Deep Learning Classifiers for Power System Time Series Data

Nathan O'Sullivan
Science and Engineering Faculty
Queensland University of Technology
2 George Street, Brisbane, 4001, QLD, Australia

Abstract—**Time Series Classification is a challenging field of machine learning. Recent research in this area has shown promising results from deep neural network-based classifiers. This paper details the application of two deep neural network-based classifiers to time series data which has been generated by power system simulation software. The project demonstrates that the two classifiers tested are well suited to this application.**

Keywords—deep learning; convolutional neural networks; power system simulation; time series; classification

I. Introduction

The aim of this project is to determine whether current state-of-the-art deep learning models can be used to classify time series data generated by power system simulation software. In particular the research aims to demonstrate the feasibility of using deep learning Time Series Classification (TSC) models to classify power system simulation results automatically as part of a real-world workflow.

TSC has been considered a challenging problem in data mining for the last two decades [11]. TSC problems differ from traditional classification problems in that each attribute (or feature) is ordered by time. Many non-deep learning TSC algorithms of varying complexity have been proposed over the years [4]. The success of deep learning models in many data mining applications has also seen them applied to the task of TSC. A recent review of deep learning TSC models [11] found that they performed well compared to the state-of-the-art non-deep learning algorithms when applied to the UCR/UEA time series archive.

Most TSC research uses the UCR/UEA time series archive as the source of time series datasets [8] and there is little research publicly available that applies these algorithms to real world datasets. This project will help clarify whether the theoretical state-of-the-art TSC algorithms can be applied to practical applications.

The paper which follows first briefly outlines the background of the problem to which the classifiers are being applied and reviews the current literature on TSC. The paper then details the methodology used in this project from collecting input datasets through to assessing classifier results. The results are then presented and discussed. The conclusion provides a summary of the paper and suggests avenues for future research.

II. Problem Background

In Australia, Transmission Network Service Providers (TNSP) in the National Electricity Market are required to model network changes and simulate various scenarios that are likely to occur in reality. The aim of these simulations is to make sure that all equipment connected to the electrical network meet the specified performance criteria.

The time series data used in this project are the voltage waveforms that are generated by power system simulation software. The voltage waveforms must meet various criteria such as minimum and maximum values, oscillation damping, and maximum magnitude of oscillation.

Fig. 1 shows two examples of the time series data produced from the simulations. They both show the voltage waveforms at a particular network node. The time series on top passes the performance criteria but the time series on the bottom fails (due to persistent oscillations).

A single simulation scenario could include several hundred time series like these. Each time series needs to be analysed by a power system engineer to confirm conformance to the applicable performance criteria. Simulations like this are becoming more critical and more frequent with the high rate of proposal and commissioning of renewable generation in the electricity network [2]. This means that any ways to increase the efficiency the simulation assessment workflow would be beneficial.

III. Review of Literature

A review of deep learning models for TSC was conducted by [11]. In this review nine models were tested on the UCR/UEA archives of univariate and multivariate time series datasets. The results of these classifiers were compared against each other and also against the state-of-the-art non-deep learning algorithms for TSC published by [4].

Two of the nine deep learning algorithms for TSC performed well across all datasets. These were the Fully Convolutional Neural Network (FCN) and Residual Network (ResNet). These two algorithms were previously identified by [23] as performing well for the task of TSC. They have the advantage of being 'end-to-end' classifiers as opposed to the state-of-the-art non-deep learning classifiers which require heavy preprocessing of the raw data or feature crafting.

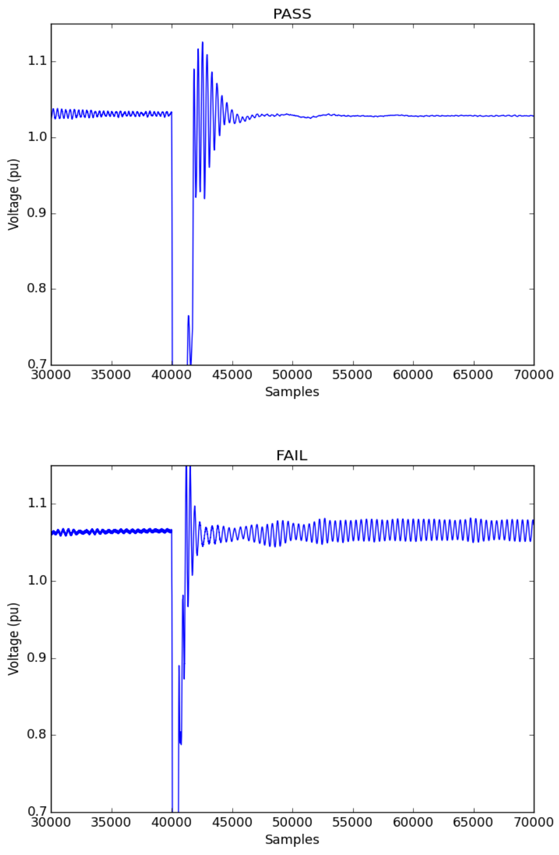

Fig. 1. Examples of PASS and FAIL time series

The UCR/UEA time series archives are used frequently in TSC research. The UCR archive currently contains 128 sets of univariate time series that have been used in over 1,000 research papers [8]. The companion UEA archive contains sets of multivariate time series. Both archives are publicly available for download. There are criticisms in the TSC community about the overuse of these archives and how research may be skewed to recommending classifiers that do not provide good results in real world applications [8].

