university of groningen / faculty of science and engineering

Comparing different feature extraction techniques on time series classification with a feedforward neural network

Machine Learning Project 2020-2021

Hristo Stanulov, s4287606
Ritten Roothaert, s2929244
Kaan Yesildal, s4467787
Alfiuddin R. Hadiat, s2863685

Abstract **– This paper compares different feature extraction methods on a feedforward neural network time series classifier. The Japanese Vowel dataset is used, which consists of 12 cepstrum coefficient channels of 9 male speakers vocalizing the vowel "ae". The project aims to apply various feature extraction methods to reduce the method's dimensionality and compare their performance with a 3-layered feedforward neural network. An evaluation pipeline using cross-validation is used to evaluate the models across varying dropout rates and find an optimal model. Despite achieving an accuracy of 78.3% with the testing set, it did not reach the performance of a baseline neural network that used zero-padding. Further points are explored in the discussion section.**

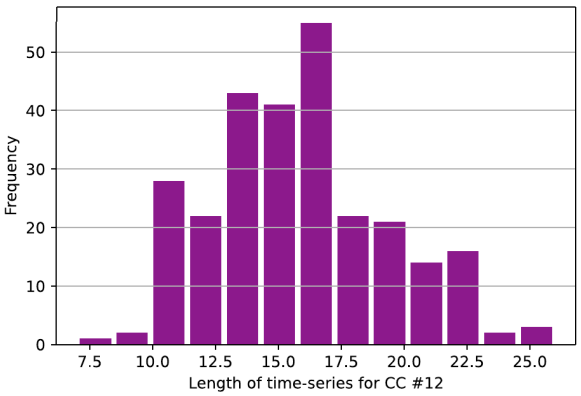

Figure 1.1: Distribution of length for all time-series within channel 12

1 Introduction

Time series is a prevalent form of data structure found across diverse fields that are used for a wide variety of different machine learning tasks. The focus of this paper is to investigate the influence of different feature extraction methods on the performance of a feedforward neural network classifier. Specifically, a classifier for a dataset found on the UCI dataset repository (5) and first collected by Kudo et al. (6). In the last 20 years, machine learning researchers have used the dataset to evaluate a variety of different classifiers.

Generating an accurate classifier for this dataset is not trivial. The dataset contains the recording of 9 Japanese men saying the vowel "ae". There are 12 channel recordings, and each sample length ranges from 7 to 29 data points for the whole dataset. In other words, the lengths of the features are not uniform. We can see this non-uniformity in Fig. 1.1, which shows the distribution of sample length for a single channel in the training set, where the maximum length is 26. Before a classifier can be trained, the dataset should either be transformed to have uniform feature lengths or extract other features from the dataset.

Researchers have used different approaches to generate a classifier for the dataset. Section 2 explores a select number of these approaches. Three feature extraction methods were investigated for feedforward neural networks. The first is the sample segmentation used in (3). The second one is also sample segmentation, but with filling the missing data with zero-padding and the third is passing-through-regions technique proposed by Kudo et al. (6). Specifically, the binning procedure of PTR is used to generate features for the neural network. Moreover, PCA is used for this approach to reduce the dimensionality of the features.

2 Related Works

Over the years, research has shown that most contemporary competitive classifiers work on time series data.

For example, a stacking solution was used to generate classifiers for a variety of publicly available time-series data (7). The results showed that standalone machine-learning algorithms, such as decision trees and support vector machines, were able to achieve good accuracy scores with the time-series datasets. When placed into an ensemble classifier, the stacked algorithms achieved even better accuracy scores.

Other research found similar results with these machine-learning algorithms. A decision trees using a split-criterion for time proximities achieved a performance on par with other classifiers, such as hidden-Markov models and time segmentation approaches (1). A support vector machine was able to achieve good performance when it used featured generated from boosting (8). Several forms of self-organizing maps were able to perform well on time-series data (2). A particular self-organizing map, Merge SOM, achieved an error rate of 1.6% with the Japanese Vowel dataset (9).

