A Comparison of Modern Deep Learning Methods for Time-series Trend Prediction

Morgan Blem
BLMMOR002
University of Cape Town

ABSTRACT

In this study Temporal Convolutional Networks (TCN), Convolutional Neural Networks (CNN), and Multilayer Perceptrons (MLP) were compared on their ability to predict trends in time series data using only a sequence of historical trends. Three datasets were used for the study: daily Cape Town air temperatures, daily S&P500 closing prices, and a household voltage dataset. Input data was run through a median filter to reduce noise before being segmented into trend lines using the sliding window algorithm [19]. Two experiments were run: the first was to use the traditional approach of training a model to predict both components of a trend simultaneously, and the second was to train independent models to predict trend slope and trend duration separately. The best performance was measured when using a TCN with a single prediction approach - though all DNNs performed well on the problem. Across all DNNs, there was an average RMSE reduction of 3.1% when using the proposed single prediction approach.

1 INTRODUCTION

Time series modelling is a task that has applications in a wide variety of scientific fields. While most time-series problems are usually classification or point forecasting problems, there are cases where the more general directional nature of trends offers more use than point forecasting [12]. Some basic examples may be for guiding the timing of buying or selling stocks, preparing for changes to weather patterns, or predicting the rate of a company's income growth over a period. Numerical modelling techniques, which have historically been used as a traditional approach for forecasting time series data [23], are an excellent choice in cases where there is a high level of theoretical understanding, though in many highly complex problems theoretical understanding can be limited due to the interactions of multiple systems. [8].

With the emergence of deep learning and other machine learning methods, modelling of complex systems has seen a gradual shift from analytical methods to machine learning methods due to their high capacity for modelling complex non-linear functions and their ability to make generalisations when dealing with unseen data [8]. As a consequence of there being infinitely many possible architectural ANN configurations, however, researchers have struggled to develop optimal network topologies and learning algorithms [16].

University of Cape Town BSc (Honours) Research Project, June 2021, Cape Town, South Africa
2021. . DNN-TRND. . .

Fueled to some degree by public interest towards artificial intelligence and data science, there has been a large volume of recent research on the topic of deep learning. In 2018 Bai et al. [2] described a generic architecture for convolutional sequence prediction which incorporated recent advancements in convolutional architectures. They referred to this architecture under the umbrella term of Temporal Convolutional Networks. Only one study prior to this has tested TCNs in the problem of trend prediction [6], though it was done using a modified TCN architecture specific to stocks. As such, TCNs still remain relatively unexplored in this problem. For most deep learning practitioners, recurrent neural networks (RNNs) are the go to choice for one dimensional time series data, though recent results indicate that convolutional architectures, such as the CNN and TCN, can outperform recurrent networks on these tasks [2].

Due to the widespread list of applications for time-series trend prediction, effective solutions are needed so that we may further advance our capacity to predict trends, especially in problems where theoretical complexity may limit our ability to create numerical models. This study aims to review the use of two convolutional methods: convolutional neural networks and the more recent temporal convolutional networks as an alternative to the more straightforward approach of using a multilayer perceptron, in the problem of time series trend prediction - as well as test an alternative approach to the one used by Kouassi & Moodley [13], wherein two seperate models are used for the prediction of each component of a trend - which we refer to as a single prediction approach.

1.1 Problem formulation

We begin by defining a Time-Series as follows:

$$X = \{x(t_0), x(t_1), x(t_2), ..., x(t_i)\}$$

where X is a random variable and each x is a real observations at time t [7]. The trend sequence T of the random variable X is defined as follows:

$$T = \{< l_1, s_1 >, < l_2, s_2 >, ..., < l_i, s_i >\}$$

where T is obtained by performing a piecewise linear approximation on X [13]. The values l_i and s_i represent the duration and slope of the trend. The neural network is then tasked with predicting the next sequence $<l_{i+1}, s_{i+1}>$ in the trend sequence [13].

1.2 Research Questions

(1) What performance improvements, if any, do TCNs offer over CNNs and MLPs when being used to predict trends in time-series data?
(2) Do two single prediction models offer better predictive capability than one model that predicts both slope and trend?

2 BACKGROUND AND RELATED WORK

2.1 Multilayer Perceptron (MLP)

2.1.1 Background. The MLP is a simple feed-forward DNN architecture which uses only fully connected layers. It has been used since the 1970s, and is commonly referred to as a vanilla DNN. Due to its simplicity, the method is flexible and can be easily re-purposed for a multitude of tasks and, as such, has seen large historical use in time-series applications [8]. Despite its simplicity, the MLP has been shown to be an effective method for time-series forecasting, outperforming many other machine learning methods when tested by Ahmed et al. 2010 [1].

2.1.2 Performance in time series forecasting. Related to forecasting, one recent study by Galicia de Castro et al. 2018 [24] found the MLP to outperform ML methods such as linear regression, gradient-boosted trees, and random forests - though the MLP was not compared to other DNNs. A recent large scale study by Lara-Ben ́ıtez et al. 2021 [16] compared the performance of various DNN methods over multiple time-series forecasting problems. The study found that MLPs provided poor predictive performance when compared to RNNs, LSTMs, CNNs, and TCNs - performing worse than these for all 12 datasets tested. In general, for time-series forecasting, the MLP performs worse than most alternative DNN architectures though usually better than most other ML methods.

2.1.3 Performance in time series trend prediction. When applied to the more relevant problem of trend prediction, there is less published research on its performance. Li et. al 2017 [18] compared the MLP to various DNNs and other ML methods in a time-series trend prediction problem where classification was used to identify trends of a constant length (unlike this study, wherein regression is used to give the slope angle and duration, which is not constant). The MLP outperformed all ML methods, though performed worse than the LSTM and RNN which were the only other DNN methods in the study. In a study by Kouassi & Moodley. [12], which follows a similar experimental method to this study, the MLP performed similarly to RNNs, LSTMs and CNNs, but was outperformed by TreNet.

2.2 Convolutional Network (CNN)

2.2.1 Background. The CNN builds on the MLP through the inclusion of convolutional layers in addition to the fully connected layers. It was originally designed for image-processing, though it can be adapted to process 1D signals. CNNs are considered state-of-the-art for a variety of problems such as image recognition and natural language processing as they are able to pull high level features from input data using its convolution layers. Alternative configurations for CNNs have been proposed such as gated convolutional networks used by Dauphin et al. [5], and fully convolutional networks which contain only convolutional layers and no fully connected layers. In this study, the traditional CNN architecture using both convolutional layers and fully connected layers was used.

2.2.2 Performance in time series forecasting. A large amount of recent work has been done on CNNs capacity for time-series forecasting due to their ability to perform similarly to state-of-the-art methods: Kanniainen et al. 2017 [25] found CNNs to outperform SVMs and MLPs in stock price forecasting, Huang et al. 2018 [14] found CNNs to outperform LSTMs and MLPs in an energy load dataset, and the large scale study by Lara-Ben ́ıtez et al. 2021 [16] found that CNNs had the best speed/accuracy tradeoff, making them a good choice for realtime operations. Additionally, the study found CNNs to be one of the most performant models (though on aggregate it was outperformed by LSTMs in this study).

2.2.3 Performance in time series trend prediction. Related to trend prediction, less studies have been carried out. Kouassi & Moodley [13] compared CNNs accuracy in trend prediction to MLPs and LSTM networks and found the CNN to be the most performant in 2 out of 4 datasets. Another study conducted by the same authors ([12]) tested CNNs in a similar experiment against LSTMs, MLPs and TreNet and found them to perform similarly to LSTMs and MLPs, though worse than TreNet. Overall, the performance of CNNs in trend prediction tends to be high, meaning it should be considered when deciding on an architecture for solving the problem.

2.3 Temporal Convolutional Network (TCN)

2.3.1 Background. Bai et al. 2018 described a generic convolutional architecture for sequence prediction [2] which was informed by, but still unique from, recent convolutional architectures for sequential data such as WaveNet (by van den Oord et al., 2016 [26], for generating raw audio waveforms), ByteNet (by Kalchbrenner et al., 2016 [11], for machine translation), a gated convolutional network (by Dauphin et al., 2017 [5] used for language modelling) and a convolutional encoder model (by Gehring et al., 2017 [9] for neural machine translation). The model can be described as a Fully Convolutional Network (FCN) that uses causal convolutions and dilated convolutions. Causal convolutions are described as convolutions where an output at time t is convolved only with elements from time t and earlier in the previous layer [2]. Dilated convolutions are a method proposed by van den Oord et al. 2016 [26] which aim to increase the receptive field of convolutions. The TCN was shown to outperform state-of-the-art methods in a variety of sequence modelling tasks [2].

2.3.2 Performance in time series forecasting. Subsequent research testing TCNs in time series forecasting problems found TCNs to have strong performance. In a study by Bohte et al. 2019 [3] TCNs outperformed LSTMs on financial data forecasting. Two other studies by Liu et al. 2019 [27] and Lara-Benítez et al. 2020 [15] tested TCNs on meteorology related and energy related time series respectively. Both studies repeated the finding by Bohte et al. 2019, and saw TCNs outperforming LSTMs. In another study by Chen et al. [4] TCNs outperformed simple RNNs in retail sales forecasting.

2.3.3 Performance in time series trend prediction. Due to the recency of the model, there has only been one study prior to this testing TCNs in the problem of trend prediction. This study, conducted by Chen et al. 2019 [6], used a modified knowledge driven TCN model and found TCNs to greatly outperform LSTMs and CNNs. It should be noted that the methodology differs greatly to this study, and a study that uses only past trends for trend prediction has not yet been conducted for TCNs.

