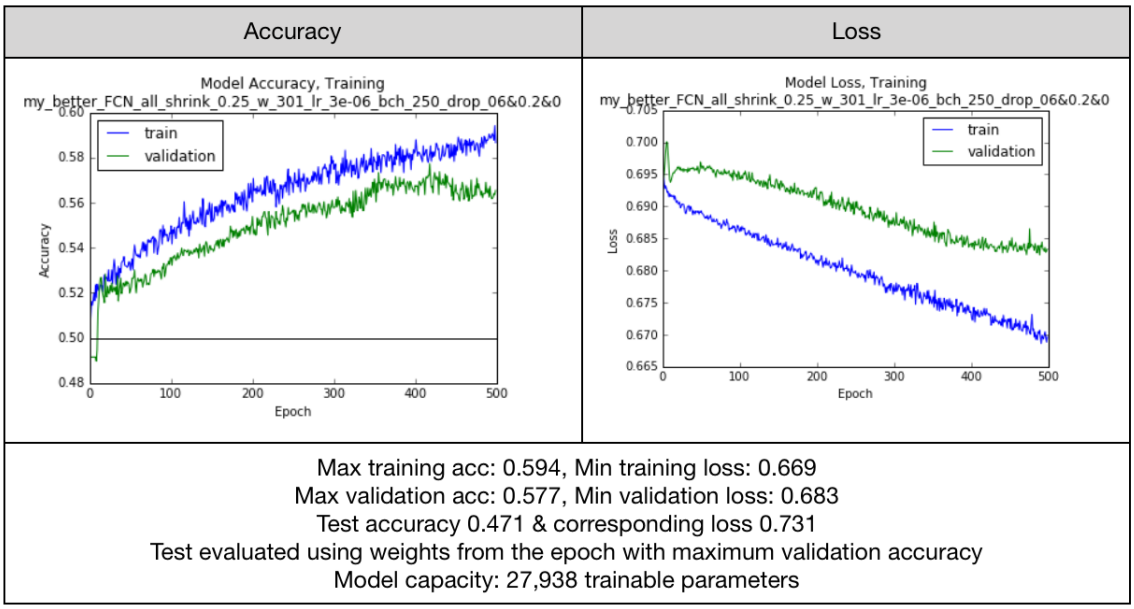


Figure 4.6: Results Summary for *My Better FCN* with Dropout
Learning rate fixed at 3e-6, look-back period is 301 min, batch size 250

4.3.4 Impact of Activation Function

Up to this point many avenues for the FCN's improvement have been explored with limited success. Thus, in order to further develop the *My Better FCN* changing the activation function was attempted as the last remaining possible modification.

When it comes to activation functions, the widely used ReLU was replaced with some of its more sophisticated variants. The exponential linear unit (eLU) was used but fared worse than ReLU. Some improvement in training results was noticed when employing the Leaky ReLU activation, shown on figure 4.7 alongside the ReLU. It can be seen that instead of setting gradients to zero for negative X values, the Leaky ReLU function preserves a slope α, which enables gradients to persist in

neurons instead of being permanently set to zero. As explained by Xu et al.(2015) [41] incorporating a nonzero slope for the negative part of the rectified activation unit consistently improves results, especially in cases where bottleneck layers are present. Such is the case in *My Better FCN* where the *conv1x1* layer is an effective bottleneck. Thus, in the final FCN variant attempted a Leaky ReLU activation was employed with the slope α set to 0.3, as per default settings. Due to time and computational resources constraints the α parameter was not tuned. In line with the preceding discussion, filter and kernel sizes, dropout and data normalization were kept unchanged. Figure 4.8 summarizes the results obtained from training this last model. It can be seen that on training and validation data this FCN variant was marginally weaker than its ReLU counterpart. On the other hand, on the test set Leaky ReLU performed marginally better. This results are not conclusive and the impact of the LeakyReLU is not as pronounced as was hoped. However, room for improvement definitely exists through tuning of the α parameter.

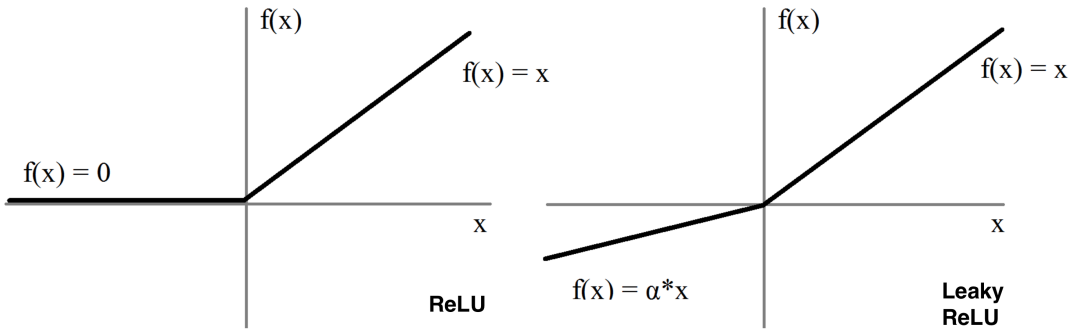
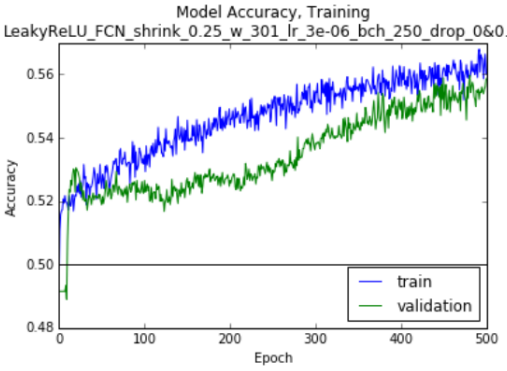
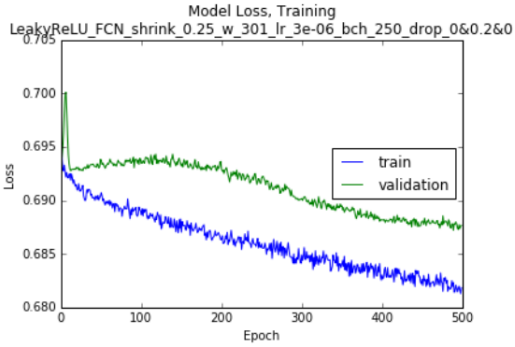


Figure 4.7: ReLU and Leaky ReLU
Adapted from [23]

Accuracy	Loss
	

Max training acc: 0.568, Min training loss: 0.681
Max validation acc: 0.560, Min validation loss: 0.687
Test accuracy 0.490 & corresponding loss 0.715
Test evaluated using weights from the epoch with maximum validation accuracy
Model capacity: 27,938 trainable parameters

Figure 4.8: *My Better FCN* with Leaky ReLU and Dropout
Learning rate fixed at 3e-6, look-back period is 301 min, batch size 250