The research tells us that there are several valid methods for time-series classification. One method we are particularly interested in is the passing-through-regions (PTR) classifier developed by Kudo et al. (6). PTR will be further expanded upon in section 3. For now, it is enough to note that the PTR was able to achieve a classification accuracy of 94.1% with the Japanese vowel dataset. We are interested in whether the PTR can generate features that would improve a feedforward neural network classifier for the Japanese Vowel dataset.

3 Methods

The process of time-series classification, like most other classification tasks, can be divided into two major components: feature extraction and training the classification model. The Feed-forward Neural Network used for the classification requires an input vector $\mathbf{x}_i$ where $\mathbf{x}_i \in \mathbb{R}^n$ and n is fixed. This is however not the case with the original data set used. As the length of the signals within the training data varies between 7 and 26 units, the dimensionality of $\mathbf{x}_i$ varies between the shortest signal $\mathbf{x}^{min} \in \mathbb{R}^{7\times 12}$ and the longest signal $\mathbf{x}^{max} \in \mathbb{R}^{26\times 12}$.

To overcome this, we implemented three feature extraction methods $\mathbf{f}_{seg\_n}(\mathbf{x})$, $\mathbf{f}_{seg\_z}(\mathbf{x})$ and $\mathbf{f}_{PTR\_PCA}(\mathbf{x})$ such that $(\mathbf{f}_i)_{i\in\{seg\_n,seg\_z,PTR\_PCA\}} : \mathbb{R}^m \rightarrow \mathbb{R}^{60}$ where $m \geq 60$. This implies that regardless of the dimensionality of the input data, it is transformed into a feature vector of length 60. The output of the feature extraction methods has the same dimensionality, such that an easy comparison can be made between the results of the models trained using the different feature extraction methods. The number of extracted features is set to 60 as this will ensure dimensional reduction from the $7 * 12 = 84$ dimensions of the shortest training signal.

To look at the effect of the feature extraction methods, a baseline method is included as well. This method $\mathbf{f}_{base}$ will simply zero-pad all the training signals in $\mathbf{X}$ such that they match the length of the longest signal $\mathbf{x}^{max} \in \mathbf{X}$. As the longest signal in $\mathbf{X}$ has length 26, this method provides a mapping $\mathbf{f}_{base} : \mathbb{R}^m \rightarrow \mathbb{R}^{312}$. While it does not directly match the output dimentionality of 60, training a different model on these features does provide an estimate on what is possible if no variability in the data is lost.

3.1 Segmentation

The first and simplest feature extraction method is the naive segmentation method $\mathbf{f}_{seg\_n}$ (proposed in (3)). This method divides each signal into k equally sized parts and for each channel, calculates the average value of x. By setting $k = 5$, $12 * 5 = 60$ features are extracted from each signal creating a feature mapping of $\mathbf{f}_{seg\_n} : \mathbb{R}^m \rightarrow \mathbb{R}^{60}$. A downside of this method is that the length of the original signal, which is arguably an important feature, is not represented in the extracted features.

This is less of an issue in the second feature extraction method, $\mathbf{f}_{seg\_z}$. This is a variation on the naive segmentation method, but before a signal is divided into k equal parts, the signal is zero-padded such that it has the same length as $\mathbf{x}^{max}$. Afterwards, the same procedure as for $\mathbf{f}_{seg_n}$ is applied. By first zero-padding the signal, shorter signals will in general have a value closer to 0 towards the end of the extracted signal compared to longer signals, allowing them to be distinguished from each other more easily.

3.2 Passing Through Regions

The final feature extraction method $\mathbf{f}_{PTR\_PCA}$ is a more complicated feature extraction method and has two components: an implementation of the Passing Through Regions proposed by Kudo et al. (6) ($\mathbf{f}_{PTR}$) and an implementation of Principle Component Analysis ($\mathbf{f}_{PCA}$) to obtain the desired output dimensions.