3 METHODOLOGY

3.1 Datasets

Three datasets were used for the experiment:

(1) Cape Town daily surface air temperatures dataset (fig. 7) containing 9266 datapoints adapted from a Kaggle dataset of global city weather data. The dataset was coarsely segmented into 72 trends. It is a volatile and seasonal dataset.
(2) Daily closing prices of the S&P 500 (fig. 8) containing 3904 datapoints collected from Yahoo Finance. The dataset was finely segmented into 1036 trends. It is a slightly volatile dataset with very low seasonality.
(3) Household voltage dataset (fig. 9) from the UCI Machine Learning Repository truncated to 40000 points. The dataset was segmented into 425 trends. It is a highly volatile and seasonal dataset.

3.2 Data Preprocessing

Datasets were preprocessed using four steps which include: missing data imputation, min-max normalization, median filtering, and piecewise linear segmentation. fig. 1 shows a visual summary of these processes.

3.2.1 Missing Data Imputation: This step was performed only on datasets which contained missing values. In most datasets all timestamps are present and missing values are usually denoted by either a "?" or an empty field. In some cases, however, the date must be used to detect when an observation is missing: a larger than expected timestamp difference between two points implies an observation is missing. In these cases, the most recent known value is repeated for the missing point.

3.2.2 Min-max Normalization: Due to the inconsistent magnitudes found across datasets, min-max normalization was used to ensure all datasets were at an equivalent scale. This normalization method scales inputs into a range between 0 and 1 while preserving the relational integrity of the data, and since there are no negative values in any of the datasets this is an appropriate range. Mathematically, a transformed element x_t', can be expressed as:

$$x_t' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

The resulting values were then scaled by a factor of 100 to ensure variation in the slope of trend lines, as a scale from 0 to 1 resulted in poor segmentation performance due to the small angle of inclination between normalized points: using a scale of 0 to 1 the largest possible slope between two points is *45°*, while using a scale of 0 to 100 the largest possible slope between two points is *89.4°*. It should be noted that the time series is not being directly inputted into the networks, and so normalization of the time series does not result in faster train times - rather, the slope and duration of each trend which is calculated **using** the normalized data is inputted, which was later normalized again to ensure faster train times.

3.2.3 Median Filtering. Median Filtering is a non-linear filtering method which, when applied to signals, can be used to reduce signal noise. This algorithm was implemented to aid the functioning of the piecewise linear segmentation by isolating the general pattern of the data before it is segmented. The method is fairly simple and works by setting each value in the series to the median values of itself and its neighbours within a user defined filter size. Mathematically, a filtered element at a given time x_t' can be expressed as:

$$x_t' = median\{x_{(t-filter)} \ldots x_{(t+filter)}\}$$

A visual demonstration of the effect this has on input data is shown below in fig. 2 using a small subset of the Cape Town Surface Temperature dataset containing 5 years of daily temperature data. Notice the similarity of yearly patterns in the filtered series compared to the unfiltered series:

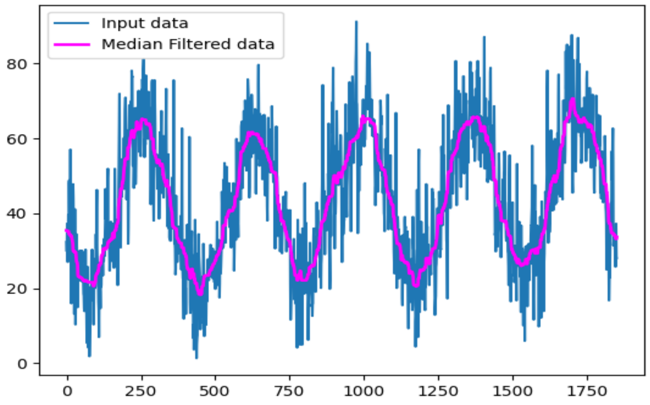


Figure 2: Median Filtering on Cape Town Surface Temperatures

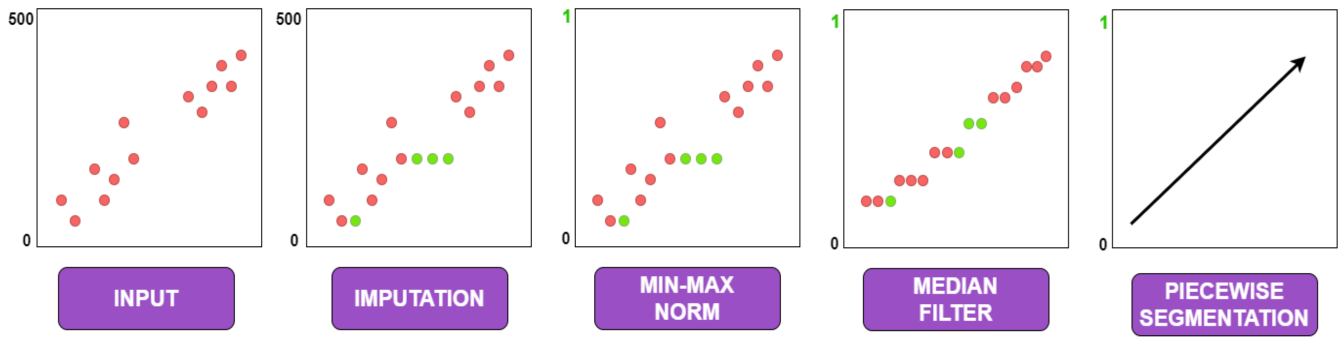


Figure 1: Flow chart of the data preprocessing steps

3.2.4 Piecewise Linear Segmentation: The last step of preprocessing was to segment the data into a sequence of trend lines. This problem is discussed thoroughly in an overview by Lovric et al. 2014 [20] which summarises four main algorithms for segmentation. One method which is discussed is the Sliding Window Algorithm. This method usually achieves a higher approximation loss than algorithms such as Bottom-up, Top-Down and SWAB though it can be implemented in linear time [20]. Due to the large size of datasets, this approach was favoured.

Applying the sliding window algorithm to data which has been median filtered results in an increased loss with respect to the original data, though visualisation of results reveal that the algorithm can better capture the pattern the data follows. Using the same subset shown in fig. 2 we can create two piecewise linear representations of the data to compare the performance with and without median filtering. Both examples shown below in fig. 3 and fig. 4 use the same maximum error parameter, the only difference is that fig. 4 applies the segmentation to median filtered data:

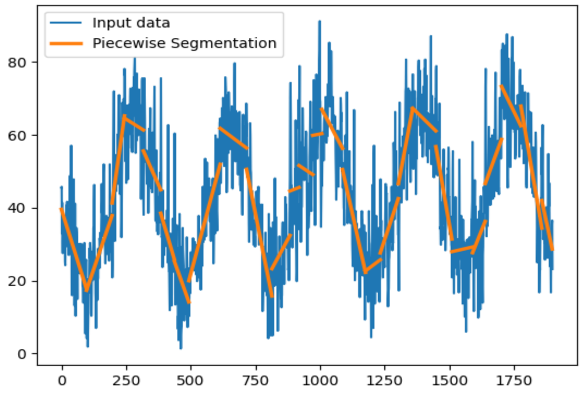


Figure 3: PLS on unfiltered Cape Town Surface Temperatures

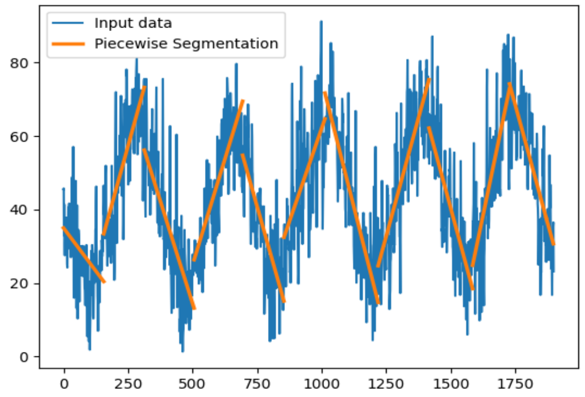


Figure 4: PLS on filtered Cape Town Surface Temperatures

Once data had been segmented, angles were simply normalized to a range of -1 to 1 by dividing each slope by 90°, and trend durations were normalized to a range of 0 to 1 using min-max normalization. This worked to reduce learning times, and prevent learning bias towards either output of the dual prediction approach.

3.2.5 Dual Prediction and Single Prediction Approaches. In the single prediction approach which is proposed, input-output pairs were created using a sequence of trends as inputs, each of which containing a slope and duration value, and using either the slope, or duration of the next trend in the sequence as the output. For the dual output model, the same input sequence was used though the output contained both the slope and duration.

3.3 Hyperparameter Tuning

A total of 1512 configurations were considered during optimisation using the search grid shown in fig. 15, with each DNN's best configuration for a dataset being chosen based on its average validation loss over 3 instances. fig. 13 and fig. 14 show the configurations that were selected. Averages were taken to mitigate the effect random values generated during the initialisation have on performance since these values can cause significant performance gaps between two instances of an identically parameterised model. We refer to a configuration's consistency of results between instances as the robustness of the configuration [12]. In general, configurations tended to perform similarly regardless of hyperparameters.

We aimed to increase robustness by initialising linear layers using the He initialisation technique with normal distribution, fan-in mode, and a ReLU activation function, as done by Jian et al. 2015 [10].

