Towards Zero Feature Engineering: A Comparison between Several Deep Learning Approaches on Time-domain Signal for Human Activity Recognition

Yang Zhang
Human-Computer Interaction Institute
yangz3@andrew.cmu.edu

Wanli Ma
Language Technologies Institute
wanlim@andrew.cmu.edu

Ruochen Xu
Language Technologies Institute
ruochenx@andrew.cmu.edu

1 Introduction

With the advance of electronics, machine learning has been used to power up many applications on modern personal devices, including human-wearable devices. However, many machine learning approaches require non-travail preprocessing steps or proper feature engineering, which could be very time consuming and requires a lot of domain knowledge. To better address this challenge, we are interested to explore the possibilities of "zero feature engineering", i.e. machine learning algorithms directly using the raw data collected by the sensors.

To achieve this, we consider to use deep learning models which have been shown to be successfully trained on raw signals (e.g. images, voices). Many deep learning models are designed to learn filters or kernels, which are sort of like manual features extraction in traditional machine learning, but can be done through iterations of updates. Therefore we hope to investigate the potentials of using deep learning models to achieve zero feature engineering.

In this project, we picked the task of using data collected from a wrist-worn 3-axis accelerometer to classify what activity the user performs, aka., Human Activity Recognition (HAR). We consider this task as a time series classification problem. This tasks has been a research problem of great interest to researchers since it can be used to power many applications such as Internet of Things, human/robot cooperation, and health monitoring. We plot the some of the activities' data collected from all participants in Figure 1. Note that there are three channels, each of which represents data collected from one axis of the accelerometer.

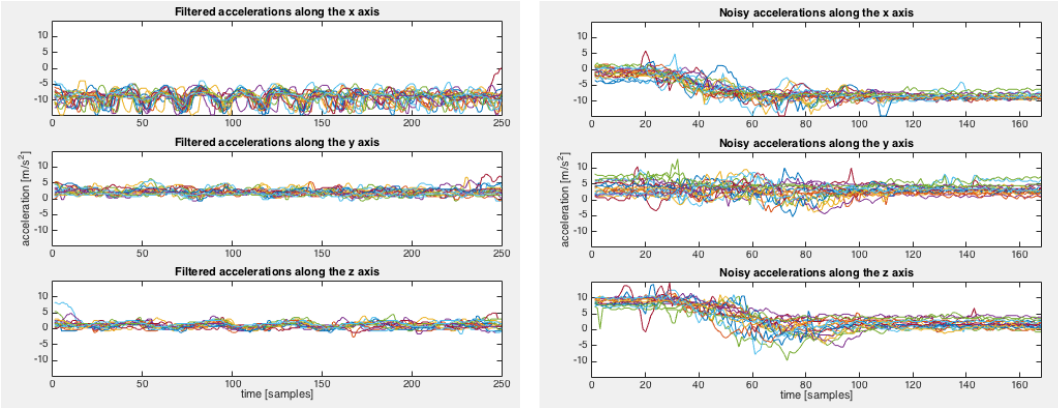
For more details of the dataset: we used the dataset originally published by Bruno et al [3] in their work for human activity recognition. The data was collected from 16 participants (including 5 females) using a wrist-worn three-axis accelerometer with a sensing rage of 3 G, coding the information using 6 bit per axis, at 32 frames per second. Participants were asked to perform seven activities: 1) Climbing stairs 2) Drinking with glass 3) Getup bed 4) Pour water 5) Sit down 6) Stand up and 7) Walk. Enough data points were collected to cover at least one period of the movement. We split the dataset into a training set by 75%-25% for training and test.

2 Baseline

To evaluate the effectiveness of the deep learning models we explored, we first performed a baseline test with a margin-based model – Support Vector Machine (SVM). We select SVM for that it has been widely used in many real world classification problems. By finding the hyperplanes that can best

10807 Class Project (2016).

Figure 1: Trials of two activities



separate characteristic data points (i.e. Support Vectors) between classes, SVM is able to perform robust classifications with a fast training process. In this baseline test, we first compute features based on the raw datapoints. These features include averaged X, Y and Z acceleration signal, derived from which we also compute the statistics such as minimum, maximum, mean, standard deviation, and median. This process results in 370 features. We used SVM implementation from Scikit-learn [8] with default parameters (i.e. RBF kernal with auto gamma value). The result shows an accuracy of 0.829 across all activities.