4.4 Improved Residual Network (SResNet)

Taking into account the findings from the FCN model, the logical first step for the SResNet architecture was to reduce the filter numbers by a factor of 10. This resulted in a desirable, considerable reduction in model capacity. It should be noted that unlike the FCN, the SResNet contains 3 stacks of convolutional layers, with 3 layers in each stack and skip connections between stacks. As per the baseline model, filter numbers were kept constant in each stack and only altered from stack to stack. Thus the initial filter numbers were set to: $f_s = (12, 25, 25)$, where s indexes the stack. Kernel sizes varied within each stack and were set to: $k_i = (8, 5, 3)$ where i indexes the convolutional layer. In line with the original SResNet by Wang et al. (2016) dilation was not added and padding was kept at *Same*, which is a implementation requirement for the skip connections to work. The activation function used in the hidden layers was ReLU. Similarly to the FCN, Global Average Pooling was employed after the hidden layers prior to the final softmax layer. Table 4.2 and figure 4.9 show the results obtained and compare with the baseline SResNet model. From training scores it can be seen that *My SResNet* overfitted substantially less than its baseline counterpart. When it comes to best validation and test results these remained largely unchanged. However, the shapes of *My SResNet* learning curves were much improved: with validation accuracy similar to other experiments and a stable validation loss.

Table 4.2: Comparison of SResNet Models

Best refers to Max Accuracy and Min Loss

	Baseline		Modified SResNet	
	Accuracy	Loss	Accuracy	Loss
Training Best	0.991	0.072	0.796	0.796
Validation Best	0.547	0.693	0.543	0.690
Testing Best	0.504	0.721	0.508	0.796

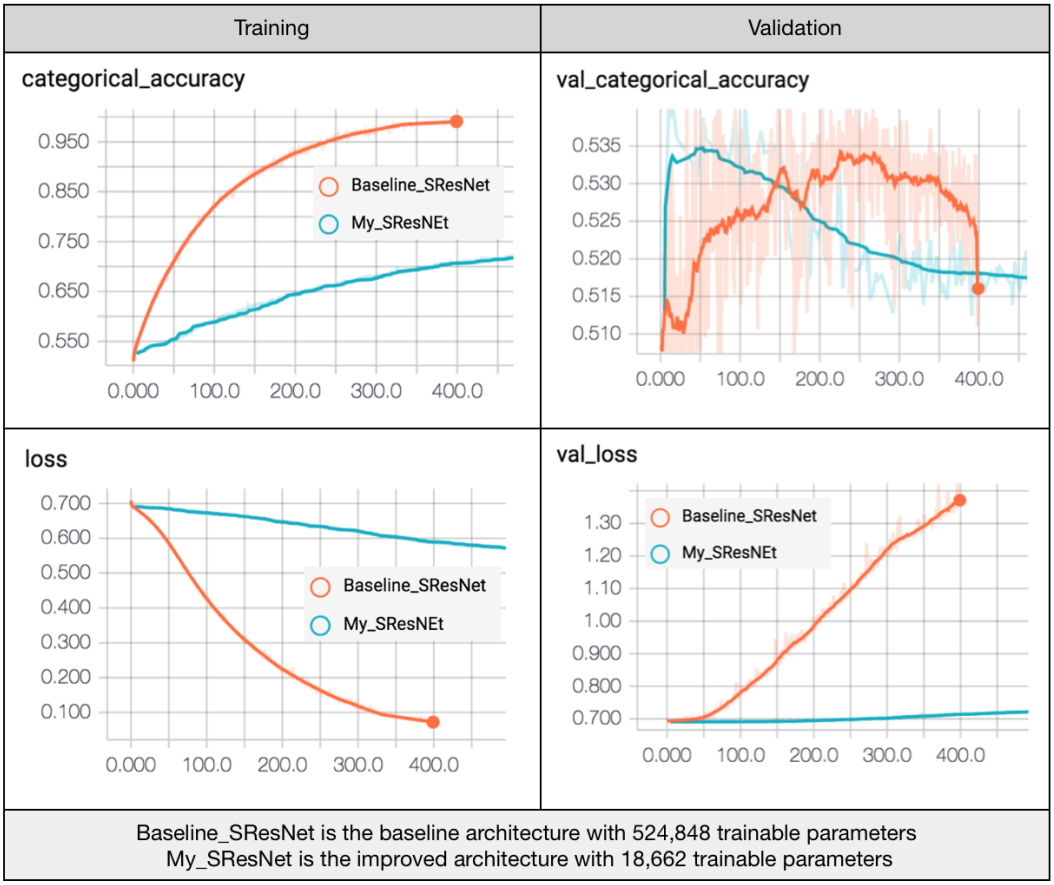


Figure 4.9: Comparison of all SResNet learning curves vs. Epoch

4.5 Chapter Conclusion

4.5.1 Comparison between Final FCN and SResNet Models

Comparing *My Better FCN* and *My SResNet* it should be noted that the FCN architecture has more parameters than the SResNet, this could be why it performs better on training and validation. On the other hand the SResNet achieves marginally improved results on the out-of-sample data. With less parameters and substantially less architectural changes *My SResNet* achieves a validation accuracy of just above 54%, pointing to the effectiveness of employing residual connections. From the learning curves *My Better FCN* is still superior to SResNet due to higher and stable validation accuracy. However, contrary to initial results, further work on the SResNet architecture appears to hold some promise, which is elaborated in section 5.2.3.

4.5.2 Test vs Validation Scores as Evaluation Metrics

Reflecting on the results presented thus far, it is clear that the test set scores were worse than the validation scores. It should be noted that, models were developed looking predominately at validation results as an indicator of performance. Thus, the desired improvement in validation results was achieved at least by the *My Better FCN* model. The poor out-of-sample performance does not necessarily mean the models did not learn. Instead this could point to a problem with the nature of the test set itself.

As explained before, the financial data is inherently noisy and consists of anomalous events of which no further knowledge is available. Sampling 'good' entry-points from the market price series is a crucial and complex task, certainly affected by the general behaviour of the stock market. This implies that the data is likely to exhibit a covariate shift meaning the distribution and the nature of the data has changed between 2010 and 2017. It should be noted that, since the data was split into: training, validation and test sets sequentially time order was preserved. Introducing a validation set, which sits between the training and test sets, created an effective time gap between the data the model was trained on and the data it was tested on. The sequential nature of the data-set was thus interrupted and the model was asked to predict over too large a time span, in which the characteristics of the data set have changed. This would explain why the test set results were consistently worse than random and justifies why validation results should be the focal evaluation metric. It is likely that if the validation set was omitted and the model was trained on all the data up to the end of 2016, the test results would mimic those seen in the validation set today.