3.4 Performance Evaluation

3.4.1 Regression Metrics. In this study, trend prediction was done using regression models and, for this reason, comparisons between DNNs are made on their regression performance. Two metrics were used for comparison: RMSE (for measuring predictive capacity) and standard deviation of RMSE across instances (for measuring robustness). Both of these metrics were recorded for both slope and duration in each model. RMSE can be calculated in the following way where y_t is the actual output and $\hat{y}_t$ is the predicted one:

$$RMSE = \sqrt{\frac{1}{T}\Sigma_{t=1}^{T}\left(y_t - \hat{y}_t\right)^2}$$

3.4.2 Classification Metrics. In some cases, regression metrics can be difficult to interpret and/or contextualise. A common alternative method for trend prediction is to classify trends as either up-trends or down-trends, and many of the metrics associated with classification such as accuracy, sensitivity, specificity and F1 scores are easier to contextualise.

For this reason, we included a simple transformation to classify predicted trends based on the sign of the predicted slope: if a predicted trend has a positive slope, it is classified as an up trend, otherwise it is classified as a down trend. A 2° tolerance around 0 was included to ensure that very slightly inaccurate sideways predictions are not incorrectly counted as wrong. These metrics are mainly included for interest sake and are not used as a basis for comparison due to the fact that the models have not explicitly been trained to classify. Below is an outline of all metrics used where TP is the number of true positive classifications, TN is true negative, FP is false positive, and FN is false negative.

(1) **Accuracy** is used as a percentage representation of how often a classification is correct. It gives a good idea of general classification performance. It is calculated using:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

(2) **Sensitivity** is a percentage measure of how often the model correctly classifies uptrends

$$Sensitivity = \frac{TP}{TP + FN}$$

(3) **Specificity** is a percentage measure of how often the model correctly classifies downtrends

$$Specificity = \frac{TN}{TN + FP}$$

(4) **F1 score**, lastly, is a composite of sensitivity and specificity that gives a percentage measure of overall performance.

$$F1 = 2 \times \frac{Sensitivity \times Specificity}{Sensitivity + Specificity}$$

These metrics should be able to help put into context whether or not the models are performing effectively or not.

3.4.3 *Evaluation.*

A holdout validation approach was used wherein each dataset was partitioned into a train, validation and test set in the ratio of 70:15:15. These splits were done chronologically, in other words the first 70% of inputs made up the train set, the next 15% made up the validation set and the final 15% made up the test set.

During the training process, each model was run on the validation set at the end of a training epoch. If the validation loss on a particular epoch was the lowest encountered that far, then the model was saved. When training was completed, the model that had the lowest validation loss was used to predict the test set. The RMSE of predictions on the test set were used as the measurement of performance. To obtain the final measurement, this process was repeated 10 times for each configuration and the mean RMSE was recorded, as well as the standard deviation of the RMSE across those runs.

3.5 System Development and Implementation

An important goal in the completion of this experiment was the creation of a reusable testing bench that may aid future studies, as well as students or individuals who would like to familiarise themselves with the experimentation process. The final system used in this study has been well abstracted for ease of use and requires little familiarity with deep learning to run experiments. The main functionality it provides is allowing a user to run trend prediction on a time series using an MLP, CNN, TCN, RNN, LSTM or BiLSTM network with a set of user defined hyperparameters, and output all performance metrics associated with the experiment. The final code base can be accessed at http://github.com/SlackEight/DNN-TRND.

3.5.1 *Implementation.*

The system was implemented in python and jupyter notebook. DNN implementation was done using the pytorch library, with numpy and matplotlib being used for graph generation. Complex functionality of the system was encapsulated in a utils folder containing all scripts that run low level code such as data preprocessing, model training and model testing. This was done to abstract the test bench and make it less overwhelming to individuals with little knowledge of pytorch or deep learning. A Jupyter Notebook script, playground.ipynb , was included which aimed to demonstrate how the test bench works in a low level step by step fashion.

All models are implemented in the models.py script. An out of place dropout was included in all models, and all linear layers were initialised using the He initialisation technique with normal distribution, fan-in mode, and a ReLU activation function, as done by Jian et al. 2015 [10]. Below is a description of how each model tested in this experiment was implemented:

(1) **MLP:** input layer -> 3 fully connected layers -> output layer passed through a tanh activation function.
(2) **CNN:** input later -> 1d convolutional layer with kernel size 2 and ReLU -> 2 fully connected layers -> output layer.
(3) **TCN:** implemented exactly as is from the paper by Bai et al. [17] at https://github.com/locuslab/TCN

3.5.2 *Ease of use.*

The system requires very little familiarity with deep learning to conduct experiments, and all code is clearly and thoroughly explained through comments. Unnecessary complexity was avoided at all times, and the system was also designed with future development in mind. Below is a list of some key usability aspects included:

- Function call side effects were minimized to ensure that updates to certain methods don't require alterations to other code in the base
- New models can be added by simply adding them to the models.py file - no extra work is needed beyond that.
- Running a custom experiment can be done with minimal effort - just set the hyperparameters and pick a model in test_model.py then run the script (see fig. 16).
- A grid_search.py script was included which can be used to optimise for any dataset, and includes progress bars.

Future studies may opt to make improvements to the test bench by implementing a better suited validation method such as the walk-forward validation procedure, with the successive and overlapping training-validation-test partition - used by Chen and Luo 2013 [21], or by implementing a different hyperparameter search method - such as a halving grid search or a Bayes grid search - which may lead to better hyperparameter optimisation. Additions such as these may affect the performance hierarchy and uncover more information about the optimal functioning of these models in the problem of trend prediction - and us such is highly encouraged.

4 RESULTS AND DISCUSSION

4.1 Experiment 1: Dual prediction approach

In this experiment, each architecture was used to predict the next trend's slope and duration simultaneously using two output nodes. This is the approach taken by Kouassi & Moodley [13]. The table below shows the results of testing each architecture using a dual prediction and the methodology presented previously:

Table 1: Comparison of RMSE results for dual prediction approach

Cape Town Air Temperature			
Model	Slope	Duration	Average RMSE
MLP	**0.070 ± 0.005**	0.193 ± 0.005	0.132 ± 0.05
CNN	0.079 ± 0.012	**0.170 ± 0.007**	0.125 ± 0.01
TCN	0.072 ± 0.006	0.173 ± 0.006	**0.123 ± 0.006**

S&P 500 Daily Closing Prices			
Model	Slope	Duration	Average RMSE
MLP	0.585 ± 0.005	0.122 ± 0.004	0.354 ± 0.005
CNN	**0.575 ± 0.008**	**0.105 ± 0.013**	**0.340 ± 0.011**
TCN	0.577 ± 0.005	0.109 ± 0.006	0.343 ± 0.006

Household voltages			
Model	Slope	Duration	Average RMSE
MLP	0.294 ± 0.004	0.189 ± 0.002	0.242 ± 0.003
CNN	**0.293 ± 0.005**	0.187 ± 0.003	**0.240 ± 0.004**
TCN	0.296 ± 0.004	**0.185 ± 0.002**	0.241 ± 0.003

Using the dual prediction approach with the datasets tested, the CNN is the most performant model overall achieving the highest average RMSE on 2 out of 3 of the datasets, and losing only very slightly to the TCN in the first dataset. CNN duration predictions outperformed both other DNNs and overall slope performance was similar to the TCN. The CNN made one poor slope prediction in the first dataset - which contains fewer data points than the other datasets - which was likely due to the tested configuration favouring the reduction of duration RMSE during the training process - a problem that can be seen in the opposite direction with the MLP in the first dataset (favouring reduction of slope RMSE over duration). The CNN was, however, the least robust DNN in the test.

The TCN is a very close second, achieving the best average RMSE for the temperature dataset and very close seconds for both other datasets, while also being more robust. An interesting thing to note about the TCN performance is that despite performing very well on average throughout the experiment, it is usually not the best performing DNN for either slope or duration. This suggests that unlike the other two models which both showed a bias towards either slope or duration predictions at some point, the TCN performed well in both areas in all tests and was almost never the worst choice for any test.

Lastly, the MLP achieves the worst average performance. It often showed a bias towards slope predictions, though it was still the worst on average for this component despite this bias. Duration predictions were far worse than the TCN and CNN, making the MLP the least performant model overall. The performance hierarchy for this experiment is
CNN -> TCN -> MLP.

4.2 Experiment 2: Single prediction model

The second experiment involved using two separate models to independently predict slope and duration of the next trend (meaning each model had a single output node, and was trained only to predict one trend part). This approach allows for 'splicing' of models, where one architecture can be used to predict slope while a different architecture is used to predict duration. Additionally, different hyperparameter configurations could be used for the same architecture when predicting different parts of the trend. This means that each architecture will no longer need a configuration that offers a balance between slope performance and duration performance, and can rather be optimised for a single prediction. The table below shows the results of testing each architecture using a single prediction model with the methodology presented in section 3:

Table 2: Comparison of RMSE results for dual prediction approach

Cape Town Air Temperature			
Model	Slope	Duration	Average RMSE
MLP	0.066 ± 0.007	0.163 ± 0.007	0.115 ± 0.007
CNN	0.062 ± 0.011	0.168 ± 0.003	0.115 ± 0.007
TCN	**0.060 ± 0.006**	**0.162 ± 0.006**	**0.110 ± 0.006**

S&P 500 Daily Closing Prices			
Model	Slope	Duration	Average RMSE
MLP	0.581 ± 0.003	0.107 ± 0.007	0.344 ± 0.005
CNN	0.573 ± 0.005	0.101 ± 0.008	0.337 ± 0.007
TCN	**0.570 ± 0.003**	**0.100 ± 0.006**	**0.335 ± 0.005**

Household voltages			
Model	Slope	Duration	Average RMSE
MLP	0.294 ± 0.002	0.187 ± 0.002	0.241 ± 0.002
CNN	0.292 ± 0.003	**0.185 ± 0.004**	0.239 ± 0.004
TCN	**0.290 ± 0.003**	**0.185 ± 0.002**	**0.238 ± 0.003**

With this approach, there is a reduction in the average RMSE across the datasets for all predictions. The TCN sees a large improvement going from seldom being the most performant model for slope or duration to achieving the best slope, duration *and* average for every dataset. This suggests the TCN truly shines in the single prediction approach, and for the most accurate predictions two single prediction TCNs should be used. Overall the TCN had a 3% improvement when using a single prediction approach over the dual prediction approach (percentage improvement here refers to the percentage reduction in average RMSE and is calculated by simply dividing the average single prediction performance by the average dual prediction performance and subtracting 1).