No studies were found that covered classification of power system simulation data. Data mining research in power systems focused around either forecasting power usage [1] and market prices [15] or classification and clustering of customer types [10]. A variety of deep learning techniques that apply to power systems data are discussed in [24]. However, classification of time series data is not covered.

The literature review did not find any textbooks that specifically covered the topic of deep learning applications for TSC. A general deep learning textbook [7] was chosen for use in this project as it covers the basics of deep learning algorithms through to advanced applications. The textbook uses the `Keras` deep learning library for the Python programming language in a range of applications.

IV. METHODOLOGY

This project used a standard machine learning approach to apply the deep learning TSC models to the collected simulation dataset. This standard approach is the "universal workflow of machine learning" defined in [7]:

A. Defining the problem and assembling a dataset
B. Choosing a measure of success
C. Deciding on an evaluation protocol
D. Preparing your data
E. Developing a model that does better than a baseline
F. Developing a model that overfits
G. Regularizing your model and tuning your hyperparameters

A. *Defining the problem and assembling a dataset*

The problem definition has been covered above, but just to reiterate: The aim of this project is to determine whether current state-of-the-art deep learning models can be used to classify time series data generated by power system simulation software. In particular, voltage waveforms were collected from simulation output files generated by the PSCAD™/EMTDC™ power system simulation software used by the TNSP of Queensland - Powerlink. The following sections describe the process of collecting, labelling and processing the dataset.

1) Collection

The files were stored on Powerlink's corporate network organised in folders based on project, test scenario and geographic region. For each simulation output, the data was stored in a series of formatted text files with ten time series recorded per text file. Each time series was arranged as a column in the text file. In addition to the collection of text files there was an accompanying XML file containing the metadata about each time series.

For each simulation a wide range of data points were recorded, in addition to the voltage waveforms. A Python script[1] was developed to find the output files in the directory structure, identify the voltage waveforms, extract only that time series and save in a CSV file with appropriate metadata stored in the header rows.

2) Labelling

The dataset compiled contained almost 7000 unlabeled time series. The next step was to add labels to this dataset so that it could be used for model training.

Using project summary information and guidance from Powerlink engineers, each time series was labelled. If the time series met the response criteria it was labelled '1' (a pass). Otherwise, it was labelled '0' (a fail). This process was time consuming but sped up by using a Python script to quickly view and assess the waveforms. The project summary information also allowed me to mark certain collections of time series as a pass.

Through the process of labelling the data, a number of corrupted or incompatible waveforms were identified. Removing these 'bad' samples left 6267 time series in the dataset. Of these 4972 were 160,000 samples long, 710 were 120,000 samples long and 585 were 80,000 samples long.

A summary of the labelled dataset is shown in Table I. The datasets are imbalanced, with the number of passes exceeding the number of fails.

[1] All Python code used in this project is available at https://github.com/manworldloves/PSCAD_TSC

TABLE I. DATASET SUMMARY

Record Length (samples)	# of Records	Passes	Fails	% Pass/Fail
80,000	585	525	60	89.7%/10.3%
120,000	710	672	38	94.7%/5.3%
160,000	4972	4199	773	84.5%/15.5%

3) Processing

The labelled dataset consisted of several hundred files each containing Powerlink project information. A Python script was used to consolidate all the time series of the same sample length into a single file. This process also stripped out any identifying information from the dataset and maintained only the minimum amount of metadata required.

At this stage the data was down sampled by a factor of 100. This was done because of concern about the training time for a model with ~100,000 inputs. The datasets used by [11] from the UCR/UEA archives [8] contained between 24 and 2709 samples. Sample lengths up to 1600 seemed reasonable as a starting point. The data could be resampled later if the model training was not successful.

B. Choosing a measure of success

The standard measure of success in classification tasks is the overall accuracy of the predictions made by the model on test data [7]. That is, how many time series are correctly identified as either passing or failing to meet the performance criteria. Another important metric that is measured during training is the model loss. This is a measure of the total amount of error in the model. The training process attempts to minimise the loss value.

In addition to accuracy and loss, other practical considerations of interest were the time required to train the models and overall size (in megabytes) of the models. It is also valuable to know how well the models generalise on input data for various length, resolution or alignment.