Frequency-domain features are often helpful for time-domain signal analysis. In this baseline test, we also computed Discrete Fourier Transform (DFT) coefficients using FFT to add more features to the feature set. Figure 2 shows a plot of FFT results of all training data. The seven different activities are rendered in different colors. As we can see most human activities have low frequencies (<1Hz). We used an FFT window size of 32 which produced 17 FFT coefficients, and these 17 features were then concatenated with all features mentioned in the above section, resulting in a feature length of 387. As in the previous section, we used SVM with default parameters and achieved an accuracy of 0.792 across all activities.

It is not surprising to see the performance drop from 0.829 to 0.792 given that all activities' FFT results are all very similar (see Figure 2). This justifies our motivation for using only raw time-domain signals, since having more useless features might decrease the performance.

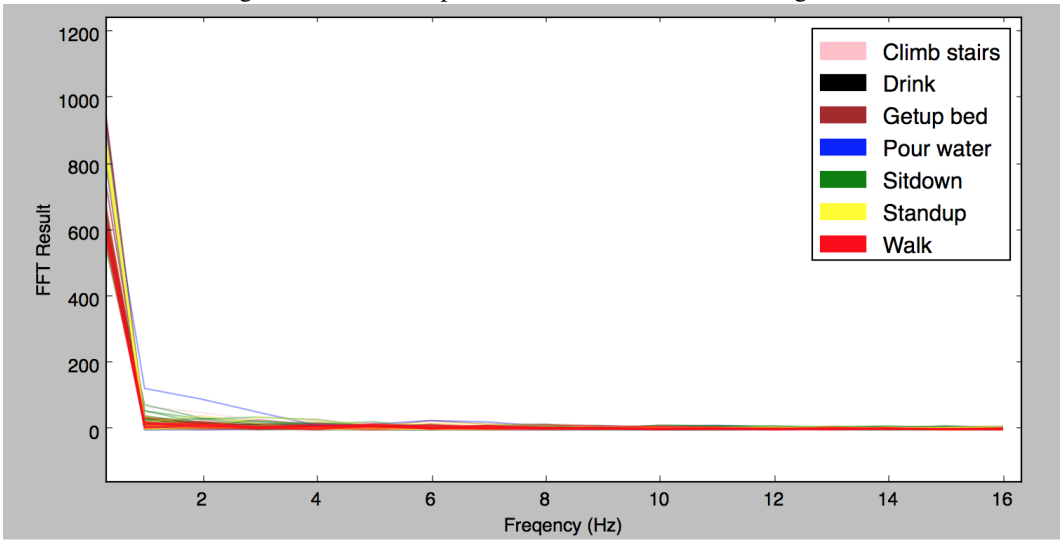
3 Methods and Results

With limited time and resources, it is impossible for us to exhaustively evaluate every deep learning model. Therefore, we picked three representative models that best fit into this problem space, namely the dense neural network, the convolution neural network (CNN) and the recurrent neural network (RNN).

3.1 Dense Neural Network

We started with a basic Dense Neural Network with 1 fully connected hidden layer, implemented using TensorFlow. Classification labels were represented by one-hot vectors and we used softmax cross-entropy as the loss function. We ran this After 300 epoches this model achieved 0.921 ± 0.051 accuracy. Compared with the baseline result from SVM, we saw an improvement of 0.09 in accuracy. This result showed a great promise of using deep learning on raw sensor signal. We then visualized the learned weights (i.e. The first hidden layer). We can see from the plot that weights that for X axis learned the most from the accelerometer data, which makes sense because the x-axis acceleration has the highest signal fidelity. This result showed that our model successfully learned the differences between labels, mostly from x-axis data. Beyond this model, we decided to explore more with different architectures.

Figure 2: FFT result plots from all labels in the training set



3.2 Convolutional Neural Network (CNN)

The CNN we implemented includes two convolutional layer at bottom, followed by a densely connected layer and a softmax layer on the top. Each convolutional layer uses the RELU activation and has a max pooling layer on top of it. Same as the Dense Neural Network, we use the cross-entropy as loss function. The overall networks can be visualized by TensorBoard as shown in Figure 3.

Without too much turning of model parameters, the CNN achieved a test accuracy of 95.7 $\pm$ 3.4%. The accuracy was evaluated by using 10 different random initialization of train/test split and model parameters and reported as mean(accuracy) $\pm$ std(accuracy).