Given N training samples in training data set $\mathbf{X}$ where a single training sample is denoted by $(\mathbf{x}_i)_{i=1,\ldots,N}$, the $\mathbf{f}_{PTR}(\mathbf{x}_i)$ method is described as followed. First it subdivides the entire input space into a grid. It uses U equally spaced vertical lines which are located from 0 to the maximum length of all training samples in $\mathbf{X}$ and V equally spaced horizontal lines, located from the minimal value of x to the maximum

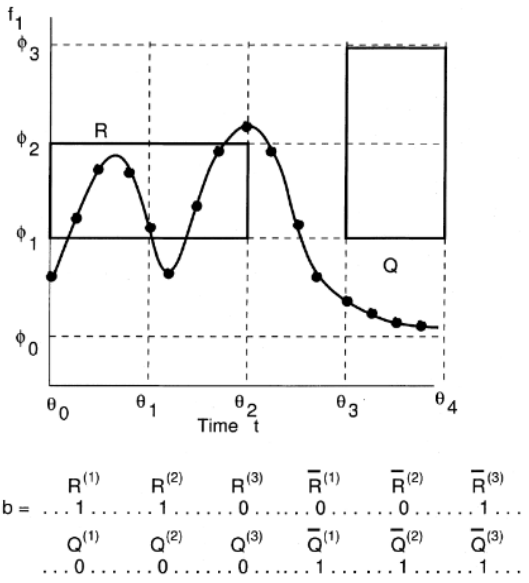

Figure 3.1: Bits related to K=3 for R and Q, as explained by Kudo et al. (6)

value of x. For all possible rectangles $\binom{U}{2} \cdot \binom{V}{2}$ within the discrete domain of this grid, it is counted how often the signal of one channel passes though this rectangle. This count is translated to a bit-signal with the use of parameter K. For all values $k \in \{1, 2, .., K\}$, one bit is set to 1 if the signal passes through the rectangle at least k times and set to 0 otherwise. Another bit is set to 1 if the signal passes through the rectangle less than k times and is set to 0 otherwise. A graphical representation of two arbitrary passing through regions in combination with the associated bits is shown in Figure 3.1. The process is repeated for all dimensions n, where a dimension is analogous to a channel in our current problem. The final result of this method is a feature vector of length B where

$$B = 2nK\binom{U}{2}\binom{V}{2} \tag{3.1}$$

When this algorithm is applied, it provides a feature mapping of $\mathbf{f}_{PTR} : \mathbb{R}^m \rightarrow \mathbb{R}^B$. We decided to use the parameter setting from Kudo et al. (6), where $U = 4$, $V = 20$ and $K = 3$, as it is reported to yield good results. A major downside it that B is substantially bigger than the original number of input dimensions ($\mathbf{f}_{PTR}(\mathbf{x}) \in \mathbb{R}^{82080}$ compared to $\mathbf{x}^{max} \in \mathbb{R}^{312}$).

We decided to reduce the dimensionality using PCA, part of the sk-learn library. This is done by first transforming the original training data set $\mathbf{X}$ by applying $\mathbf{x}'_i = \mathbf{f}_{PTR}(\mathbf{x}_i)$ for all $i \in \{1, ..., N\}$ and combining these $\mathbf{x}'$ into the new dataset $\mathbf{X}'$. Using the first 60 principle components of $\mathbf{X}'$, a feature mapping $\mathbf{f}_{PCA} : \mathbb{R}^{82080} \rightarrow \mathbb{R}^{60}$ is created in which 80.6% of the variability in $\mathbf{X}'$ is preserved. By combining $\mathbf{f}_{PTR}$ and $\mathbf{f}_{PCA}$, the final feature-mapping $\mathbf{f}_{PTR\_PCA}(\mathbf{x}) = \mathbf{f}_{PCA}(\mathbf{f}_{PTR}(\mathbf{x}))$ is created where $\mathbf{f}_{PTR\_PCA} : \mathbb{R}^m \rightarrow \mathbb{R}^{60}$.

3.3 Classification model

For this project, a feed-forward neural network is used to train the input data for the experiment. A feed-forward neural network is an artificial neural network which consists of number of interconnected processing units called neurons. Each of these neurons are modelled from a single processing unit called a perceptron which is used to linearly separate the input data into many partitions using an activation function. Neurons are connected via synaptic connections. These connections send the input data into to next perceptron to be processed by an activation function. Each of these connections are associated with a synaptic weight which is multiplied with the input data before the activation function of the perceptron uses this to process the input data to generate a new output. The structure of an feedforward neural network can be represented by a weight matrix $\mathbf{W}$ where $\mathbf{W}(i, j) = w_{ij}$ is the weight link leading from unit j to unit i. The activation function $\mathbf{f}$ computes the output of the neuron using the $\boldsymbol{\theta}$ as the vector of input weights and u as the activation function such as: $\mathbf{x} = \mathbf{f}(u, \boldsymbol{\theta})$. Generally, a multilayered feed-forward neural network consists of one input and output layer and many hidden layers.