C. Deciding on an evaluation protocol

As the model setup requires all input time series to have the same sample length only one of the datasets could be used for training. The 160,000-sample dataset (down sampled to 1600 samples) was selected for two reasons. Firstly, this is the largest dataset and the larger the dataset, the better chance of building a good classifier [7].

Secondly, the pass/fail imbalance in this dataset is the lowest at approximately 85%/15%. Ideally, the training dataset would have an equal split between classes. By having an 85%/15% split in classes, the classifier will have to perform with an overall accuracy better than 85% to be adding any value.

In order to assess the significance of any improvements in the classifier accuracy, the McNemar's statistical hypothesis test will be used. This is recommended by [9] to assess whether the performance of two binary classifiers is significantly different or not. The test is valid where both classifiers use the same test data.

D. Preparing your data

The models were trained on 80% of the dataset. The remaining 20% of the dataset was used for testing the performance of the trained classifier. This is standard practice in machine learning. When doing the training/test split of the dataset it is important to maintain the pass/fail imbalance. The `StratifiedShuffleSplit` class from the `sklearn` library [19] was used for this purpose.

The training and test datasets were then z-normalised. Z-normalisation results in training input values with an average of 0 and standard deviation of 1. Again, this is standard preprocessing step for machine learning. It is particularly important for deep learning models so that the input values are not too large [7].

Once the dataset has been split into training/test datasets and z-normalised, there are several options to address the imbalance in the pass/fail classes. The simplest option is to just use the imbalanced data to train the model. As discussed by [13], other options include oversampling or undersampling the respective classes so that the overall split is 50% between pass/fail. Oversampling involves sampling the minority class with replacement until the number of minority class time series equals the majority class. Undersampling involves sampling the majority class so that the number of time series equals the minority class. Both of these options were tested initially. Two other, slightly more sophisticated methods of resampling the training dataset were also tested.

E. Developing a model that does better than a baseline

Based on the results published by [11] the Fully Convolutional Network (FCN) and Residual Network (ResNet) deep neural networks were selected for this project. Both models performed well when tested using the UCR/UEA archive and compared to other deep learning networks. They also performed well against the state-of-the-art non-deep learning models reviewed by [4]. Detailed architectures of the ResNet and FCN models are shown in Fig. 2.

The ResNet and FCN models were originally applied to TSC by [23]. Both of these network architectures were originally used for computer vision tasks. FCNs were proposed by [17] for the task of semantic segmentation of images. ResNets were proposed by [12] for image classification. This work led to them winning the 2015 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Both of these architectures are variations of the well-known AlexNet Deep Convolutional Neural Network (CNN) proposed by [14] which made CNNs popular for computer vision tasks.

F. Developing a model that overfits

The ResNet and FCN models were trained on the 160,000-sample dataset to get a baseline. From this point the models were expanded to include more convolutional and fully connected layers. Both of these changes increase the feature space of the model and potentially leads to overfitting. The models were also tested with a reduced number of convolutional layers to see the impact.

Overfitting is the phenomenon that occurs when the accuracy and loss of the model improve when measured against the training dataset, but the accuracy and loss of the model deteriorates when measured against another dataset. This means that the model is not generalising well.

The methodology used in this project to avoid overfitting is to hold out some of the training dataset to use for validation. 20% of the training data was not used to train model parameters and instead used to calculate the accuracy and loss of the model after each training iteration (or epoch). The point where the loss is minimum is the point where the model begins overfitting, and this is where training should be stopped. This phenomenon is shown in Fig. 3.