Additionally, in other successful application of CNNs, such as image classification, data is reshuffled to promote learning. In the case of time-series and in the profession of trading this cannot be done as time-order carries information. This notion further strengthens the argument that 'standard' practices of splitting out a validation set and then evaluating performance looking at the test scores may not be the most suitable in the context of sequential data.

Chapter 5

Conclusion and Further Work

This chapter concludes on the research undertaken in this body of work (section 5.1). Section 5.2 suggests possible avenues for further work.

5.1 General Conclusion

Even though the FCN and SResNet models have shown strong performance on benchmark time-series classification data-sets their application to the financial data-set led to far less satisfying outcomes. This can be attributed to the notorious difficulty of distinguishing signals from noise in this type of data. Furthermore, the assumption that data is independent and identical distributed breaks down since long-timescale correlations exist and the data is non-stationary. For these reasons, to properly compare this work with literature, validation results should be emphasized over test results as more meaningful representations of the models actual perforce. Looking at validation scores it can be concluded that the modified FCN architecture *My Better FCN* outperforms the baseline models and *My SResNet* model. From the experiments undertaken it was found that the key components to the modified FCN's success were: the adjustment of number of channels, the introduction of dilation in the hidden layers, the addition of a final *conv1x1* convolutional layer and the introduction of dropout to prevent overfitting. Among all models, *My Better FCN* with a ReLU activation function achieved the highest validation accuracy of almost 58%. For financial time-series this can be considered a very good result, which would yield positive net profits if employed as a trading strategy alongside

suitable risk-management practices.

5.2 Suggested Further Work

5.2.1 Data Augmentation

Due to the large model capacity and the comparatively small size of the inherently noisy financial-data set, it is believed that the single most influential improvement to the results presented could come form increasing the amount of data available for training. This was already attempted by the creation of the *big* data-set described in chapter 3. However, the data augmentation technique employed did not produce desired improvements. Therefore, one avenue for further work is to focus on developing a data augmentation technique which effectively aids the training of the models. Similar augmentation has boosted training in image classification tasks, but remains a challenge for financial time-series due to data-set noise and the difficulty of identifying which data features to augment, while preserving temporal order. Thus, the subject itself is vast enough to give rise to another research project.

5.2.2 Further FCN Improvements

As was demonstrated in chapter 4, many avenues for improving the FCN have been attempted with limited success. However, one remaining naive way to improve results would be to undertake tuning of the α slope parameter in the LeakyReLU activation, which was not attempted due to time and resource constraints. A more elegant way to achieve the same is to employ the Parametric Rectified Linear Unit (PReLU) instead of Leaky ReLU. PReLU learns the optimal slope α via backpropagation during training[42] eliminating the need for tuning. This was attempted but encountered some difficulty in the Keras implementation. Therefore, another avenue for improvement is to develop a bespoke solution to implement PReLU in Keras correctly.

5.2.3 Further SResNet Improvements

In retrospect, it could well be that further work on the SResNet would yield an architecture superior to the FCN. This is due to the visible effectiveness of residual

connections employed by the SResNet, which allow for the reduction of model parameters without critically detracting from validation or test accuracy. Since the SResNet architecture replicates the hidden part of the FCN architecture in three blocks, it is logical to apply findings from the FCN's development to the SResNet. Further steps could thus include introducing: dilation, a 4th convolutional layer in each stack or changing the activation function to LeakyReLU. It should however be noted that this experiments will be considerable more time and resource consuming than their counterparts on the FCN model.

5.2.4 Other CNN Architectures

As claimed by Borovykh et al. (2017), financial time-series data can be classified with a high degree of accuracy if time series for various securities are conditioned on one another. The authors achieve this by implementing a conditional WaveNet model, which could be an alternative to the FCN and SResNet architectures trailed in this work. The employment of this model would require a departure from using information about one security (crude oil) in favour of a multivariate data set with at lest a few securities. According to Borovykh et al. (2017), this approach improves classification results thorough also learning the co-dependencies between securities in time. The use of the WaveNet model akin to Borovykh et al. (2017) requires a considerable change in the problem formulation, but if other avenues for improvement fail to produce desirable results it could well be the only remaining way forward; implying that the financial data available can only be classified conditionally.

Appendix A

Baseline Models Validation

Multiple variants of the FCN and SResNet were trailed to identify suitable settings and hyperparameters, which allowed for a close reproduction of original results reported by Wang et al. (2016). In addition to information provided in Chapter 3, it was determined that:

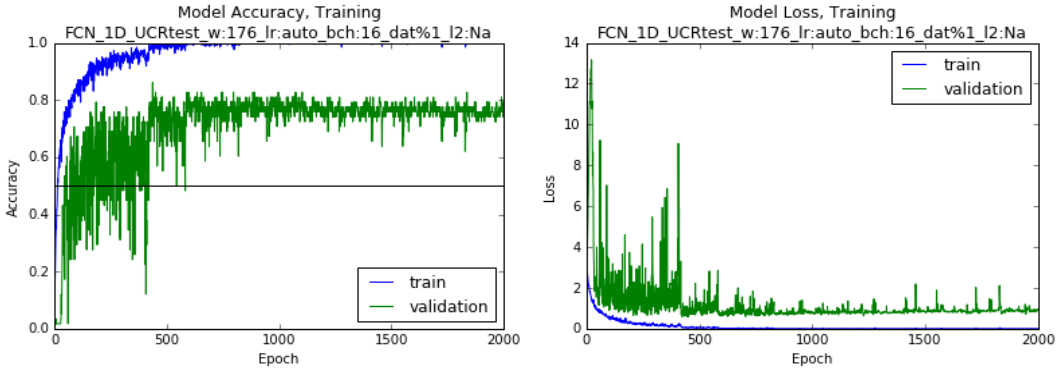
- Padding was set to *Same* throughout the network
- Striding and dilation were kept at 1, i.e. were not used
- No explicit weight initialisation was used
- No regularisation other than batch normalisation was used
- Default Adam optimizer with a learning rate of 0.001 was used together with the ReduceLROnPlateau callback, set to monitor validation loss.
- The Global Pooling layer was in fact a Global Average Pooling layer as opposed to a Global Max Poling layer.

Finally, the best results were obtained using batch size 16 and a full look back period equal to the length of the time-series itself. Two variants of the FCN and SResNet architecture implementations were tested: one using the new and faster Keras *Conv1D* command and the other using the established *Conv2D* command. For the Adiac data-set Wang et al. (2016) report the FCN to achieve 0.143 error, which is equivalent to 0.857 accuracy. Whereas the ResNet achieves 0.174 error,

which is equivalent to 0.826 accuracy.