3.3.1 Feedforward Architecture

For this experiment a feed-forward neural network with 2 hidden layers is used. This architecture is a rather shallow one. This is because although increasing the number of hidden layers give more separability to the data, it is more computationally expensive. Furthermore, it can cause poor generalization performance. Multi-layered neural networks are better suited for tasks that have much more samples than the current experiment. Thus, only 270 training samples and 9 classes it is more suitable to use feed-forward neural network with only 2 hidden layers. The input dimensionality is 312 for the baseline method and 60 for the rest of the methods. The number of units in the input layer for all of the methods equals to **inputdimensionality**. The number of units in the first hidden layer is **2*inputdimensionality**. The number of units in both the second hidden layer is **int(inputdimensionality/2)** and the output layer is

9 for all of the methods. Batch Normalization is used for rescaling the layers of the network. This is required because for each batch the inputs to layers distribution might change and this may cause the algorithm to not converge. This is avoided by standardizing input layers for each layer of the network. Batch normalization is used between all layers in the network.

3.3.2 Activation Function

The activation function preferred for the task is the Rectified Linear Unit (ReLU) which is known as the rectifier unit. It can be modelled as follows:

$$f(x) = x^+ = max(0, x) \tag{3.2}$$

This activation function is a piece-wise linear unit. This property of non linearity enables successful application of back propagation given that the ReLU function can be differentiated many times which protects the back propagation from diminishing gradients problem. Mini Batch Gradient Descent is used in the training phase. This method decreases the computation time significantly given that it divides the input data into many partitions(batches) and applies gradient descent on all the batches separately. 10 batches with 26 data samples are created for mini-batch gradient descent.

3.3.3 Loss Function

The cross entropy function is used for the loss function. Cross entropy function is a modified version of the negative log-likelihood of the softmax activation function. The softmax function is as follows:

$$S(f_{yi}) = \frac{e^{f_{yi}}}{\sum_j e^{f_j}} \tag{3.3}$$

The softmax function takes the ratio of correctly classified labels over all the classes to assign probabilities to each observation. Then their negative log-likelihood is calculated as follows:

$$Loss(y) = -log(y) \tag{3.4}$$

and all the negative log-likelihood values are summed over all the classes. This is modelled as follows for the binary classification case using cross entropy:

$$loss = -(ylog(p) + (1-y)log(1-p)) \tag{3.5}$$

For multi-class problems the loss function becomes

$$loss = -\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{3.6}$$

, where M is the number of classes which is 9 in this case. $\mathbf{y}$ is the binary index if class c is the correct classification for observation o, p is the predicted probability of observation o belonging to class c. This activation function is commonly used where there are multiple classes to classify. For each class, a probability p is calculated for every observation to assign that observation to the class of the highest probability. This is possible because of the softmax function (10).

3.3.4 Regularization

Dropout is used as a regularization technique in both hidden layers. Small datasets can lead to overfitting so dropout method randomly chooses a fraction of weights and only trains them at every epoch. This allows for better generalization of the data by allowing for learning of sparse representation of the data. Dropout can cause the data to become noisy given the fact that only a small number of neurons are shown to the previous layer for training. This is caused by training sub optimal neurons because of the absence of the optimal neurons. To avoid this, the weights are averaged over all of the 26 batches.

3.3.5 Optimizer

Adam optimizer is chosen for the problem (4). Adam optimizer is widely used for deep learning tasks as a more robust version of the stochastic gradient descent. It provides reliable optimization when there are sparse gradients. This is suitable for our case given that a shallow feed-forward neural network is used with dropout as a regularization method. This is prone to creating noisy weight vectors because of the small number of weights trained every epoch.