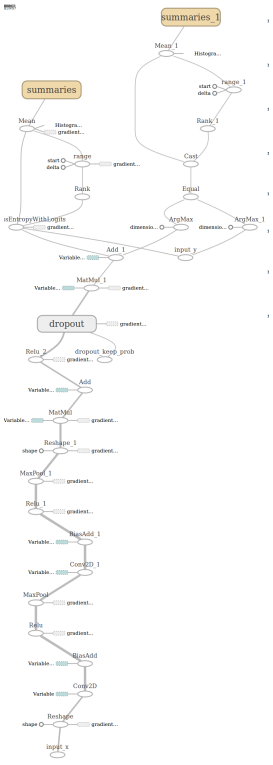


Figure 3: CNN structure

3.2.1 Which axis is more important?

Given the data from the 3-axis accelerometers, we also investigated whether the three axes are equally important for classifying activities. To do this, we run the same CNN model on each axis x, y and z individually and report the classification accuracy in table 1. For comparison, we also put the performance of using 3-axis data in the table. We observe that using only x-axis data could help us to achieve an accuracy almost the same as using all 3 axes, while the performance drops a lot when using only y or z axis. Therefore we believe that the x axis plays a deterministic role in this classification task.

We can somehow explain this finding by visualizing each of the axes. In Figure 3.2.1, each color represents one of the seven class labels. In the figure on the left which plots the time series on the x axis, curves with different colors are easier to be separated compared to the y axis and z axis. However, the difference is not significant to human eyes.

3.2.2 R-CNN: combining CNN with RNN/LSTM

We further investigated other variants of our network structure by replacing the densely connected layer on top of the CNN by a re-

Figure 4: Learned Weights

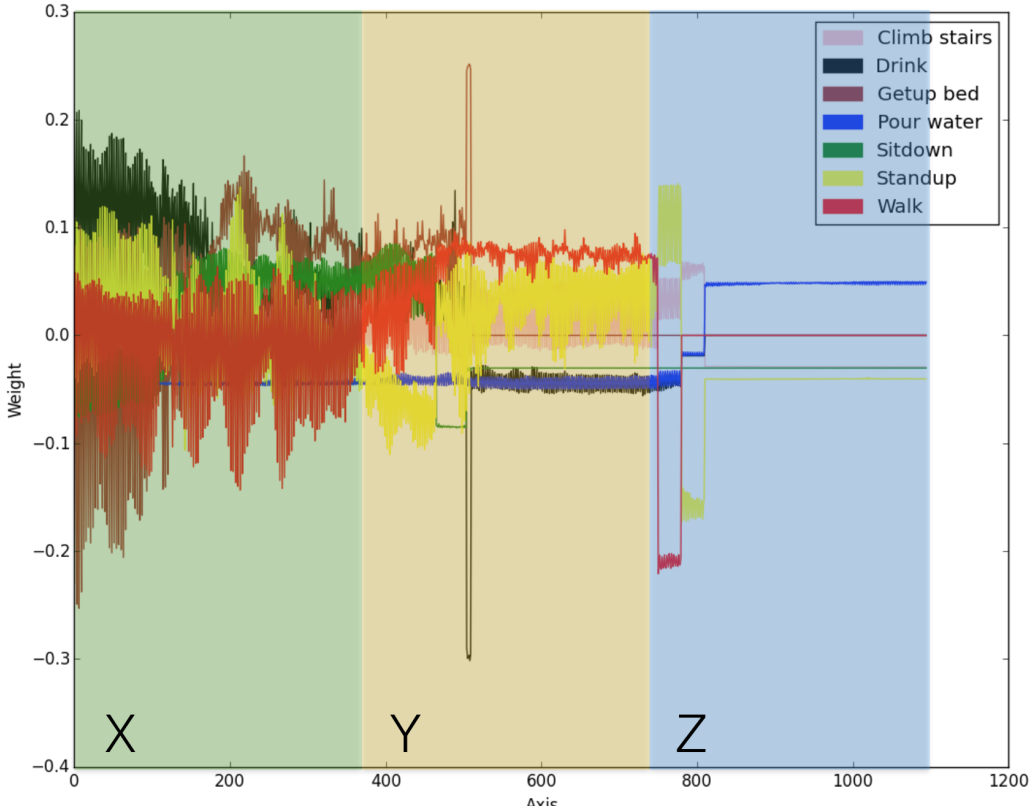


Table 1: CNN performance comparison using 1-D and 3-D data input

	3-axis	x-axis only	y-axis only	z-axis only
Accacy	0.957±0.034	0.943±0.034	0.637±0.051	0.837 ± 0.052