The CNN performs second best in this experiment, losing the lead it had over the TCN. The CNN has an overall smaller improvement of 2% when moving from the dual prediction approach to the single prediction approach. Despite this, it still has excellent performance and does not lose to the TCN by a large margin.

The MLP still performs the worst of the three though, after a 4% improvement when compared with its dual prediction performance, it is now much more in line with the performance of the CNN and TCN. The performance heirarchy in this experiment is:
TCN -> CNN -> MLP.

4.3 Trend Classification Metrics

While RMSE is the most fair metric for comparison, it does not provide a good way to understand whether or not the DNNs performed well in general, due to the fact that it lacks the context needed for understanding what a certain RMSE value actually indicates about how well a DNN has made a prediction. This is an important consideration when making the decision to use a DNN for trend prediction, since if they perform poorly in general then an alternative method might be better.

For this reason, the results of the regression predictions were also classified as positive (up trend) or negative (down trend) to gain more insight into the directional accuracy of the regression predictions. In short these metrics, which are solely for slope, are measures of how well each DNN predicted whether the next trend will be an up trend or a down trend (with a 2 degree tolerance around 0 for trends that are accurately predicted to be flat). This allows us better insight into how well these architectures perform in general. Below are two tables showing the classification metrics associated with each experiment:

Table 3: Comparison of classification metrics with a dual prediction approach (Experiment 1)

Cape Town Air Temperature				
Model	Sensitivity	Specificity	F1 Score	Accuracy
MLP	**86.3%**	**88.1%**	**87.2%**	**87.3%**
CNN	77.4%	80.7%	79.0%	79.1%
TCN	73.1%	86.2%	79.1%	80.0%

S&P 500 Daily Closing Prices				
Model	Sensitivity	Specificity	F1 Score	Accuracy
MLP	88.8%	**75.2%**	**81.4%**	**85.5%**
CNN	**89.2%**	71.0%	79.1%	84.6%
TCN	88.9%	72.5%	79.9%	84.8%

Household voltages				
Model	Sensitivity	Specificity	F1 Score	Accuracy
MLP	70.6%	65.9%	68.2%	68.3%
CNN	**71.4%**	66.7%	**69.0%**	**69.0%**
TCN	61.1%	**76.6%**	68.0%	68.9%

Table 4: Comparison of classification metrics with a single prediction approach (Experiment 2)

Cape Town Air Temperature				
Model	Sensitivity	Specificity	F1 Score	Accuracy
MLP	88.0%	84.0%	86.0%	86.0%
CNN	84.0%	86.0%	84.9%	85.0%
TCN	**92.0%**	**88.0%**	**90.0%**	**90.0%**

S&P 500 Daily Closing Prices				
Model	Sensitivity	Specificity	F1 Score	Accuracy
MLP	88.7%	**71.8%**	79.4%	84.4%
CNN	89.1%	71.5%	79.4%	84.6%
TCN	**89.6%**	**71.8%**	**79.7%**	**85.0%**

Household voltages				
Model	Sensitivity	Specificity	F1 Score	Accuracy
MLP	63.7%	**69.4%**	66.4%	66.7%
CNN	**66.7%**	64.4%	65.5%	65.5%
TCN	65.7%	69.1%	**67.3%**	**67.5%**

4.3.1 ***Disclaimer:*** It should be noted that these DNNs were not explicitly trained to classify trends. Rather, they have been trained to predict the exact slope of the next trend using regression (a more specific prediction than just up or down). The classification metrics here are therefore obtained by classifying positive predictions as up trends or negative predictions as down trends, which could be considered a lossy conversion as the exact values of the prediction are discarded and only the sign is used. If, during training, a more suitable activation and loss function had been used instead, and the models had been trained only on classification inputs, then these results may have been different. As such, these metrics should not be used as a measure each DNNs trend classification capacity - but rather, as an additional insight into the directional behaviour of the regression performance shown in experiment 1 and 2, as well as into the overall predictive capacity of the models. For this reason, these results have not been considered to be a separate experiment but rather an elaboration of results for the two experiments done.

4.3.2 Discussion. All models perform very well, consistently achieving high F1 and Accuracy scores, indicating that the regression performance in the experiments was strong. The first dataset, shown in fig. 10, followed the most predictable pattern - usually containing a summer up trend followed by a winter downtrend, though there are nicks where smaller mid season trends occur. This explains the high but imperfect accuracy that DNNs have on this dataset. The single prediction TCN achieves the best accuracy and F1 score for this dataset, which is in line with the high regression accuracy it achieved (shown in table 2). The dual prediction MLP also has very good performance on this dataset. In the single prediction models, there is no strong bias towards sensitivity or specificity - though the dual prediction CNN and, especially, TCN show a greater bias towards specificity, leading to lower F1 scores.

Perhaps the most notable result for analysis is the S&P500 performance. It is worth considering that due to the stable growth of the S&P 500, median filtering with a window size of 10 elements and segmenting with a max error of 10, as was done in this study, results in 74.51% of trends being up trends (as can be seen in fig. 11). This means that predicting a positive trend for every prediction would result in a 74.51% directional accuracy, 100% sensitivity, and 0% specificity. For this reason all DNNs, being more likely to predict a trend as upwards, demonstrate high sensitivity. This means they are very likely to correctly predict up trends - though many times a highly sensitive model sacrifices specificity to achieve its accurate classification of positive results. This can be seen in this dataset, as every model has a higher sensitivity than specificity. Despite this, every model still achieves between a 71% and 76% specificty - meaning they can accuractely predict down trends too, and all DNNs strongly outperform the "always guess up" accuracy - achieving an average of 84.8% across both approaches.

Finally, DNNs perform worse on the house voltage dataset (shown in fig. 12) than in the other datasets, though this dataset is far more volatile. Despite not having the lowest RMSE for the dataset, the dual prediction CNN achieves the best classification performance with an F1 and accuracy of 69%. Due to the dataset not having more uptrends than downtrends, models tend to have similar sensitivity to specificity, and predict both trend directions with similar accuracy.

4.3.3 Differences between single and dual prediction approaches. The classification results do not show the same hierarchy found when comparing models using RMSE, and upon further inspection they do not appear to be very closely correlated. A good example of this is the dual prediction MLP in the S&P500 dataset, which had the worst RMSE of all models but had the best classification results. In the first dataset single prediction models strongly outperformed dual prediction models - though in both other datasets a dual prediction model obtained the best performance. When using a single prediction approach, TCN predictions were still the most directionally accurate, though in many cases a dual prediction appraoch offered better accuracy. Once again, however, these metrics should not be used as an indication of each model's trend classification performance but rather as evidence that each DNNs regression performance is strong.

4.4 Choosing a model

Despite the performance hierarchy, all models still perform very similarly in general - and any improvement in performance between different models are slight. This is demonstrated below in fig. 5 which compares the average RMSE of all predictions made in each experiment. Note that RMSE is similar across all models tested.

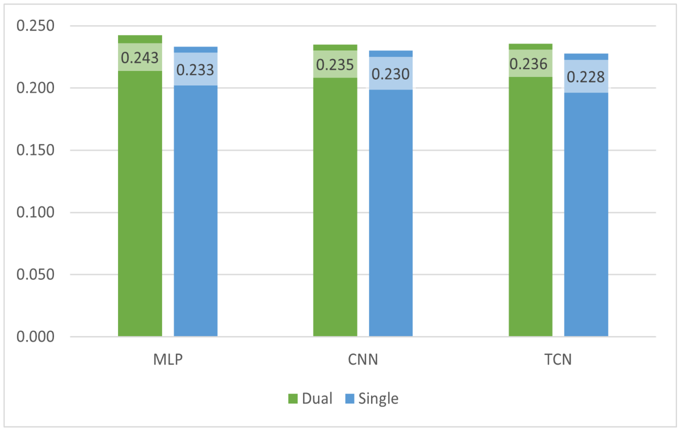


Figure 5: Average RMSE of all predictions made by each model

Additionally, there are trade offs that should be considered when implementing more complex solutions - to name some:

- A single prediction approach takes twice as long to perform hyperparameter grid searches.
- A single prediction approach has a more complicated implementation due to the need to combine predictions.
- Simpler models are usually easier to implement.
- Models with less hyperparameters require less time spent optimising.
- Some models train faster than others