FCN with Conv1D:

The results of my FCN using *Conv1D* are depicted in Figure A.1 and can be summarized as follows: Average Training: Accuracy 0.9700 +- 0.0841, Validation Accuracy: 0.7068 +- 0.1384, Loss: 0.1070 +- 0.3012, Validation Loss: 1.1694 +- 1.1978 Test Scores: Accuracy 0.8261 , Loss 0.8752

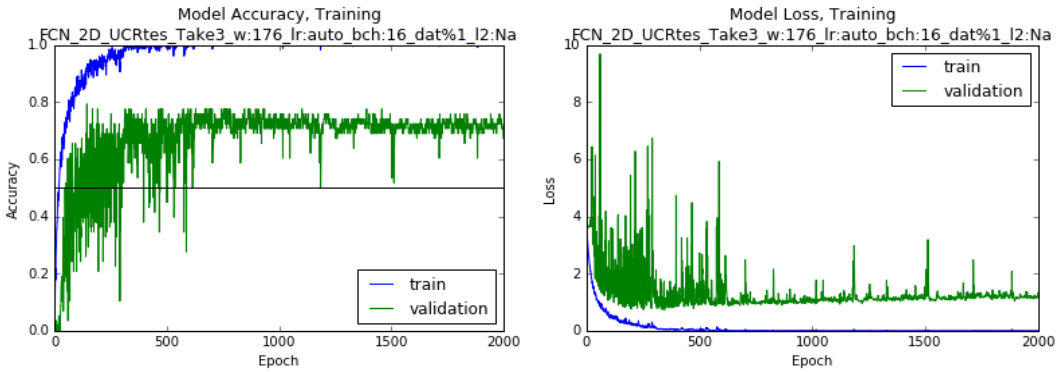


(a) Model Accuracy **(b)** Model loss

Figure A.1: Validation Results for FCN-Conv1D on Adiac UCR data-set

FCN with Conv2D:

The results of my FCN using *Conv2D* are depicted in Figure A.2 and can be summarized as follows: Average Training: Accuracy 0.9739 +- 0.0898, Validation Accuracy: 0.6696 +- 0.1330 Loss: .1024 +- 0.3471, Validation Loss: 1.3164 +- 0.6621 Test Scores: Accuracy 0.8159 , Loss 0.9945

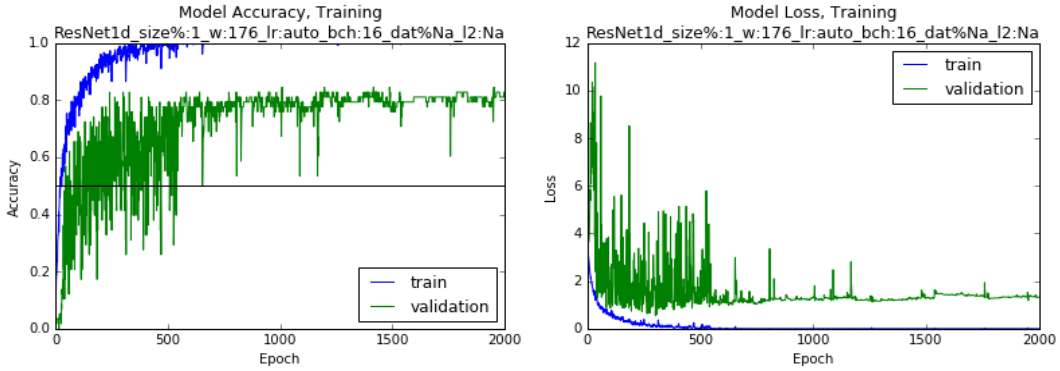


(a) Model Accuracy **(b)** Model Loss

Figure A.2: Validation Results for FCN-Conv2D on Adiac UCR data-set

SResNet with Conv1D:

The results of my SResNet using *Conv1D* are depicted in Figure A.3 and can be summarized as follows: Average Training: Accuracy 0.9684 +- 0.0984, Validation Accuracy: 0.7196 +- 0.1556, Loss: 0.1048 +- 0.3389, Validation Loss: 1.5436 +- 0.9530 Test Scores: Accuracy 0.7980 , Loss 1.2759

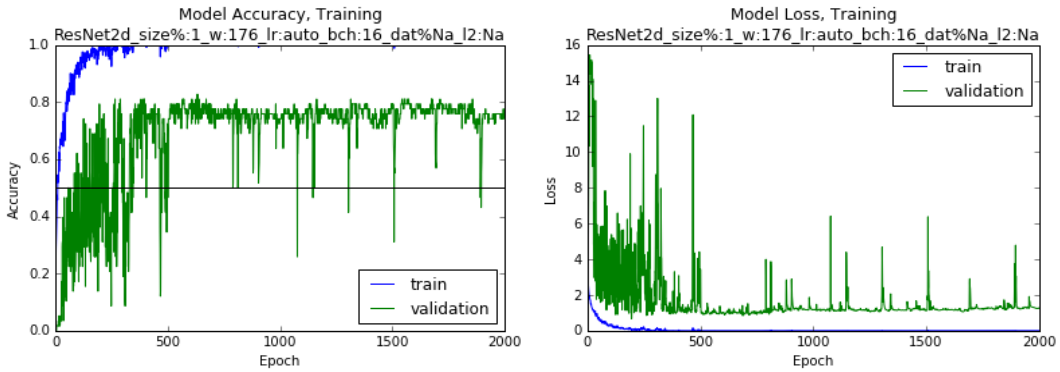


(a) Model Accuracy **(b)** Model Loss

Figure A.3: Validation Results for SResNet-Conv1D on Adiac UCR data-set

SResNet with Conv2D:

The results of my SResNet using *Conv2D* are depicted in Figure A.4 and can be summarized as follows: Average Training: Accuracy 0.9837 +- 0.0691, Validation Accuracy: 0.6869 +- 0.1665, Loss: 0.0591 +- 0.2482, Validation Loss: 1.7376 +- 1.7441 Test Scores: Accuracy 0.7852 , Loss 1.1607



(a) Model Accuracy **(b)** Model Loss

Figure A.4: Validation Results for SResNet-Conv2D on Adiac UCR data-set

CONCLUSION

Both FCN and SResNet implementations achieved similar results, with *Conv1D* slightly outperforming *Conv2D*. Thus, *Conv1D* was employed going forward. Given the random weights initializations and possible subtle differences in implementations, the results above are a satisfactory match with results reported by Wang et al.(2016). Furthermore, my models were evaluated using the final weights recorded after 2000 epochs of training, as opposed to the *best* weights i.e. corresponding to the epoch with highest accuracy. If the best weights were used then testing results would surely match those reported by Wang et al. (2016) even more closely. However, this was not the focal point of this body of work.

Appendix B

FCN Initial Architecture Testing

Below is an extended overview of the key findings obtained in the preliminary phases of training and improving the baseline FCN model. This is an enhancement to the summary given in chapter 4. However, significantly more results were obtained to support the model development and parameter selection. All results reported in this section were obtained using the Z normalized *small* data set without regularization.