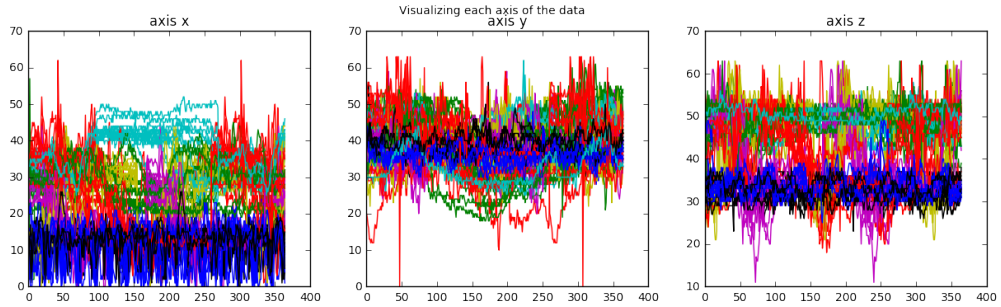
current layer. The intuition behind this is to take advantage of the
ability of RNNs to better capture temporal dependency. Since the filters in the lower convolutional layers are supposed to capture short-time temporal dependencies, we used LSTM as the upper layer in order to capture long-time temporal dependencies.

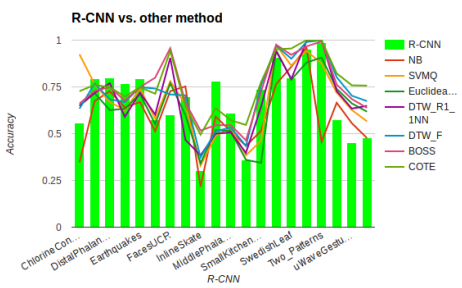
The experiment results are negative. The best accuracy we could get is only 0.786 ± 0.111, which is significantly worse than only using either CNN or LSTM. This result is achieved when we use all the output of the LSTM at each step (i.e. using the matrix of size $N_{\text{LSTM step}} \times N_{\text{LSTM demension}}$) to make a prediction. During training, the softmax loss converges pretty close to zero and the training accuracy achieves 100%, but the validation accuracy remains under 90%. If we reduce the size of convolutional channels or the dimension of LSTM, the accuracy would drop significantly. When only using the last step output of the LSTM, or use a max-pooling layer over the LSTM steps (in this case, using a vector of size $N_{\text{LSTM demension}}$), the training process does not converge and network always predict a single label on the test set.

3.2.3 Result on other datasets from the UCR archive

As we mentioned in the midway report, the classification task on this dataset is easier than we expected, probably due to the nature of the chosen dataset. Therefore we evaluated our CNN and CNN+LSTM models on a broader collection of time series datasets from the UCR archive [4] and compared the performance with other state-of-the-art[1] methods. In the UCR archive their are about 85 time series classification datasets with various sizes, types of source, difficulties and class

Figure 5: Data visualization by axis





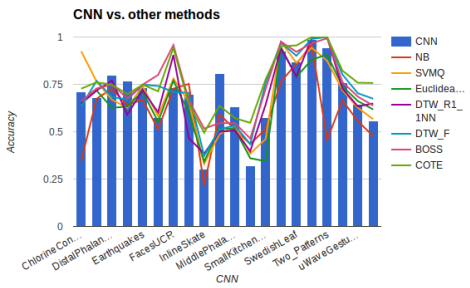


Figure 6: Recursive-CNN (CNN+LSTM) Figure 7: CNN, evaluated on the UCR archive

distributions. Researchers have evaluated a large number of classical machine learning algorithms for time series classification (and some variants) on this data archive, including SVM based methods, Dynamic Time Warping(DTW) based method. Many state-of-the-art feature extraction methods and classification methods for time series are also evaluated on this archive.

We show the results of our model on a subset of the UCR datasets in Figure 6 and 7. The dataset names are listed along the horizontal axes. The green bars in Figure 6 show the accuracy of R-CNN (RNN+LSTM) and the blue bars in Figure 7 show CNN. From the UCR archive website and the authors of [1]'s website, we collected the performance of other baseline methods (including naive Bayes (NB), SVM with quadratic kernel (SVMQ) and DTW+1NN, etc) as well as state-of-the-art methods published in the passed 2 years (including Bag of SFA Symbols (BOSS), Collective of Transformation Ensembles (COTE), etc.), and plotted them using curves of different colors in the two figures. We see that our CNN can achieve a higher accuracy than all other methods on at least 4 out of the 20 evaluated datasets.