The last two points are particularly important when large grid optimisations are being done and completion time is a factor. In cases like this, it is especially important to consider factors that will dictate how long the optimisation process will take. Convolutional networks like the CNN and TCN intrinsically have more hyperparameters than MLPs due to hyperparameters in convolution layers like kernel size, and as such will often require more time spent optimising to find the best configuration - though it should also be considered that Lara-Benítez et al found that CNN performance tended to be less contingent on having an optimal configuration [16]. Another factor to consider is the complexity of a model which will dictate the speed at which it trains and predicts. To gain insight into the second factor, the time taken to optimise each model was recorded, and the average time to train and test 100 models (specific to the datasets used in this experiment) was subsequently calculated. The figure below shows the results of this calculation:

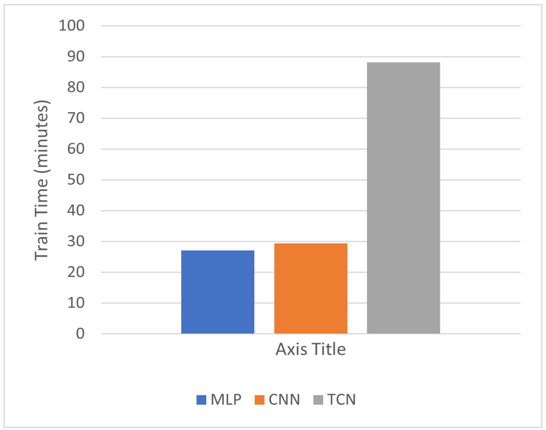


Figure 6: Average time to search through 100 configurations using each architecture

As is evident, the TCN takes much longer to configure due to the slower training and set up times. By contrast, the CNN is similar in speed to the MLP. In the tests done in this study, the TCN took 3.0x longer per model than the CNN and 3.3x longer per model than the MLP. The time sensitivity of a project in which a model is being applied should therefore be considered when choosing models. CNNs offer a good middle ground of speed, ease to configure and performance - though TCNs may offer the best regression performance.

4.5 Summary

In summary the main points of note are:

- When using a dual prediction approach, a CNN will likely lead to the best performance.
- When using a single prediction approach, a TCN will likely lead to the best performance.
- Regardless of approach, the MLP had the worst regression performance.
- RMSE was improved in every model when using a single prediction approach instead of a dual prediction approach.
- Associated classification metrics were not improved when moving from a dual prediction approach to a single prediction approach.
- TCNs took over 3x longer than the CNN and MLP to train.
- All models performed well, and improvements between any two architectures/approaches are slight.

5 CONCLUSIONS

In this study, the performance of MLPs, CNNs and TCNs was compared in the problem of time series trend prediction when using both the traditional dual prediction approach used by Kouassi & Moodley [13] and when using two DNNs with one output node each to independently predict trend slope and duration, which we referred to as a single prediction model.

In the first experiment, which used a dual prediction approach, CNNs were found to be the most performant architecture, slightly outperforming the TCN by an average of 0.3% and outperforming the MLP by an average of 3.2%. In the second experiment, the single prediction model was tested. All models had improved performance when using a single prediction approach. With this approach, the TCN was the most performant model in the entire experiment, outperforming the single prediction CNN by 1.0% and the single prediction MLP by 2.4%.

All architectures performed well in the task, as can be seen by the high predictive performance in the classification metrics, suggesting DNNs are an effective method of trend prediction when employing the methodology outlined in this study. Every DNN tested was able to outperform the S&P500 on the test set, and make accurate predictions with respect to weather changes and household voltages. In time sensitive use cases, it is recommended to use a dual prediction CNN as it has a simple implementation, trains very quickly, and performs very well even compared to the best methods found in this comparison.

The main improvements that can be made on this study would be to use the walk-forward evaluation procedure, with the successive and overlapping training-validation-test partition used by Luo, L. & Chen, X. [22] for more accurate prediction of the models' ability to generalize. Additionally, due to equipment and time constraints, large optimisation grids could not be carried out while using large datasets that contain many trends. For this reason, future studies may obtain clearer results by performing more rigorous testing with larger datasets containing more trend lines.

For the datasets tested, the research questions were answered as 1) When using a single prediction model, TCNs offer a slight improvements over other models and 2) Two single predictions **do** offer better predictive capability than a dual model.

REFERENCES

[1] Nesreen K. Ahmed, Amir F. Atiya, Neamat El Gayar, and Hisham El-Shishiny. 2010. An Empirical Comparison of Machine Learning Models for Time Series Forecasting. *Econometric Reviews* 29, 5-6 (2010), 594–621. https://doi.org/10.1080/07474938.2010.481556 arXiv:https://doi.org/10.1080/07474938.2010.481556

[2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. (2018). arXiv:cs.LG/1803.01271 https://arxiv.org/pdf/1803.01271.pdf

[3] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. 2018. Dilated convolutional neural networks for time series forecasting. *Journal of Computational Finance, Forthcoming* (2018).

[4] Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. 2020. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing* 399 (2020), 491–501. https://doi.org/10.1016/j.neucom.2020.03.011

[5] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, 933–941. https://proceedings.mlr.press/v70/dauphin17a.html

[6] Shumin Deng, Ningyu Zhang, Wen Zhang, Jiaoyan Chen, Jeff Z. Pan, and Huajun Chen. 2019. Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network. In *Companion Proceedings of The 2019 World Wide Web Conference (WWW '19)*. Association for Computing Machinery, New York, NY, USA, 678–685. https://doi.org/10.1145/3308560.3317701

[7] W.A. Fuller. 2009. *Introduction to Statistical Time Series*. Wiley. https://books.google.co.za/books?id=tI6j47m4tVwC

[8] M.W Gardner and S.R Dorling. 1998. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment* 32, 14 (1998), 2627–2636. https://doi.org/10.1016/S1352-2310(97)00447-0

[9] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. 2017. A Convolutional Encoder Model for Neural Machine Translation. (2017). arXiv:cs.CL/1611.02344

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR* abs/1502.01852 (2015). arXiv:1502.01852 http://arxiv.org/abs/1502.01852

[11] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2017. Neural Machine Translation in Linear Time. (2017). arXiv:cs.CL/1610.10099

[12] Kouame Hermann Kouassi and Deshendran Moodley. 2020. An Analysis of Deep Neural Networks for Predicting Trends in Time Series Data. In *Artificial Intelligence Research*, Aurona Gerber (Ed.). Springer International Publishing, Cham, 119–140.

[13] Kouame Hermann Kouassi and Deshendran Moodley. 2020. Automatic deep learning for trend prediction in time series data. (2020). arXiv:cs.LG/2009.08510

[14] Ping-Huan Kuo and Chiou-Jye Huang. 2018. A High Precision Artificial Neural Networks Model for Short-Term Energy Load Forecasting. *Energies* 11, 1 (2018). https://doi.org/10.3390/en11010213

[15] Pedro Lara-Benítez, Manuel Carranza-García, José M. Luna-Romera, and José C. Riquelme. 2020. Temporal Convolutional Networks Applied to Energy-Related Time Series Forecasting. *Applied Sciences* 10, 7 (2020). https://doi.org/10.3390/app10072322

[16] Pedro Lara-Benítez, Manuel Carranza-García, and José C. Riquelme. 2021. An Experimental Review on Deep Learning Architectures for Time Series Forecasting. *International Journal of Neural Systems* 31, 03 (Feb 2021), 2130001. https://doi.org/10.1142/s0129065721300011

[17] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. 2016. Temporal Convolutional Networks: A Unified Approach to Action Segmentation. In *Computer Vision – ECCV 2016 Workshops*, Gang Hua and Hervé Jégou (Eds.). Springer International Publishing, Cham, 47–54.

[18] Wei Li and Jian Liao. 2017. A comparative study on trend forecasting approach for stock price time series. In *2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. IEEE, 74–78.

[19] Miodrag Lovrić, Marina Milanović, and Milan Stamenković. 2014. Algoritmic methods for segmentation of time series: An overview. *Journal of Contemporary Economic and Business Issues* 1, 1 (2014), 31–53.

[20] Miodrag Lovrić, Marina Milanović, and Milan Stamenković. 2014. Algoritmic methods for segmentation of time series: An overview. *Journal of Contemporary Economic and Business Issues* 1, 1 (2014), 31–53.

[21] Linkai Luo and Xi Chen. 2013. Integrating piecewise linear representation and weighted support vector machine for stock trading signal prediction. *Applied Soft Computing* 13, 2 (2013), 806–816. https://doi.org/10.1016/j.asoc.2012.10.026

[22] Linkai Luo and Xi Chen. 2013. Integrating piecewise linear representation and weighted support vector machine for stock trading signal prediction. *Applied Soft Computing* 13, 2 (2013), 806–816. https://doi.org/10.1016/j.asoc.2012.10.026

[23] Robert J Schalkoff. 2007. Pattern recognition. *Wiley Encyclopedia of Computer Science and Engineering* (2007).

[24] José Torres, Antonio Galicia de Castro, Alicia Troncoso, and Francisco Martínez-Álvarez. 2018. A scalable approach based on deep learning for big data time series forecasting. *Integrated Computer-Aided Engineering* 25 (08 2018), 1–14. https://doi.org/10.3233/ICA-180580

[25] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. 2017. Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks. In *2017 IEEE 19th Conference on Business Informatics (CBI)*. https://doi.org/10.1109/CBI.2017.23

[26] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. (2016). arXiv:cs.SD/1609.03499

[27] Renzhuo Wan, Shuping Mei, Jun Wang, Min Liu, and Fan Yang. 2019. Multivariate Temporal Convolutional Network: A Deep Neural Networks Approach for Multivariate Time Series Forecasting. *Electronics* 8, 8 (Aug 2019), 876. https://doi.org/10.3390/electronics8080876

APPENDIX

.