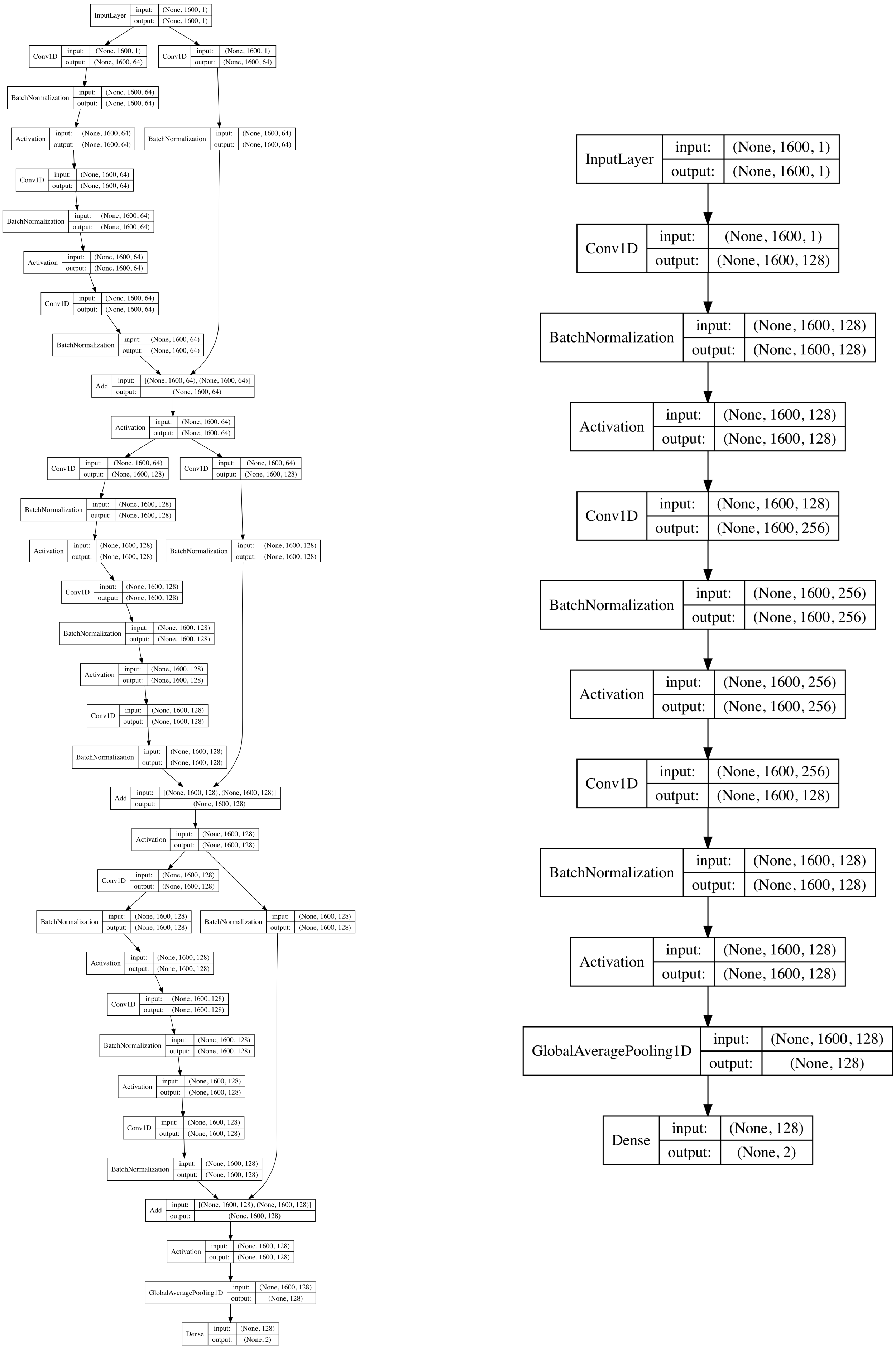
InputLayer input: (None, 1600, 1) output: (None, 1600, 1)
Conv1D input: (None, 1600, 1) output: (None, 1600, 64)
Conv1D input: (None, 1600, 1) output: (None, 1600, 64)
BatchNormalization input: (None, 1600, 64) output: (None, 1600, 64)
Activation input: (None, 1600, 64) output: (None, 1600, 64)
BatchNormalization input: (None, 1600, 64) output: (None, 1600, 64)
Conv1D input: (None, 1600, 64) output: (None, 1600, 64)
BatchNormalization input: (None, 1600, 64) output: (None, 1600, 64)
Activation input: (None, 1600, 64) output: (None, 1600, 64)
Conv1D input: (None, 1600, 64) output: (None, 1600, 64)
BatchNormalization input: (None, 1600, 64) output: (None, 1600, 64)
Add input: [(None, 1600, 64), (None, 1600, 64)] output: (None, 1600, 64)
Activation input: (None, 1600, 64) output: (None, 1600, 64)
Conv1D input: (None, 1600, 64) output: (None, 1600, 128)
Conv1D input: (None, 1600, 64) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Conv1D input: (None, 1600, 128) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
Conv1D input: (None, 1600, 128) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Add input: [(None, 1600, 128), (None, 1600, 128)] output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
Conv1D input: (None, 1600, 128) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
Conv1D input: (None, 1600, 128) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
Conv1D input: (None, 1600, 128) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Add input: [(None, 1600, 128), (None, 1600, 128)] output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
GlobalAveragePooling1D input: (None, 1600, 128) output: (None, 128)
Dense input: (None, 128) output: (None, 2)
InputLayer input: (None, 1600, 1) output: (None, 1600, 1)
Conv1D input: (None, 1600, 1) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
Conv1D input: (None, 1600, 128) output: (None, 1600, 256)
BatchNormalization input: (None, 1600, 256) output: (None, 1600, 256)
Activation input: (None, 1600, 256) output: (None, 1600, 256)
Conv1D input: (None, 1600, 256) output: (None, 1600, 128)
BatchNormalization input: (None, 1600, 128) output: (None, 1600, 128)
Activation input: (None, 1600, 128) output: (None, 1600, 128)
GlobalAveragePooling1D input: (None, 1600, 128) output: (None, 128)
Dense input: (None, 128) output: (None, 2)

Fig. 2. ResNet (left) and FCN (right) model architectures.