B.1 Initial Hyperparameters Exploration

Initially, it was noted that a reduction if filter size improved results. Figure B.1 illustrates this for a long run of the baseline FCN with and without reduced filter size. This finding supports the reduction of filter size in the modified FCN model. Thus, the model used for further exploration is : Baseline FCN with reduced filter sizes: $f_i = (12, 25, 12)$, kernel sizes: $k_i = (8, 5, 3)$ dilation $d_i = (1, 1, 1)$ where i indexes the convolutional layer. Padding set to *Same*.

Figure B.2 demonstrates the impact of changing the look-back period (window-size), while keeping other parameters fixed. Evidently, employing the largest window-size improved results and was be adopted going forward. The effect of changing the learning rate and batch sizes was also examined. Batch sizes ranging from 16, as employed by Wange et al.(2016) to 2000 were trialed, but best results were achieved in the range between 250-512 which as further explored. The adverse effects of reducing the learning rate to 5e-6 can be seen in figure B.3.

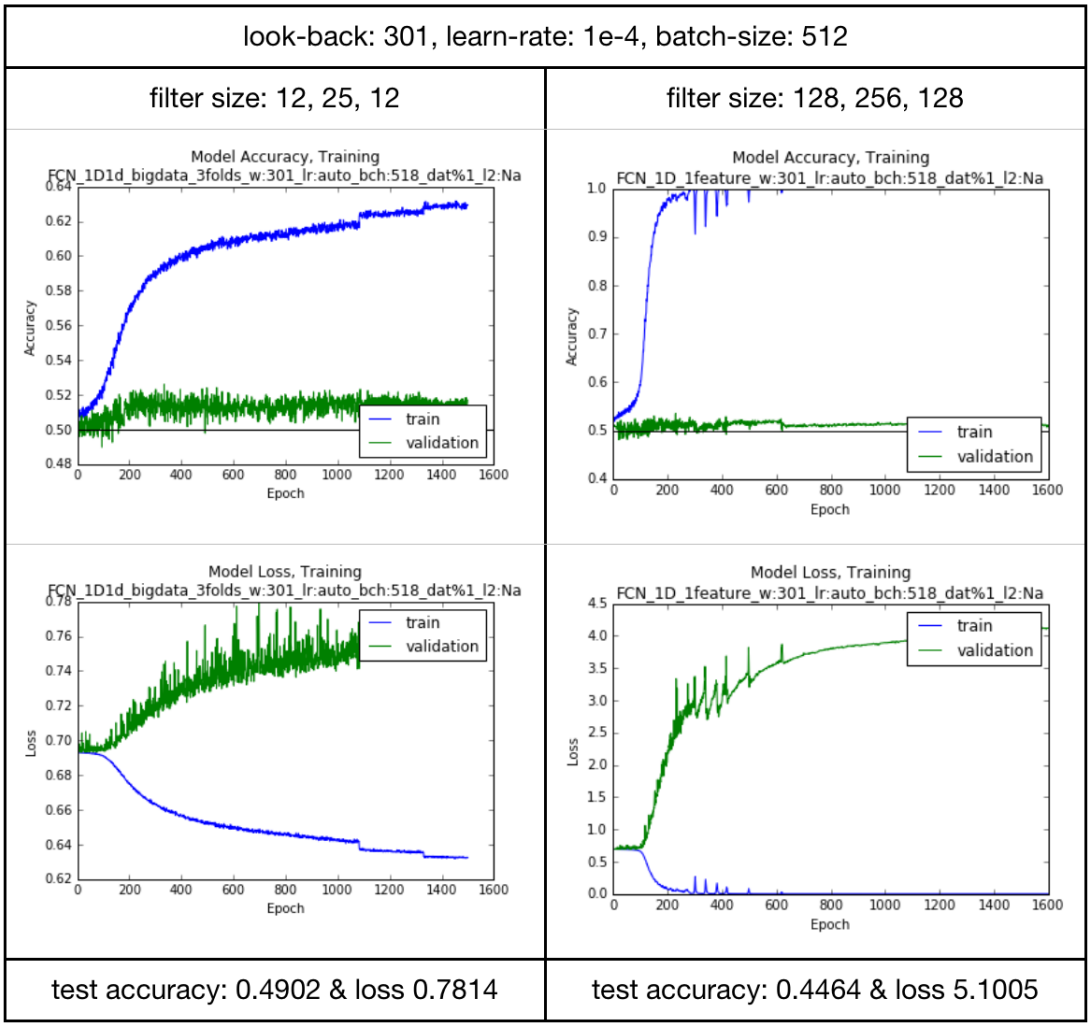


Figure B.1: Impact of filter size on training
Learning rate fixed at 1e-4 and batch size of 512

B.2 Impact of Dilation

Figure B.3 demonstrates the effect of reducing the learning rate and introducing dilation into the 2nd and 3rd convolution layers, for batch sizes 256 and 512. It can be seen that results generally, worsen with the reduction of the learning rate. However, even with this worsened results, the beneficial effects of dilation can be noticed. Additionally, it can also be seen that batch size 256 is better than 512. Figure B.4 illustrates the effect of further increasing the dilation in the final convolutional layer, in line with WaveNet [32]. However, this amount of dilation did not perform well, possibly due to the networks compact size as compared with the