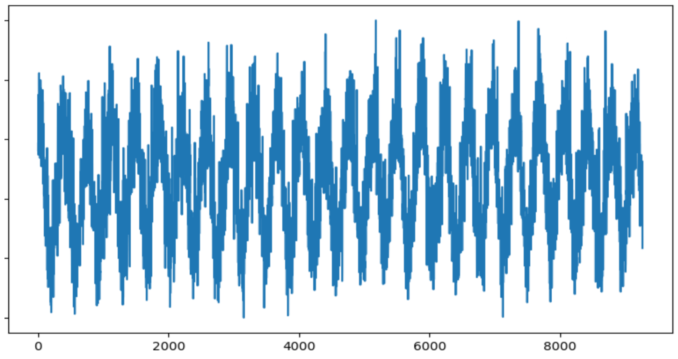


Figure 7: Cape Town Surface Temperature Dataset

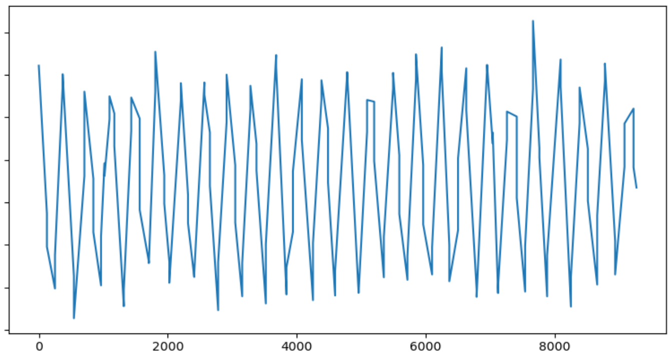


Figure 10: Cape Town Surface Temperature Dataset with PLS

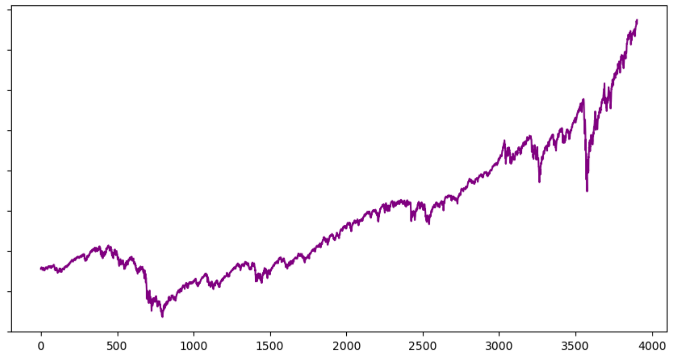


Figure 8: S&P500 Daily Closing Prices Dataset

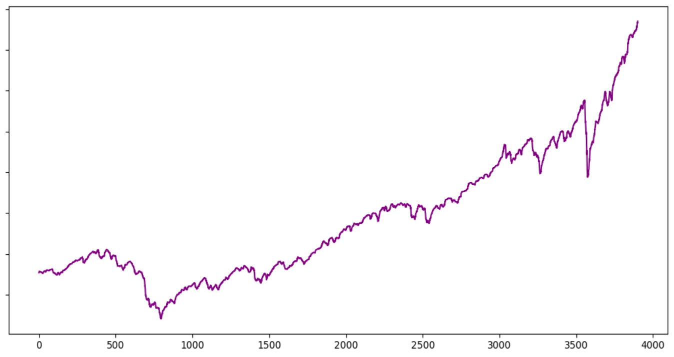


Figure 11: S&P500 Daily Closing Prices Dataset with PLS

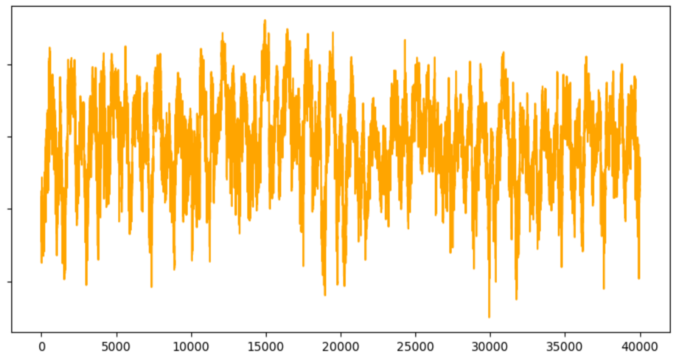


Figure 9: Household Voltage Dataset

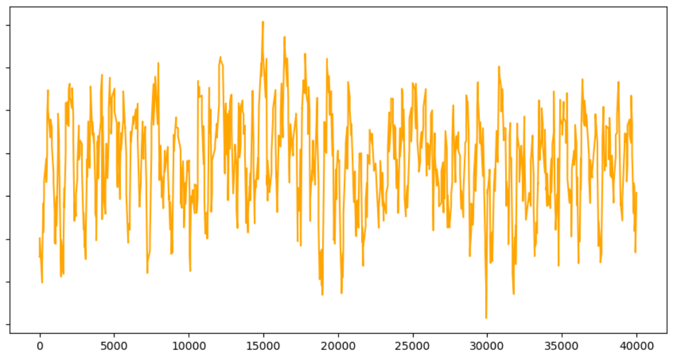
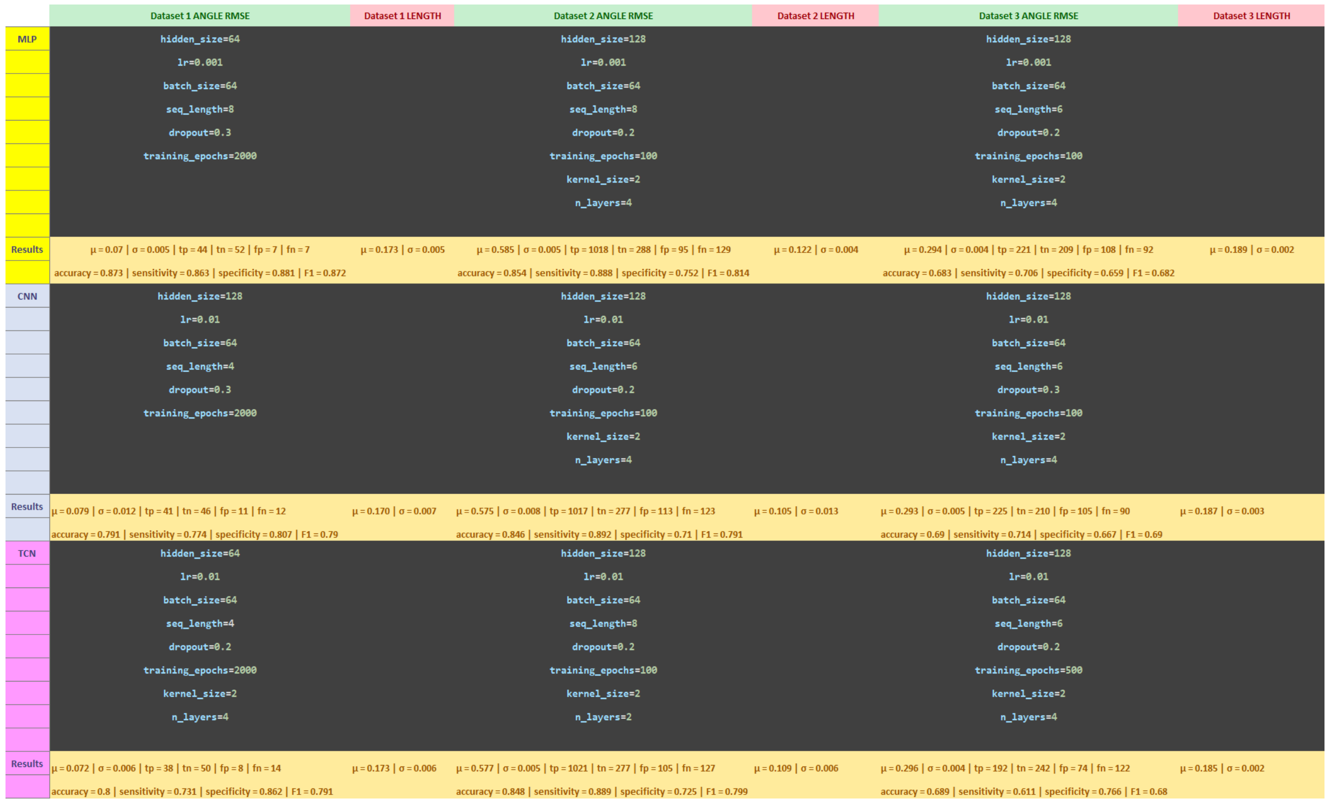
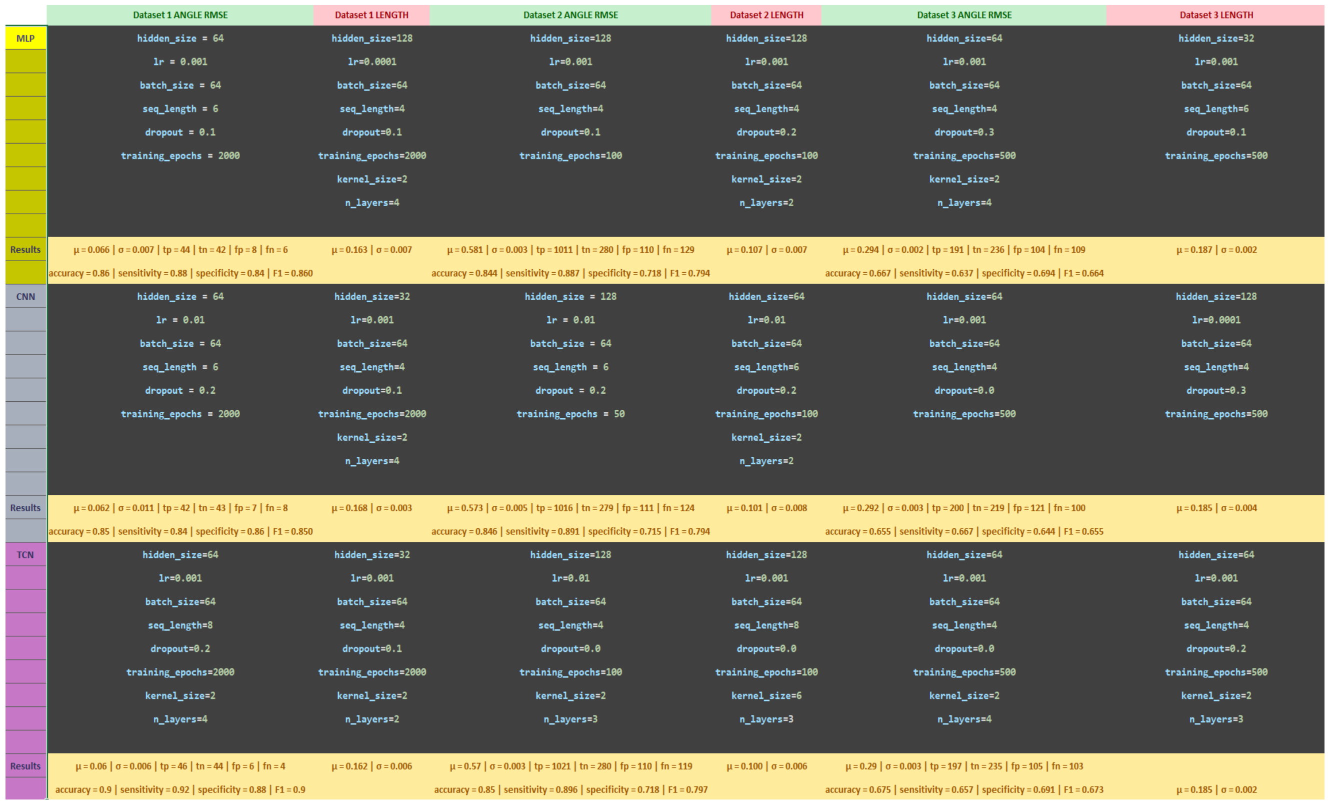