The algorithm calculates both the moving average of the first moment which is the mean, but also the second moment which is the variance. Thus, this enables Adam to keep track of the change in the moving average while comparing this change to the variance of the gradient. Thus, it does not get stuck on local minimums that cause high oscillations in the gradient and can adaptively adjust its learning rate to accelerate the learning process. The usage of the first gradient enables Adam to be computationally inexpensive and to use a small memory size. This adaptive change in the learning rate is suitable for noisy and sparse gradients. On initialization of the network the learning rate is set to 0.001.

4 Experimental Design and Results

4.1 Experimental Design

The experimental design aims to compare the performance of the feature extraction methods with the baseline method. As described in section 3, both the baseline method and feature extraction methods yielded four different datasets. It is important to mention that the feature dimensionality of the baseline dataset is 312, while the other three methods have 60. Comparing the results between the baseline method and feature extraction methods tell us whether the feature engineering improves the performance of the neural network.

The experiments were conducted via an automated pipeline described in algorithm 4.1. The pipeline processes the data, cross-validates the model, and saves the result. Almost all neural network parameters were held constant for experimental validity. The only varying parameter was the dropout rate, which is set for both hidden layers. Varying the dropout rate tells us the performance of a given method for dropout rates ranging from 0 to 0.8 with a step of 0.1. More specifically, changing the dropout rate tells us what complexity yields the best accuracy. K-fold cross validation with $k = 27$ and a batch size of 26 are used to split the training sets into 10 even batches. Given the small size of the dataset, each neural networks trains for 200 epochs with a learning rate of 0.001. These settings prove sufficient for generalization.

As already mentioned, the models are evaluated through 27-fold cross-validation. A higher than average fold was used so that the 270 training samples can be split into folds with 10 samples. While this is computationally expensive, the low amount of epochs makes this a viable option. Cross-validation allows us to counteract overfitting as training and evaluating across all folds yields each model's general performance.

The neural network was implemented using PyTorch running on an NVIDIA GTX 1660 Ti. Each method yielded 243 models, which totals to 972 models across the four methods. For analysis, the accuracy and loss scores of each model are averaged across all folds. The averaged scores are used to evaluate the performance of each model. The feature engineered model with the highest score is determined to be the optimal model. A comparison in performance between the optimal model and the baseline model determines whether the feature extraction method improves the performance of the neural network.

Algorithm 4.1 Cross-validation of a feature extraction method

for $dropout\_rate = 0.0, 0.1, \ldots, 0.8$ **do**
  for $k\_fold = 1, 2, \ldots, 27$ **do**
    Prepare the data
    Create train and validation sets
    Initialize the neural network with uniform weights and biases
    for $epoch = 1, 2, \ldots, 200$ **do**
      Train while iterating through current training batches
      Validate on current validation batch
    end for
    Append training and validation accuracy in a list
    Append training and validation loss in a list
  end for
  Save results into a serialized dictionary of lists
end for