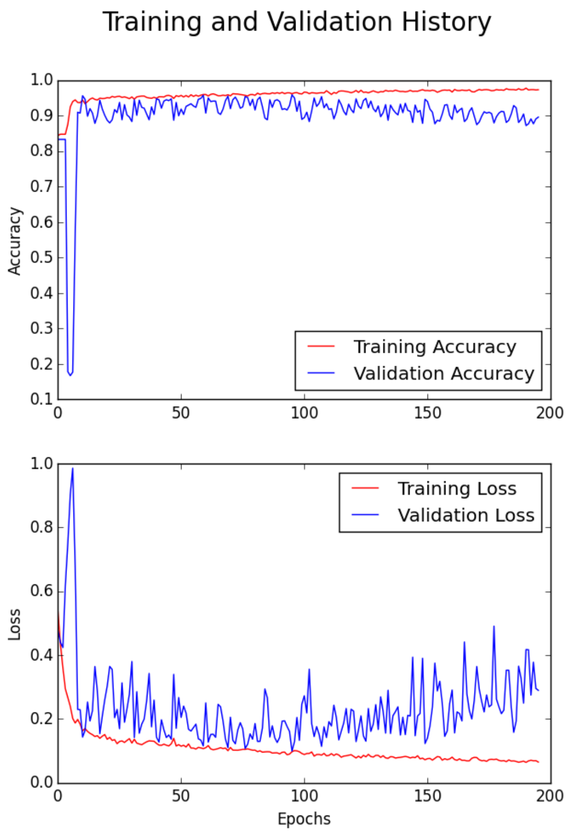

Fig. 3. Model training and validation history metrics showing overfitting starting at epoch 95

G. *Regularising your model and tuning your hyperparameters*

There are a number of other model parameters that can be adjusted to optimise the model performance. Three of those recommended by [7] have been tested for this project.

Learning rate reduction: After each epoch of training, the learning rate of the model optimiser can be adjusted. When the training statistics plateau the learning rate is reduced. The validation loss value is monitored to check for training plateaus. This allows the model weights to be fine-tuned to achieve the best possible test results.

Adding dropout: This is a regularisation technique that randomly sets the output of a layer to zero after each training iteration. The effect of this means that training time is increased but overfitting is reduced [21].

Adding weight regularisation: Another technique to reduce overfitting is to penalise weight values that get too large. This prevents certain parts of the model with large weights from dominating, in effect reducing the distribution of weight values and making all parts of the model contribute to the classification.

H. *Performance on other datasets*

The performance of the best performing models was tested on other datasets as well. FCN and ResNet are types of Convolutional Neural Networks (CNN). One feature of CNNs is that they have translational invariance. This means that the important features of the input data don't have to occur at the same point in the time series. Therefore, for this project the performance of the models on the 80,000-sample and 120,000-sample datasets should be comparable to the results from the 160,000-sample dataset. Translational invariance also implies that the 160,000-sample dataset could be offset along the time axis without impacting the classifier accuracy.

The models were also tested on the 160,000-sample dataset down sampled at different rates. This demonstrates whether the models have scale invariance.

From a practical point of view both of these tests show whether a single, well-trained model can be used for a variety of applications without requiring retraining.

V. Results and Discussion

The data used in this project was provided by Powerlink Queensland from project work completed in 2018 and 2019. The data was stored in output files generated by PSCAD™/EMTDC™ power systems simulation software. The data used has been de-identified prior to analysis to protect Powerlink Queensland and their customers' commercially sensitive information.

Once the dataset was collected, labelled and processed. The next step was to build the classifiers and assess their performance. All classifiers were developed using the `Keras` deep learning library for Python [6]. There are two main reasons for this decision.

`Keras` is a standard library used for deep learning research. The models tested in [11] were written with `Keras` and made available to the public via GitHub. Therefore, it is easy to incorporate these existing published models into the project.

Power System engineers at Powerlink Queensland are familiar with the Python programming language and several power system simulation software packages include a Python API. From a practical perspective, a `Keras`-based classifier could be seamlessly integrated into the power system simulation workflow.