Figure 12: Household Voltage Dataset with PLS



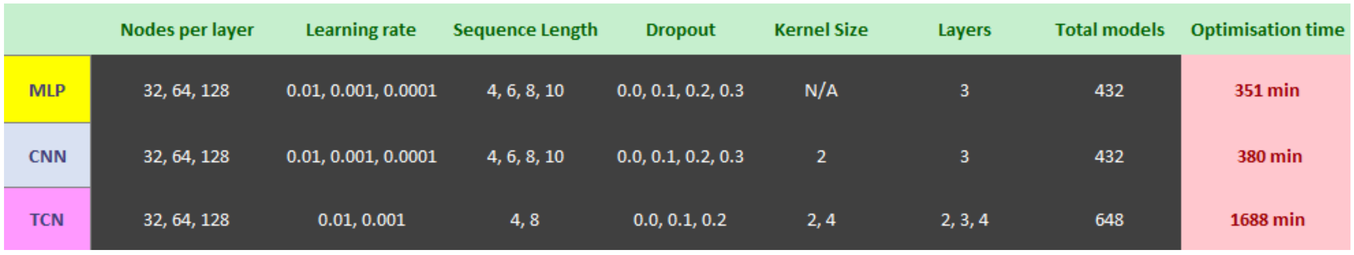
	Dataset 1 ANGLE RMSE	Dataset 1 LENGTH	Dataset 2 ANGLE RMSE	Dataset 2 LENGTH	Dataset 3 ANGLE RMSE	Dataset 3 LENGTH
MLP	hidden_size=64 lr=0.001 batch_size=64 seq_length=8 dropout=0.3 training_epochs=2000		hidden_size=128 lr=0.001 batch_size=64 seq_length=8 dropout=0.2 training_epochs=100 kernel_size=2 n_layers=4		hidden_size=128 lr=0.001 batch_size=64 seq_length=6 dropout=0.2 training_epochs=100 kernel_size=2 n_layers=4	
Results	μ = 0.07 | σ = 0.005 | tp = 44 | tn = 52 | fp = 7 | fn = 7 accuracy = 0.873 | sensitivity = 0.863 | specificity = 0.881 | F1 = 0.872	μ = 0.173 | σ = 0.005	μ = 0.585 | σ = 0.005 | tp = 1018 | tn = 288 | fp = 95 | fn = 129 accuracy = 0.854 | sensitivity = 0.888 | specificity = 0.752 | F1 = 0.814	μ = 0.122 | σ = 0.004	μ = 0.294 | σ = 0.004 | tp = 221 | tn = 209 | fp = 108 | fn = 92 accuracy = 0.683 | sensitivity = 0.706 | specificity = 0.659 | F1 = 0.682	μ = 0.189 | σ = 0.002
CNN	hidden_size=128 lr=0.01 batch_size=64 seq_length=4 dropout=0.3 training_epochs=2000		hidden_size=128 lr=0.01 batch_size=64 seq_length=6 dropout=0.2 training_epochs=100 kernel_size=2 n_layers=4		hidden_size=128 lr=0.01 batch_size=64 seq_length=6 dropout=0.3 training_epochs=100 kernel_size=2 n_layers=4	
Results	μ = 0.079 | σ = 0.012 | tp = 41 | tn = 46 | fp = 11 | fn = 12 accuracy = 0.791 | sensitivity = 0.774 | specificity = 0.807 | F1 = 0.79	μ = 0.170 | σ = 0.007	μ = 0.575 | σ = 0.008 | tp = 1017 | tn = 277 | fp = 113 | fn = 123 accuracy = 0.846 | sensitivity = 0.892 | specificity = 0.71 | F1 = 0.791	μ = 0.105 | σ = 0.013	μ = 0.293 | σ = 0.005 | tp = 225 | tn = 210 | fp = 105 | fn = 90 accuracy = 0.69 | sensitivity = 0.714 | specificity = 0.667 | F1 = 0.69	μ = 0.187 | σ = 0.003
TCN	hidden_size=64 lr=0.01 batch_size=64 seq_length=4 dropout=0.2 training_epochs=2000 kernel_size=2 n_layers=4		hidden_size=128 lr=0.01 batch_size=64 seq_length=8 dropout=0.2 training_epochs=100 kernel_size=2 n_layers=2		hidden_size=128 lr=0.01 batch_size=64 seq_length=6 dropout=0.2 training_epochs=500 kernel_size=2 n_layers=4	
Results	μ = 0.072 | σ = 0.006 | tp = 38 | tn = 50 | fp = 8 | fn = 14 accuracy = 0.8 | sensitivity = 0.731 | specificity = 0.862 | F1 = 0.791	μ = 0.173 | σ = 0.006	μ = 0.577 | σ = 0.005 | tp = 1021 | tn = 277 | fp = 105 | fn = 127 accuracy = 0.848 | sensitivity = 0.889 | specificity = 0.725 | F1 = 0.799	μ = 0.109 | σ = 0.006	μ = 0.296 | σ = 0.004 | tp = 192 | tn = 242 | fp = 74 | fn = 122 accuracy = 0.689 | sensitivity = 0.611 | specificity = 0.766 | F1 = 0.68	μ = 0.185 | σ = 0.002