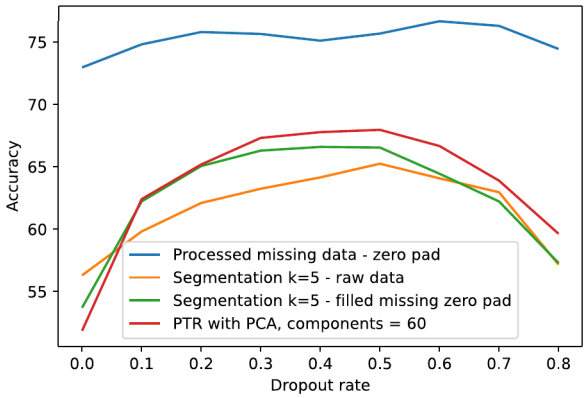

Figure 4.1: Validation Accuracy of all neural networks

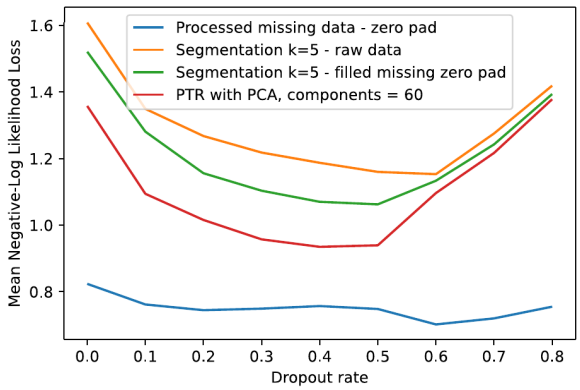

Figure 4.2: Mean Negative Log-Likelihood Loss of all neural networks

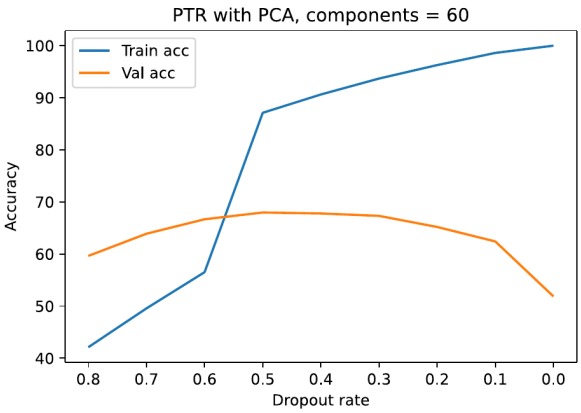

Figure 4.3: Training and Validation of neural network using PTR with PCA

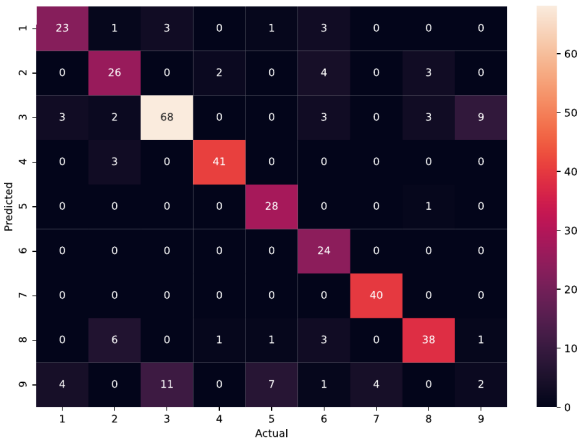

Figure 4.4: Confusion matrix of PTR with PCA neural network classifications on test dataset

4.2 Results

The validation accuracy score of each neural network is shown in figure 4.1. The line plot describes the rise and fall in accuracy across different dropout rates. Except for the baseline model, all models achieved their highest accuracy rate with a dropout rate of 0.5. In all models, the accuracy rate before and after 0.5 is shown to decline. We can see the same trend in figure 4.2, which describes the negative log-likelihood loss of each model across models. Again, except for the baselines model, all models achieved their lowest loss with a dropout rate of 0.5. Figure 4.3 shows the training and validation accuracy for the PTR with PCA model. The figure shows the validation accuracy peaking at a dropout rate of 0.5, which is further evidence for 0.5 being the optimal dropout rate.

According to the accuracy and negative log-likelihood loss scores, the most optimal model is the PTR with PCA. Figure 4.1 shows the model has a better accuracy score than the second-highest accuracy score, which is the 5-segmentation zero-pad filling model. While the accuracy difference is marginal, figure 4.2 shows a clear difference in negative log-likelihood loss, with the PTR with PCA model having the lower loss. As such, the PTR with PCA model was chosen as the optimal model. The model was evaluated using the testing dataset set aside before training.

The optimal model did not reach the performance of the baseline model. This would suggest that the feature extraction methods did not improve the feedforward neural network. Even though the dimensionality was reduced, the raw data with zero-padding yielded the better result. Possibly, the result is due to the zero-padded raw dataset having more dimensionality. Simply put, a higher dimensionality gives the neural network more variance to work with, which results in better optimization.

The PTR with PCA neural network model was able to achieve an accuracy of 78.3% and a negative log-likelihood loss of 0.83 with the testing dataset. This means that the model can make more accurate classifications than random guesses. However, there is a particular issue with the model. Figure 4.4 shows a confusion matrix of classifications made by the model on the testing dataset. The figure shows us that the model was able to achieve accurate classifications with all but the ninth class. This result suggests that the model has classification issues with the ninth class but the reason behind it is not investigated in this paper.