`Keras` can be run on top of various tensor processing libraries. `TensorFlow` was chosen as the tensor processing library for this project. The following software versions were used throughout this project for model training and evaluation:

- Python version 2.7.16
- `Keras` version 2.2.5 using `TensorFlow` version 1.14.0 backend
- `statsmodel` version 0.10.1

The training was conducted on a combination of hardware to assist with speeding up the training time. This included:

- iMac 2019: 3 GHz Intel Core i5 with 8GB RAM running macOS 10.14.6
- Google Collaboratory: online Python notebook with GPU processing support

A. *Comparing Resampling Strategies*

As shown in Table I, the datasets used for this study are imbalanced. In all cases the pass is the majority class and fail the minority. Methods for dealing with imbalanced data used in deep learning convolution models are discussed in [13]. Broadly they are divided into two strategies: resampling techniques or cost sensitive loss functions. Cost sensitive loss functions are designed to penalise misclassification of the minority class more than misclassification of the majority class. Due to time constraints, resampling strategies where tested in this project but cost sensitive loss functions were not tested.

Five resampling strategies were selected and applied to the training dataset. The outputs of this resampling were then used to train the FCN and ResNet models. The five resampling strategies chosen were:

Stratified sampling: Training dataset retains the same imbalance as the original dataset.

Naïve Undersampling: Sample the majority class until there is a 50/50 split in the training dataset.

Naïve Oversampling: Sample the minority class with replacement until there is a 50/50 split in the training dataset.

SMOTE Oversampling: SMOTE is an acronym for Synthetic Minority Over-sampling TEchnique. SMOTE generates new minority class samples. The implementation of SMOTE used for this project is from the `imbalanced-learn` package [16]. See [5] for more details.

NearMiss Undersampling: NearMiss is an undersampling strategy that selects majority class samples that are 'close' to the minority class samples. The implementation of NearMiss used for this project is from the `imbalanced-learn` package [16]. See [18] for more details.

Table 2 summarises the results of the different resampling strategies using ResNet and FCN models. In both cases the simple stratified sampling method produced the best overall accuracy. With ResNet, all of the more sophisticated sampling techniques produced significantly worse results. The results with FCN were more consistent, with only SMOTE producing significantly worse overall accuracy. The models trained with the stratified sampling data will be used as the baseline results in all further tests. The stratified training dataset was used for training all further models.

TABLE II. OVERALL ACCURACY FOR RESAMPLING STRATEGIES

Model	**Resampling Strategy**				
	Stratified	***Naïve Under-sampling***	***Naïve Over-sampling***	***SMOTE***	***NearMiss***
ResNet	**96.98%***	95.18%	91.06%	93.87%	91.26%
FCN	**96.98%***	96.28%*	96.28%*	94.67%	96.48%*

NB: The results in bold have the best overall accuracy for that model. Results marked with an Asterix (*) are the best performing results that are not significantly different based on McNemar's Test (confidence of 95%). For example, The FCN model which used the Stratified training dataset gave the best overall accuracy. However, three of the other training datasets gave results that were not significantly different. This convention applies to all tables that follow. The implementation of McNemar's Test used is included in the `statsmodel` package [20].

It is also worth commenting on the training time required for the two classifiers. The average time to train a ResNet model was 59.1 minutes, while the average for an FCN model was 53.8 minutes. The ResNet model has almost twice as many parameters to train so the training time for each epoch is longer. However, the ResNet model converged on a good solution in fewer epochs. The average number of epochs required to train the ResNet model was 120.8. FCN required 289.8 epochs. The trained ResNet and FCN models are 3-4 megabytes in size which is not onerous to deal with.

B. *Comparison with Non-deep learning model*

In order to compare the performance of the deep learning classifiers, the same training and test dataset was used with a non-deep learning classifier. The study by [3] concludes the K-Nearest Neighbour (KNN) with Dynamic Time Warping (DTW) is a good performing baseline classifier for TSC. KNN with DTW has the advantage that it is very simple to implement and has few parameters to configure and was used by [4] as a baseline comparison for other more complex classifiers.

KNN with DTW was implemented using the `tslearn` package [22]. The overall accuracy achieved was 87.44%. This is much lower than either of the deep learning models and only marginally higher than the baseline of 84.5%. One major disadvantage of KNN based classifiers is the time required to make a prediction. While the training time is almost instantaneous, it took in excess of 8 hours for the 995 records in the test dataset to be classified. If the training dataset was larger the prediction time would increase as well.

On the other hand, both deep learning models took less than 30 seconds to classify the same test dataset. As the training dataset increases, the training time for the deep learning models will increase. However, the time to classify the test dataset will remain the same.

C. *Model architecture changes*

The baseline FCN and ResNet models achieved a good overall accuracy of almost 97%. In order to try to improve this result, the baseline models were adjusted to add more convolution or densely connected layers. Models with fewer convolution layers were also tested. Table III details the changes to each of the models. Table IV compares the baseline model results against three different model architecture changes.

TABLE III. DESCRIPTION OF MODEL ARCHITECTURE CHANGES

Model	**Model Change**	
	Name	***Description***
ResNet	Extra Convolution	A fourth residual block was added.
	Extra Densely Connected	A second densely connected layer added with 256 neurons added after the GAP layer and before the final dense layer.
	Reduced Convolution	The second residual block was removed.
FCN	Extra Convolution	Convolution layer with 128 filters added after the input layer. The Kernel size of this layer is 11.
	Extra Densely Connected	A second densely connected layer added with 256 neurons added after the GAP layer and before the final dense layer.
	Reduced Convolution	The second convolution layer was removed.