3.3 Recurrent Neural Network

Recurrent neural networks(RNN) are designed to make use of sequential information. The advantage of RNN over CNN is that RNN can better capture contextual information. CNN tends to use convolutional kernels with fixed window sizes, which is hard to decide. To address this limitation, we tried three popular RNN models: fully connected RNN(SimpleRNN), Gated Recurrent Unit(GRU)[5] and Long-Short Term Memory[6]. LSTM and GRU models are widely used in modeling sequential data in these days. The earlier model LSTM was proposed to capture long term dependencies, and the more recent GRU model is a simpler but more efficient variant of LSTM. For each model, we feed input sequence to a single layer of RNN, where each cell has 800 hidden units.

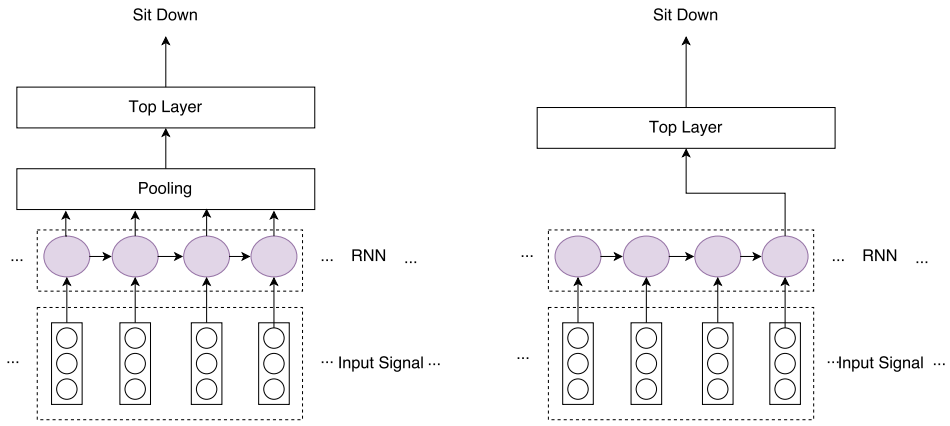
Pooling over time for better accuracy One intuitive way of building classifier on top of RNN output is to stack a top layer on the final output of RNN. However, this design requires RNN to remember all relevant information of long sequence of data to the last time step. Empirically we find the "output until the end" approach performs worse than pooling. To reduce such requirement of long memory, we chose to let each RNN to return its output at every time step, and then apply a pooling layer over each output(pooling over time). With pooling layer, each RNN unit is only required to output correspondingly to its current input and nearby context. During our experiment, we found max

pooling is better than average pooling. The results in this report are all based on max pooling. The intuition is that only certain spans of time series signal are strong evidence of movement type, like the vertical acceleration for sit down and stand up. A max pooling layer will enforce model focus more on those characterized time span. A dropout layer and standard dense layer are added on top of pooling layer. Finally we have a softmax activation layer which generates probability score for each category of movements. See figure 8 for an illustration of our model.

Bidirectional RNN for better accuracy It is a common practice to add one more layer of RNN going in the opposite direction for accuracy improvement. The idea is that we want each RNN output at time t not only capture information of previous signal sequences t, but also the future ones. We concatenate the output of a forward RNN(taking input at time order) with a backward RNN(taking input at reverse time order). Figure 9 shows the architecture of bidirectional RNN(with pooling layer). Table 3 shows the accuracy improvement by having this bidirectional RNN architecture. We can see that bidirectional RNN always outperform forward ones.

Chopping for speeding up training It is suggested by [7] that pooling has the merit of enabling faster training via chopping. Since pooling enables selection of relevant span of time, it is no longer necessary to feed the RNN model from the first signal sequentially to the end. Instead, we can chop the time series data into segments of fixed length that is sufficiently long to capture some context information. Then the RNN is run on each segments as mini batches in parallel. In our experiments we found that directly chopping harms the performance, so we make each segments slightly overlap with each other. The maximum length of our signal is 365, we segment it into 5 segments of length 73 and concatenate with an overlapping of length 20. In table 4 we show how the chopping efficiently reduces the running time without sacrificing the accuracy.