Figure 13: Hyperparameters used for each dataset in the dual prediction appraoch



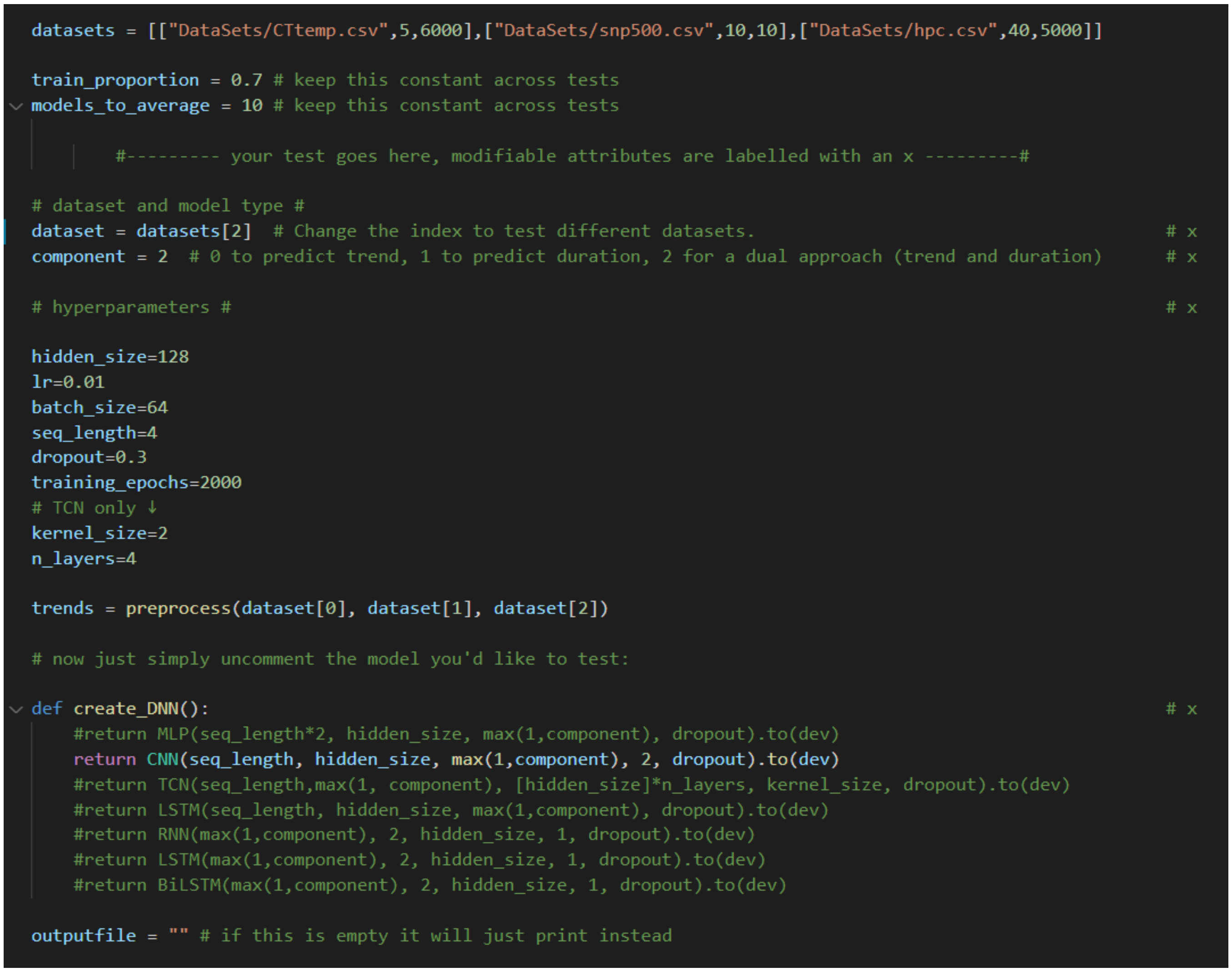
	Dataset 1 ANGLE RMSE	Dataset 1 LENGTH	Dataset 2 ANGLE RMSE	Dataset 2 LENGTH	Dataset 3 ANGLE RMSE	Dataset 3 LENGTH
MLP	hidden_size = 64 lr = 0.001 batch_size = 64 seq_length = 6 dropout = 0.1 training_epochs = 2000	hidden_size=128 lr=0.0001 batch_size=64 seq_length=4 dropout=0.1 training_epochs=2000 kernel_size=2 n_layers=4	hidden_size=128 lr=0.001 batch_size=64 seq_length=4 dropout=0.1 training_epochs=100	hidden_size=128 lr=0.001 batch_size=64 seq_length=4 dropout=0.2 training_epochs=100 kernel_size=2 n_layers=2	hidden_size=64 lr=0.001 batch_size=64 seq_length=4 dropout=0.3 training_epochs=500 kernel_size=2 n_layers=4	hidden_size=32 lr=0.001 batch_size=64 seq_length=6 dropout=0.1 training_epochs=500
Results	μ = 0.066 | σ = 0.007 | tp = 44 | tn = 42 | fp = 8 | fn = 6 accuracy = 0.86 | sensitivity = 0.88 | specificity = 0.84 | F1 = 0.860	μ = 0.163 | σ = 0.007	μ = 0.581 | σ = 0.003 | tp = 1011 | tn = 280 | fp = 110 | fn = 129 accuracy = 0.844 | sensitivity = 0.887 | specificity = 0.718 | F1 = 0.794	μ = 0.107 | σ = 0.007	μ = 0.294 | σ = 0.002 | tp = 191 | tn = 236 | fp = 104 | fn = 109 accuracy = 0.667 | sensitivity = 0.637 | specificity = 0.694 | F1 = 0.664	μ = 0.187 | σ = 0.002
CNN	hidden_size = 64 lr = 0.01 batch_size = 64 seq_length = 6 dropout = 0.2 training_epochs = 2000	hidden_size=32 lr=0.001 batch_size=64 seq_length=4 dropout=0.1 training_epochs=2000 kernel_size=2 n_layers=4	hidden_size = 128 lr = 0.01 batch_size = 64 seq_length = 6 dropout = 0.2 training_epochs = 50	hidden_size=64 lr=0.01 batch_size=64 seq_length=6 dropout=0.2 training_epochs=100 kernel_size=2 n_layers=2	hidden_size=64 lr=0.001 batch_size=64 seq_length=4 dropout=0.0 training_epochs=500	hidden_size=128 lr=0.0001 batch_size=64 seq_length=4 dropout=0.3 training_epochs=500
Results	μ = 0.062 | σ = 0.011 | tp = 42 | tn = 43 | fp = 7 | fn = 8 accuracy = 0.85 | sensitivity = 0.84 | specificity = 0.86 | F1 = 0.850	μ = 0.168 | σ = 0.003	μ = 0.573 | σ = 0.005 | tp = 1016 | tn = 279 | fp = 111 | fn = 124 accuracy = 0.846 | sensitivity = 0.891 | specificity = 0.715 | F1 = 0.794	μ = 0.101 | σ = 0.008	μ = 0.292 | σ = 0.003 | tp = 200 | tn = 219 | fp = 121 | fn = 100 accuracy = 0.655 | sensitivity = 0.667 | specificity = 0.644 | F1 = 0.655	μ = 0.185 | σ = 0.004
TCN	hidden_size=64 lr=0.001 batch_size=64 seq_length=8 dropout=0.2 training_epochs=2000 kernel_size=2 n_layers=4	hidden_size=32 lr=0.001 batch_size=64 seq_length=4 dropout=0.1 training_epochs=2000 kernel_size=2 n_layers=2	hidden_size=128 lr=0.01 batch_size=64 seq_length=4 dropout=0.0 training_epochs=100 kernel_size=2 n_layers=3	hidden_size=128 lr=0.001 batch_size=64 seq_length=8 dropout=0.0 training_epochs=100 kernel_size=6 n_layers=3	hidden_size=64 lr=0.001 batch_size=64 seq_length=4 dropout=0.0 training_epochs=500 kernel_size=2 n_layers=4	hidden_size=64 lr=0.001 batch_size=64 seq_length=4 dropout=0.2 training_epochs=500 kernel_size=2 n_layers=3
Results	μ = 0.06 | σ = 0.006 | tp = 46 | tn = 44 | fp = 6 | fn = 4 accuracy = 0.9 | sensitivity = 0.92 | specificity = 0.88 | F1 = 0.9	μ = 0.162 | σ = 0.006	μ = 0.57 | σ = 0.003 | tp = 1021 | tn = 280 | fp = 110 | fn = 119 accuracy = 0.85 | sensitivity = 0.896 | specificity = 0.718 | F1 = 0.797	μ = 0.100 | σ = 0.006	μ = 0.29 | σ = 0.003 | tp = 197 | tn = 235 | fp = 105 | fn = 103 accuracy = 0.675 | sensitivity = 0.657 | specificity = 0.691 | F1 = 0.673	μ = 0.185 | σ = 0.002

Figure 14: Hyperparameters used for each dataset in the single prediction appraoch



	Nodes per layer	Learning rate	Sequence Length	Dropout	Kernel Size	Layers	Total models	Optimisation time
MLP	32, 64, 128	0.01, 0.001, 0.0001	4, 6, 8, 10	0.0, 0.1, 0.2, 0.3	N/A	3	432	351 min
CNN	32, 64, 128	0.01, 0.001, 0.0001	4, 6, 8, 10	0.0, 0.1, 0.2, 0.3	2	3	432	380 min
TCN	32, 64, 128	0.01, 0.001	4, 8	0.0, 0.1, 0.2	2, 4	2, 3, 4	648	1688 min

Figure 15: All hyperparameter values tested during grid optimisation (done with an i5-8400 and a GTX 1080 Ti)



```
datasets = [["DataSets/CTtemp.csv",5,6000],["DataSets/snp500.csv",10,10],["DataSets/hpc.csv",40,5000]]

train_proportion = 0.7 # keep this constant across tests
models_to_average = 10 # keep this constant across tests

        #--------- your test goes here, modifiable attributes are labelled with an x ---------#

# dataset and model type #
dataset = datasets[2]  # Change the index to test different datasets.                                  # x
component = 2  # 0 to predict trend, 1 to predict duration, 2 for a dual approach (trend and duration)   # x

# hyperparameters #                                                                                     # x

hidden_size=128
lr=0.01
batch_size=64
seq_length=4
dropout=0.3
training_epochs=2000
# TCN only ↓
kernel_size=2
n_layers=4

trends = preprocess(dataset[0], dataset[1], dataset[2])

# now just simply uncomment the model you'd like to test:

def create_DNN():                                                                                       # x
    #return MLP(seq_length*2, hidden_size, max(1,component), dropout).to(dev)
    return CNN(seq_length, hidden_size, max(1,component), 2, dropout).to(dev)
    #return TCN(seq_length,max(1, component), [hidden_size]*n_layers, kernel_size, dropout).to(dev)
    #return LSTM(seq_length, hidden_size, max(1,component), dropout).to(dev)
    #return RNN(max(1,component), 2, hidden_size, 1, dropout).to(dev)
    #return LSTM(max(1,component), 2, hidden_size, 1, dropout).to(dev)
    #return BiLSTM(max(1,component), 2, hidden_size, 1, dropout).to(dev)

outputfile = "" # if this is empty it will just print instead
```

Figure 16: A screenshot of the modifiable section of the test bench. This is all a user needs to modify to run their own experiments.