TABLE IV. OVERALL ACCURACY FOR MODEL CHANGES

Model	**Model Change**			
	Baseline	***Extra Convolution***	***Extra Densely Connected***	***Reduced Convolution***
ResNet	96.98%	94.77%	97.09%*	**97.89%**
FCN	96.98%*	97.39%*	**97.69%***	96.78%

The best overall accuracy for the ResNet model was with a reduction in the number of convolution layers. The second residual block consisting of 3 convolution layers was removed and the overall accuracy increased by almost 1% which was a significant change based on McNemar's Test. This indicates that the standard ResNet model might be overfitting the dataset used in this project. This suggestion is reinforced by the fact that the model with the extra residual block performed worse than the baseline.

The FCN model accuracy was improved by adding extra layers and removing convolution layers led to a slight decrease in accuracy. However, the changes were not significant based on

McNemar's Test. This reinforces the results from Table II in that the FCN model seems to produce more stable results with regard to changes in the input data and model changes.

D. Model optimisation

Finally, the baseline models where retrained with various optimisations to test their impact on overall accuracy. These optimisations are detailed in the Methodology section and are designed to reduce overfitting.

TABLE V. OVERALL ACCURACY FOR OPTIMISATIONS

Model	Model Optimisation			
	Baseline	*Dropout*	*Weight Normalisation*	*Learning Rate Reduction*
ResNet	96.98%*	96.18%*	96.68%	**97.48%***
FCN	96.98%*	96.98%*	**97.29%***	97.19%*

The ResNet results in Table V show the learning rate reduction produced a 0.5% improvement over the baseline however this was not significant. Again, the results for FCN shown that the training changes produced no significant improvement to the overall accuracy.

In an effort to get the best results, the results from the model architecture change and model optimisation were combined. The ResNet model with reduced convolution layers was trained with learning rate reduction. The overall accuracy was 97.89% which is the same overall accuracy of the model without Learning Rate Reduction. The FCN model with an extra densely connected layer was trained with weight normalisation. The overall accuracy was 97.48% which is not significantly different to the model with only an extra densely connected layer or the baseline model with weight normalisation.

There is a lot of scope for modifying the models combined with hyperparameter changes. Due to time and resource constraints, this project could only test a small sample of these combinations. However, these results might indicate that an upper limit of 98% is what can be achieved with the training and test dataset used on this project. This limitation may be due to errors in labelling the datasets as well as limitations in the models. In either case, more training data could potentially improve these results.

E. Performance on other datasets

Finally, the best performing models were tested on the two other datasets collected for this project. Both of these datasets are shorter, 80,000 and 120,000 samples, and are more imbalanced, 89.7% and 94.7%, respectively. Both datasets were down sampled by a factor of 100, the same as the 160,000-sample dataset. For testing, the input layer of both models was modified to fit the dimensions of the input dataset.

TABLE VI. OVERALL ACCURACY FOR MODEL CHANGES

Model	Other Datasets		
	Best Model	*80,000-sample*	*120,000-sample*
ResNet	97.89%	75.04%	98.45%
FCN	97.69%	44.27%	97.75%

Table VI shows that the models performed well on the 120,000-sample dataset but very poorly on the 80,000-sample dataset. Why did the 120,000-sample dataset perform well while the 80,000-sample dataset did not? The hypothesis for this degraded performance was that the first 40,000 samples were having a negative influence on the accuracy of the shorter time series. This first 40,000 samples of the power system simulation are known as the 'pre-fault' region. This is the time before a disturbance is applied and has no bearing on the assessment of the results. With the 80,000-sample dataset, the 'pre-fault' region takes up half of the time series. To test this hypothesis the 160,000-sample dataset was truncated so that only the first half of the time series was used. Also, the first 40,000 samples were trimmed from the 80,000s and 160,000-sample datasets.

TABLE VII. TRUNCATED AND TRIMMED DATASETS

Model	Other Datasets		
	Truncated 160k	*Trimmed 160k*	*Trimmed 80k*
ResNet	92.86%	98.19%	86.32%
FCN	83.62%	96.28%	82.05%

The results in Table VII show that truncating the 160,000-sampled dataset had a significant negative impact on the model performance. Also trimming the 'pre-fault' region of the 80,000-sample dataset improved the results but still not to the baseline levels. This only partially explains the poor results for the 80,000-sample dataset. The important point to note is that varying simulation sample length from that used to train the model may impact the accuracy of a classifier.