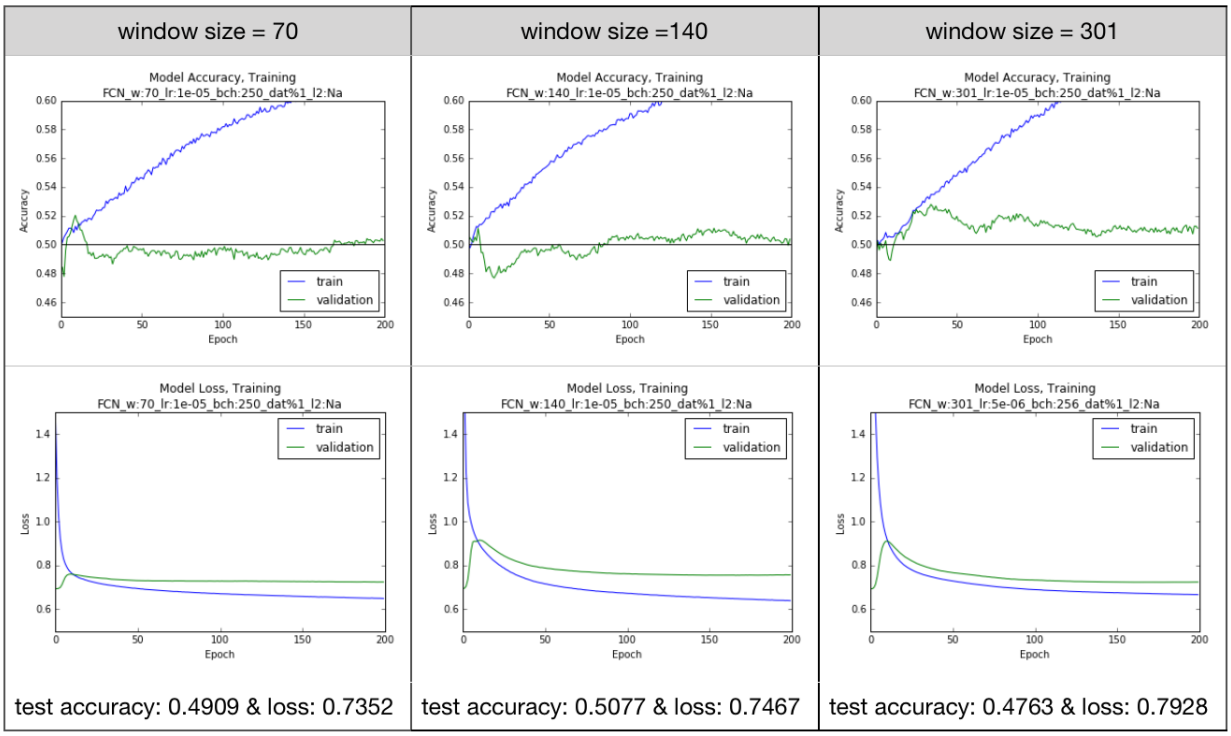


Figure B.2: Impact of look-back period on training
Learning rate fixed at 1e-5 and batch size 250

original WaveNet architecture. Based on this results, the modified model was set to: Baseline FCN with reduced filter sizes: $f_i = (12, 25, 12)$, kernel sizes: $k_i = (8, 5, 3)$ dilation $d_i = (1, 2, 2)$ where i indexes the convolutional layer. Padding was set to *Same*. The next steps were to further examine the impact of learning rate as well as the padding setting.

B.3 Impact of Learning Rate and Padding

Since, the learning rate of 1e-5 yielded better learning curves than the rate of 5e-6, this was reverted to and tested with the inclusion of dilation: $d_i = (1, 2, 2)$. Furthermore, the model was run with padding set to *Same* as well as *Valid*, the results can be seen in Figure B.5. Although, on first glance the two padding variants appear similar, *Valid* achieves better out of sample (i.e. test) scores.

The impact of learning rate and padding on results was examined further, this time testing for learning rates below 5e-6. Figure B.6 illustrates the effects of reducing the learning rate to 3e-6, which has been found to be optimal. Reductions

beyond this point yielded worse results. It should be noted, that padding *Valid* was employed in the results presented as this fared considerably better than padding *Same* under reduced learning rates. Therefore, padding *Valid* was was employed in the modified FCN architecture. Figure B.6 results were obtained for 800 epochs instead of the shorter 200 epoch runs presented before. Generally, all runs performed for this extended amount of epochs tended to overfit in later stages. However the initial stages of training achieved comparatively very strong results; consistently abode 0.52 accuracy. The intermediate batch size of 375 fared the best also in terms of the validation scores (loss and accuracy).

B.4 Impact of Global Pooling Layer

Finally, the impact of the type of Global Pooling layer was examined. It should be noted that the results presented thus far have been obtained using Global Max Pooling. However, validation of Wang et al. (2016) results (Appendix A) undertaken in parallel with these explorations had revealed, Global Average Pooling was employed in the baseline FCN architecture. Therefore, switching pooling layer type was considered; Figure B.7 shows the most promising architecture and parameter combination, but with Global Max Pooling replaced by Global Average Pooling. As can be seen, this modification does not improve results: test accuracy and loss worsen and the model does not learn. Therefore, the model put forward after this section, and referred to as *My Baseline FCN* is summarized as follows: Baseline FCN with reduced filter sizes: $f_i = (12, 25, 12)$, kernel sizes: $k_i = (8, 5, 3)$ dilation $d_i = (1, 2, 2)$ where i indexes the convolutional layer. Padding set to *Valid* and using Global Max Pooling. The optimal parameters are: leaning rate: 3e-6, batch size between 250 and 400, maximum look-back period.