Figure 8: Forward RNN for time series classification. The left one is RNN with max pooling on every output of each time step. The right one stacks the top layer on the output on the last time step



3.4 Summary of Results

Table 2: Performance Comparison with Different Methods

	SVM	DNN	CNN	SimpleRNN	GRU	LSTM
Accuracy	0.829	0.921±0.051	0.957 ± 0.034	0.974±0.0086	0.942±0.0000	0.971±0.0128

We can see that SimpleRNN achieves the best performance over linear baseline and other neural methods. This shows recurrent model is quite suitable for classifying time series. We also notice that CNN only falls behind SimpleRNN by less than 2%. Although originally designed for image data, CNN could also be successfully applied on time series data.

Figure 9: Bidirectional RNN for time series classification.

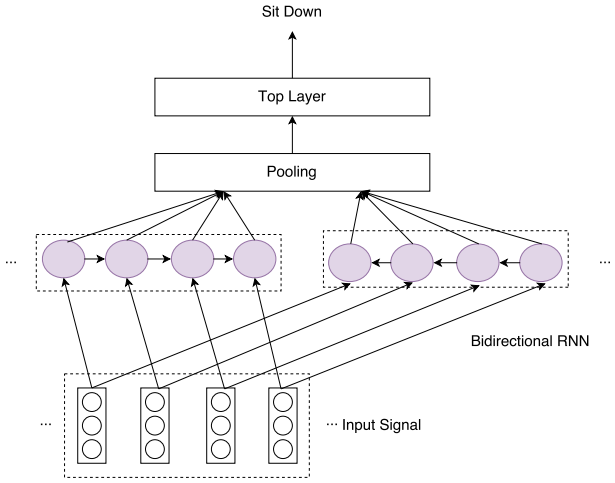


Table 3: Performance comparison of forward RNNs with bidirectional ones

	SimpleRNN	GRU	LSTM
Forward	0.971 ± 0.0000	0.909 ± 0.0280	0.946 ± 0.0085
Bidirectional	0.974 ± 0.0086	0.942 ± 0.0000	0.971 ± 0.0128

The more sophisticated LSTM and GRU fail to perform better than SimpleRNN, suggesting that they may suffer from overfitting with our relatively small training dataset. Even with same number of hidden unit, LSTM and GRU both have more parameters than SimpleRNN.

4 Future Work

We have explored using deep learning approaches with zero feature engineering and got promising result. This encouraged us to go further to deploy our system on real-time systems which are more practical but more challenging. There has been many open-sourced System-on-Chip (SoC) systems on the market, which significantly lower the barrier for implementing sensing hardware. For example, Arduino[2] which has been widely used in many hardware prototypes, mostly are equipped with micro controllers that have at least 27k bytes free flash memory and 16KHz clock frequency. These platforms are not suitable for training deep learning models, however, they can be used to carry trained models for real-time sensing application. We will explore that direction an verify our explorations on real-time applications.

We would also carry out more detailed analysis on the performance of CNN on other time series classification datasets to better understand its advantage and disadvantage. We have observed that neither neural networks nor other state-of-the-art time series classification methods can dominate the others in all UCR datasets, and understanding which methods are good for what type of data is very meaningful to study.

References

[1] Anthony Bagnall, Aaron Bostrom, James Large, and Jason Lines. The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version. *arXiv preprint arXiv:1602.01711*, 2016.

[2] Massimo Banzi. *Getting Started with Arduino*. Make Books - Imprint of: O'Reilly Media, Sebastopol, CA, ill edition, 2008.

Table 4: Accuracy and running time comparison for chopping. 1×365 means there is no chopping, 5×73 means each sequence is chopped into 5 segments of length 73. Time is the running time for one epoch with GPU.

	SimpleRNN	GRU	LSTM
Chop	1×365	1×365	1×365
Time	27s	70s	94s
Accuracy	0.974 ± 0.0086	0.942 ± 0.0000	0.971 ± 0.0128
Chop	5×73	5×73	5×73
Time	6s	13s	17s
Accuracy	0.974 ± 0.0086	0.931 ± 0.0190	0.994 ± 0.0114

[3] B. Bruno, F. Mastrogiovanni, A. Sgorbissa, T. Vernazza, and R. Zaccaria. Human motion modelling and recognition: A computational approach. In *2012 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 156–161, Aug 2012.

[4] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[5] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Rie Johnson and Tong Zhang. Supervised and semi-supervised text categorization using lstm for region embeddings. *arXiv preprint arXiv:1602.02373*, 2016.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.