While generating these results it was identified that the z-normalisation parameters were very important to the accuracy of model predictions. If the new datasets were z-normalised independently i.e. using their own average and standard deviation, the results were poor. Instead, if the average and standard deviation of the *160,000-sample training dataset* is used to z-normalise the test dataset, the results were greatly improved.

In addition to these tests on new datasets, the models were tested on modified versions of the 160,000-sample test dataset. The dataset was modified by shifting the data along the time axis between -10,000 and 10,000 samples (in steps of 1000 samples). The results confirm that the models display translational invariance with only a slight (-0.1%) impact on the overall accuracy for both FCN and ResNet models.

Finally, the best models were tested on the 160,000-sample dataset that was down sampled by factors of 50 and 200 (instead of 100 used on the training data). The results were poor with overall accuracies ranging between 85%-86%. This shows that the models do not exhibit scale invariance.

VI. CONCLUSIONS

This project demonstrates that the ResNet and FCN deep learning TSC models performed well on the power system simulation dataset collected. The best models achieved overall accuracies of close to 98%. The results were far superior to the KNN with DTW non-deep learning classifier which was tested.

Results from this project suggest that further improvements may be made by refining the models and hyperparameters. However, this should be coupled with gathering new test data to ensure that the models are still generalising well.

The models displayed translational invariance as expected. But performance on datasets of different lengths varied significantly based on the datasets that were collected. The sensitivity of the models to sample length changes could be explored in further research.

Important factors for the success of the model is that the test data must have the same resolution as the training data. The test data must also be z-normalised using the same parameters as the training data. These two issues show that the models do not exhibit scale invariance.

Further research may also look at generating a Class Activation Map (CAM) for the models. CAM visualisations highlight which parts of the time series were most critical to the class prediction.

In conclusion, this project confirms the results from previous research which showed that the ResNet and FCN deep learning models were well suited to the task of time series classification. It also demonstrates that it is feasible to deploy these classification models in real world applications.

Acknowledgment

The author would like to thank the Network Planning team at Powerlink Queensland for the support and data that they provided. Without which, this project would not have been possible.

References

[1] Al Khafaf, N., Jalili, M., & Sokolowski, P. (2019). Application of Deep Learning Long Short-Term Memory in Energy Demand Forecasting. In International Conference on Engineering Applications of Neural Networks (pp. 31-42). Springer, Cham.

[2] Australian Energy Market Operator. (2018). Integrated System Plan. Retrieved from: https://www.aemo.com.au/-/media/Files/Electricity/NEM/Planning_and_Forecasting/ISP/2018/Integrated-System-Plan-2018_final.pdf

[3] Bagnall, A., & Lines, J. (2014). An experimental evaluation of nearest neighbour time series classification. arXiv preprint arXiv:1406.4757.

[4] Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery, 31(3), 606-660.

[5] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357.G. Eason, B. Noble, and I.N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529-551, April 1955. (*references*)

[6] Chollet, F., & others. (2015). Keras. https://keras.io

[7] Chollet, F. (2017). Deep Learning with Python (1st edition). Manning Publications.

[8] Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C. C. M., Zhu, Y., Gharghabi, S., ... & Keogh, E. (2018). The UCR time series archive. arXiv preprint arXiv:1810.07758.

[9] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. Neural computation, 10(7), 1895-1923.

[10] Fahiman, F., Erfani, S. M., Rajasegarar, S., Palaniswami, M., & Leckie, C. (2017, May). Improving load forecasting based on deep learning and K-shape clustering. In 2017 International Joint Conference on Neural Networks (IJCNN) (pp. 4134-4141). IEEE.

[11] Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019). Deep learning for time series classification: a review. Data Mining and Knowledge Discovery, 33(4), 917-963.

[12] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[13] Huang, C., Li, Y., Change Loy, C., & Tang, X. (2016). Learning deep representation for imbalanced classification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5375-5384).

[14] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[15] Kuo, P. H., & Huang, C. J. (2018). An electricity price forecasting model by hybrid structured deep neural networks. Sustainability, 10(4), 1280.

[16] Lemaitre, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Journal of Machine Learning Research, 18(17), 1-5.

[17] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

[18] Mani, I., & Zhang, I. (2003, August). kNN approach to unbalanced data distributions: a case study involving information extraction. In Proceedings of workshop on learning from imbalanced datasets (Vol. 126).

[19] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

[20] Perktold, J., Seabold, S., & Taylor, J. (2018). StatsModels: Statistics in Python. https://www.statsmodels.org

[21] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

[22] Tavenard, R., Faouzi, J., & Vandewiele, G. (2017). tslearn: A machine learning toolkit dedicated to time-series data. https://tslearn.readthedocs.io/en/latest/

[23] Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In 2017 international joint conference on neural networks (IJCNN) (pp. 1578-1585). IEEE.

[24] Zhang, Y., Huang, T., & Bompard, E. F. (2018). Big data analytics in smart grids: a review. Energy informatics, 1(1), 8.