5 Discussion

There were several significant obstacles that we ran into during the course of the project. The first issue was determining what feature extraction methods would fit with a feedforward neural network. The second was figuring out how to implement the methods for the neural network. We chose to leave out two feature extraction methods. We planned to include a peak detection approach but ran into difficulties. Relating peaks together involved the creation of a Bayesian network that resulted in highly correlated features. Further information also had to be extracted through dimensionality reduction, which resulted in a bias towards certain features over others. We also wanted to include average-padding. The method involves extending a sample's channel by using the average value of that channel at a given time step. The method performed worse than the zero-padding segmentation technique, so we deemed it redundant.

Our goal was to see if the feature extraction methods would improve a feedforward neural network compared to a baseline model that employed zero-padding. As shown by the results, the reduced dimensions did not appear to improve the performance of the model. This is possibly due to the fact the neural networks have more weights to optimize if it is given more input dimensions to work with. Understandably, the baseline model would do better simply because it has more variance in the dataset to work with. Either the feature extraction methods do not improve the performance of a feedforward neural network classifier for this time series, or our implementation of it lacked the finesse that would have improved it. Given the confusion matrix shown in figure 4.4, the model appears to have a terrible time classifying the 9th class. Had we more time, we would have investigated the temporal structure between the 9th class and the 3rd class – as the model appears to misclassify the 9th class as a 3rd class – and address the similarity somehow. Indeed, we believe this is one of the main issues with the model.

More personally, we ran into issues figuring out how to translating the feature extraction methods into python as we are individuals still struggling with mathematical notations. The biggest example of this was the passing-through-regions classifier. The paper (6) explained the concept in technical English, but the actual implementation required a thorough understanding of the mathematics. We were able to successfully implement the feature extraction method such that it resulted in features we were able to use for the feedforward neural network, but only after we reduced the feature dimensionality from 82,080 to 60 using PCA. In spite of the difficulties, the struggle with the mathematics made us better at replicating methods from such papers.

We would have been interested to see what the complexity costs are for these feature extraction methods. We assume that the computational cost of PTR with PCA is more expensive than either zero-padding or segmenting because the former involves more processing. The time it takes for PTR and PCA alone is substantial; the combination of the two is possibly very expensive. Unfortunately, we neither had the time to analyze the time and computational complexity of the feature extraction methods, nor did we think it would fit the scope of this paper. This interest would fit were our goal to figure out a deployment pipeline built it repeatedly transform and learn from a model every few weeks or so. Perhaps for a future project, we could investigate the efficiency of including our feature extraction methods in a more engineering-focused domain.

References

[1] A. Douzal-Chouakria and C. Amblard. Classification trees for time series. *Pattern Recognition*, 45:1076–1091, 2012.

[2] B. Hammer, A. Micheli, N. Neubauer, A. Sperduti, and M. Strickert. Self-organizing maps for time series. *Proceedings of WSOM 2005*, pages 115–122, 2005.

[3] M. W. Kadous. Learning comprehensible descriptions of multivariate time series. In *ICML*, volume 454, page 463. Citeseer, 1999.

[4] P. D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv 1412.6980*, 2017.

[5] M. Kudo, J. Toyama, and M. Shimbo. Japanese Vowels Data Set. Retrived 1 December, 2020, from: https://archive.ics.uci.edu/ml/datasets/Japanese+Vowels.

[6] M. Kudo, J. Toyama, and M. Shimbo. Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20:1103–1111, 1999.

[7] O. J. Prieto and C. J. Alonso-Gonzalez. Stacking for multivariate time series classification. *Theoretical Advances*, 18:297–312, 2015.

[8] J. J. Rodriguez, C. J. Alonso, and J. A. Maestro. Support vector machines of interval-based features for time series classification. *Knowledge-Based Systems*, 18:171–178, 2005.

[9] M. Strickert and B. Hammer. Merge som for temporal data. *Neurocomputing*, 64:39–71, 2005.

[10] M. R. S. Zhilu Zhang. Generalized cross entropy loss for training deep neural networks with noisy labels. *arXiv:1805.07836*, 2018.