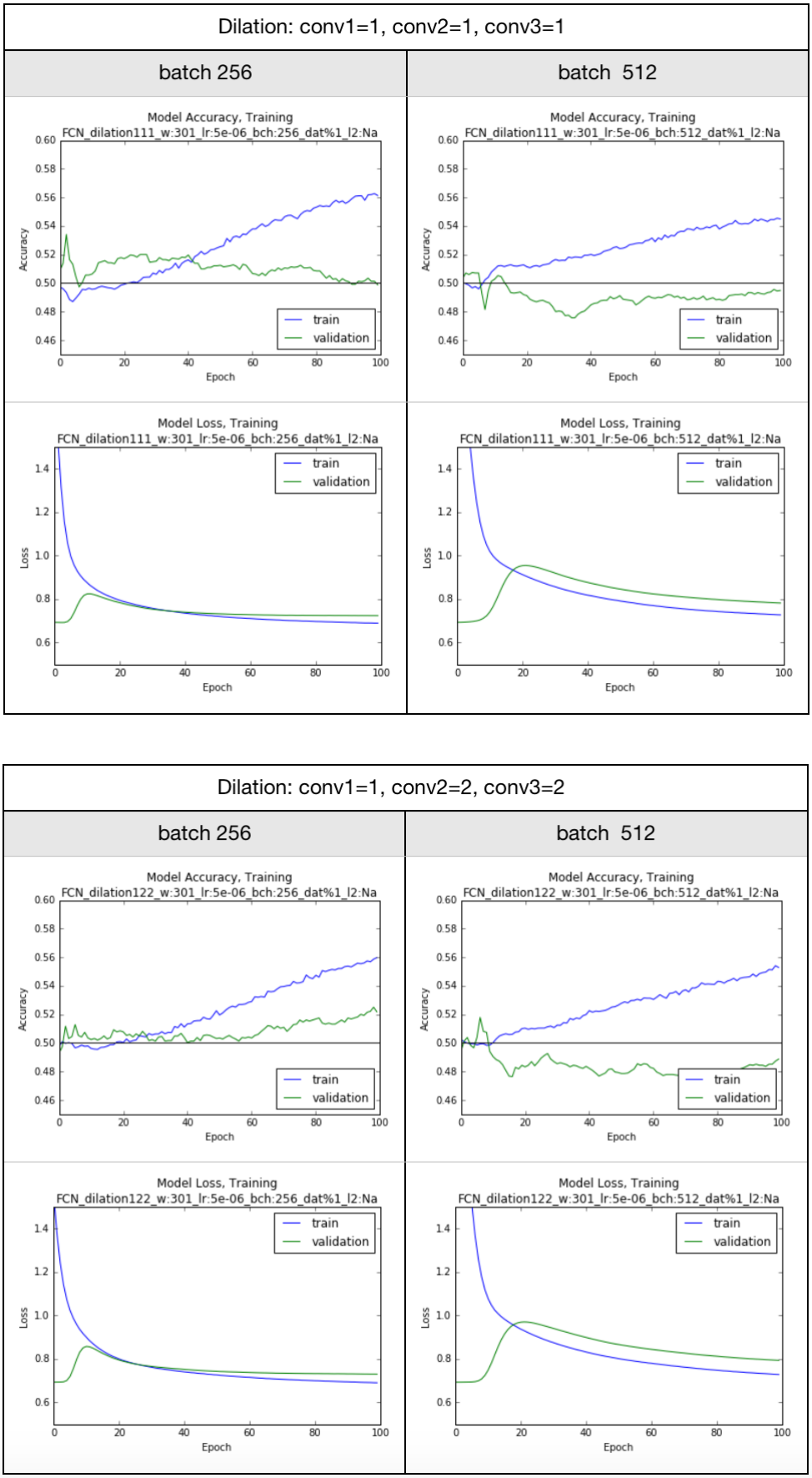


Figure B.3: Comparison of effects of dilation and batch size
Learning rate fixed at 5e-6 and look-back period of 301 min

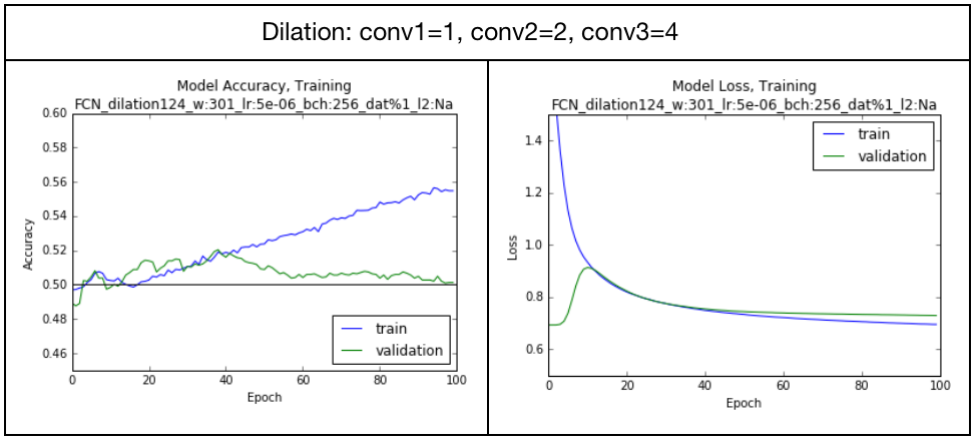


Figure B.4: Effect of increasing dilation in the final convolutional layer
Learning rate fixed at 5e-6, look-back period: 301 min, batch size: 256

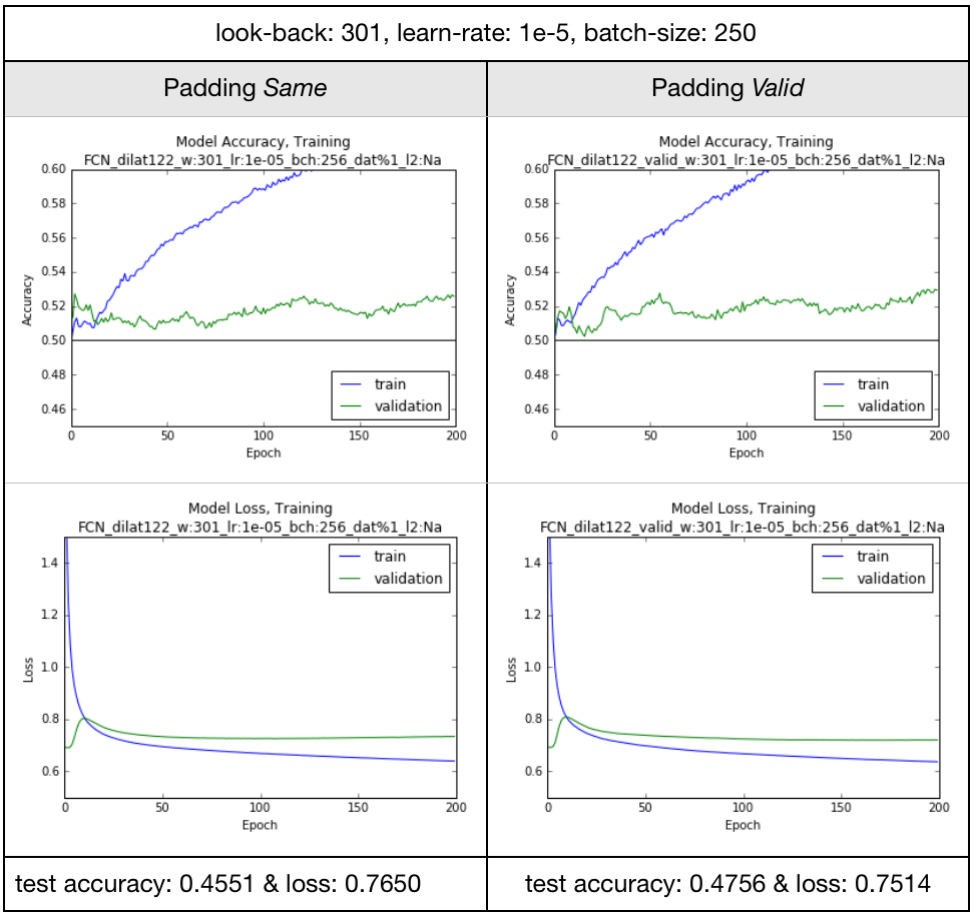


Figure B.5: Effect of learning rate and padding
Learning rate fixed at 1e-5, look-back period: 301 min, batch size: 250

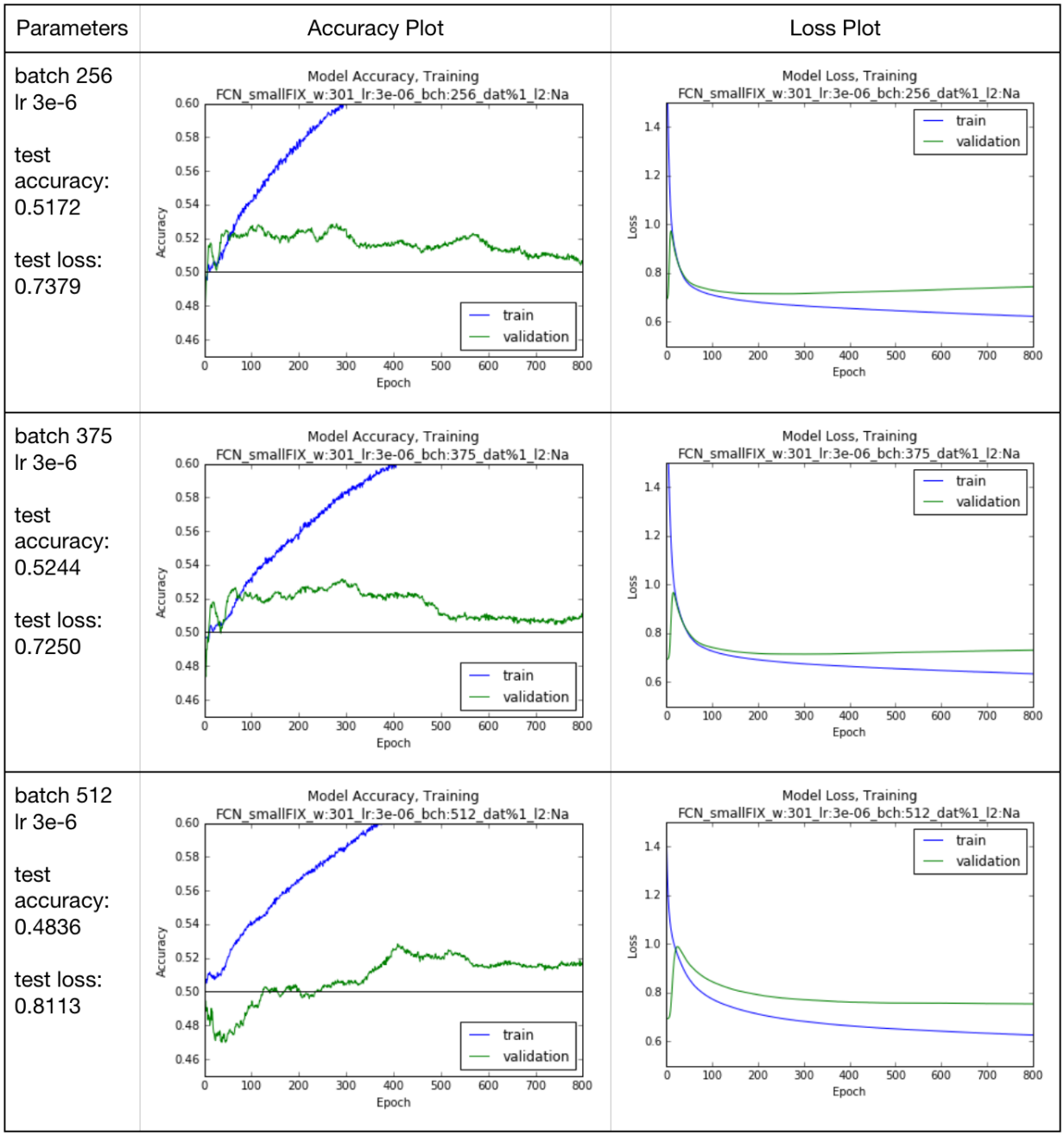


Figure B.6: Reduction in learning rate and increased epoch length for different batch sizes
Learning rate reduced to 3e-6, look-back period: 301 min, dilation: $d_i = (1,2,2)$ and padding *Valid*

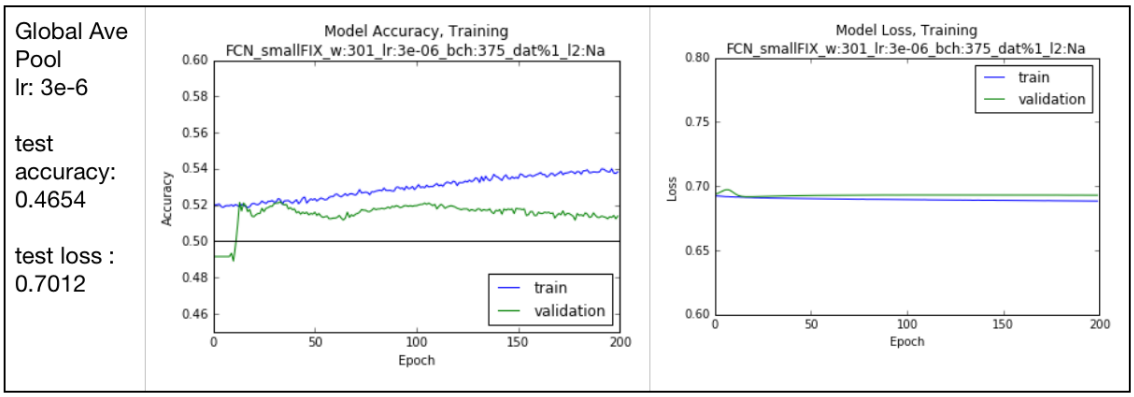


Figure B.7: Impact of change from Global Max Pooling to Global Average Pooling
Learning rate: 3e-6, batch size: 375, look-back period: 301 min, dilation: $d_i = (1,2,2)$ and padding *Valid*

Bibliography

[1] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1578–1585. IEEE, 2017.

[2] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[4] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, Online First, 2016.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[7] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, Jan 1970.

[8] Simon Osintero. Neural networks: Foundations, Jul 2017. Advanced Machine Learning Lecture at University College London.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Yann Le Cun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[11] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[12] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[13] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[15] Sven Behnke. *Hierarchical neural networks for image interpretation*, volume 2766. Springer Science & Business Media, 2003.

[16] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[17] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[18] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[19] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[21] Convolutional Neural Networks, author=Simonyan, Karen, year=2017, month=Jul, note = ”advanced machine learning lecture at university college london”.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[23] Nikolaos Sarafianos. Deep renaissance: How i approached deep learning. `https://nsarafianos.github.io/dlintro`.

[24] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.

[25] Guy P Nason. Stationary and non-stationary time series. *Statistics in Volcanology. Special Publications of IAVCEI*, 1:000–000, 2006.

[26] Masashi Sugiyama, Makoto Yamada, and Marthinus Christoffel du Plessis. Learning under nonstationarity: covariate shift and class-balance change. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5(6):465–477, 2013.

[27] Ling-Yun He and Feng Zheng. Empirical evidence of some stylized facts in international crude oil markets. *Complex Systems*, 17(4):413–426, 2008.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[29] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[33] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data augmentation for time series classification using convolutional neural networks. In *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 2016.

[34] Christopher Olah. Understanding lstm networks. *GITHUB blog, posted on August*, 27:2015, 2015.

[35] Ashwin Siripurapu. Convolutional networks for stock trading. `http://cs231n.stanford.edu/reports/2015/pdfs/ashwin_final_paper.pdf`, 2015.

[36] Roni Mittelman. Time-series modeling with undecimated fully convolutional neural networks. *arXiv preprint arXiv:1508.00317*, 2015.

[37] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification. *arXiv preprint arXiv:1603.06995*, 2016.

[38] Rob J Hyndman. Time series data library. `http://robjhyndman.com/TSDL/`, Jul 2010.

[39] Genevieve B Orr and Klaus-Robert Müller. *Neural networks: tricks of the trade*. Springer, 2003.

[40] S Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

[41] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.