L-EEGNet: An Ensemble Model for EEG Multivariate Time Series Data Classification

Jingchao Luo, Peter Nguyen, Sean Tang, Oliver Wang
University of California, Los Angeles

Abstract

In this paper, we investigate several Deep Learning based model architecture to tackle the difficult task of EEG Multivariate Timeseries Classification (MTSC) for small datasets. We introduce L-EEG net: a CNN+GRU/CNN/LSTM ensemble model that performs above classic models in the field. We discover several novel qualities about our dataset such as redundant channel information, and confirm them through kernel visualization. Finally, we reflect on the tendency of post-CNN architecture to overfit small-dataset MTSC.

1. Introduction

Electroencephalography (EEG) signals are of scientific and applications-based interest in many fields, ranging from robotics to healthcare. Understanding and decoding such signals is a primary point of improvement that our research aims to examine.

The dataset in this study is the Data set 2A from the BCI Competition IV. The training data consists of 2115 samples 22-channel, Multivariate Time Series data sampled at 250Hz sample rate, band-pass filtered to between 0.5-100 Hz, and is partitioned between 4 classes. From this set of data, we decide that the main challenge to tackle would be to predict the label of a piece of data given the EEG readings. This has applications in many fields, such as allowing people with motor disabilities to perform daily tasks and communicate more effectively.

1.1. Prior Works

Analyzing time-series data has been extensively studied in the past, with many algorithms being developed without the use of deep learning. A review study [6] pitted many of these classical techniques against one another on a wide variety of datasets. One of the high-performing contestants was Random Convolutional Kernel Transform (ROCKET) [2], which uses convolutional kernels, but not in the fashion of Convolutions Neural Networks (CNNs). Instead it pools the data and performs data augmentation by concatenating the convolutions back to the original data, finally performing ridge regression to make predictions.

Deep learning, on the other hand, has shown a recent increase in popularity for analyzing all kinds of data, including for MTSC tasks. In [7], Schirrmeister et al. specifically discuss using CNNs to analyze EEG data. From the findings, we conclude that CNNs can extract a large amount of information from the features of the data, such as by learning spectral power properties. The paper also documents hyper-parameter choices that we took into account in designing our own choices.

2. Materials and Methods

2.1. Data Exploration

To understand the dataset and gain better insight, we first conduct frequency domain analysis. In the frequency domain, we notice that the mean of the test data has a stronger high frequency component than the training data. We theorize that this difference in train and test data, while possibly problematic, can be overcome by CNN layers that can learn to filter this high frequency noise.(Figure 6).

Next, we notice that across many different channels and labels, the time series data past time-step 600 is similar and may not provide much information. We hypothesize that for most models, the removal of time series data past time-step 600 will not degrade performance (Figure 6).

Furthermore, we calculate the cosine similarity across different channels between data in different classes. Our results show that channels 1 through 6 has the lowest cosine similarity in time-domain on average across different labelled data, meaning these channels are good discriminators for different channels in the time-domain. Channels 20-22 on average have high cosine similarity (~ 0.95) between different labelled data in the time domain (Figure 7).

We also compared the increase in cosine similarity caused by the pooling (i.e. down-sampling) performed on the data. We found that for channels 1 through 6, there is an average increase of 10% in cosine similarity, demonstrating a loss of information from down-sampling. For channels

20-22 however, there was not an appreciable change to cosine similarity, further supporting our theory that channels 20-22 does not contain significant information that can be used to discriminate between classes. We hypothesize that for some models, performance will not degrade even with the removal of channel 20-22.

2.2. Data Augmentation

Due to the relatively low amount of data provided, data augmentation is done to increase the amount of samples used for training. We first split the train/val data on a 80/20 training/validation split. For each original input sample, we generate several more samples with zero-mean 0.5 std dev noise added. The result of data augmentation is 7616 samples for training and 844 samples for validation.

2.3. Model Architectures

2.3.1 Architecture Choices

The architectures that we experimented with primarily revolved around CNNs and RNNs, specifically LSTMs and Gated Recurrent Units (GRUs). CNNs are able to capture dependencies between the different timesteps of data, while RNNs are better at capturing states of the data and encoding it as information to change the output predictions dynamically based on previous history. We sought to compare how well each of these architectures fared against each other. We also tried to document any interesting results obtained from combining the two.

2.3.2 Hyper Parameters

While there were many hyper parameters, we kept some consistent across the trials. In general, we used a cross entropy loss function, the Adam optimizer [4] with a learning rate of 0.001, a batch size of 64, and unlimited epochs with early stopping on the validation loss with a patience of 5, usually stopping around 30-50 epochs. We also experimented with implementations in both PyTorch and Tensorflow/Keras.

3. Model Performance

3.0.1 Classic Model: ROCKET

We experimented with ROCKET, a non-deep learning, convolution feature extraction model for Multivariate time series classification. We used a hyper-parameter of 500 kernels. The model only reached a peak test accuracy of 0.2009 and peak training accuracy of 0.49.

3.0.2 CNN

For CNN, we experimented with several novel model architectures. We examined popular CNN based EEG classifiers, and decided to implement EEGNet [5]. Our own implementation includes modifications to model hyper-parameters such as dropout and FC neuron size as documented in Table 2.

In the end, we achieved 71%+ accuracy on our test data using the CNN only model. We trained the model using the Adam optimizer with an initial learning rate of 1e-3 for 60 epochs in PyTorch.

Inspired by [3], we experimented with Residual Networks and found that they had a strong tendency to overfit, requiring high dropout rates at 0.7 to reach 60% accuracy.

3.0.3 RNN

We experimented with various RNN architectures to exploit time-series dependence, including LSTM, Bidirectional LSTM, GRU, and Bidirectional GRU. Overall, the best test accuracy we achieved was 0.5395 using a minimal Bidirectional LSTM architecture followed by a single fully connected layer and softmax activation function as documented in Table 4.

3.0.4 CNN+RNN

We examined cascading RNNs after our CNN model to incorporate temporal relationships into the high-performing model. These included a single LSTM, GRU, or Bidirectional LSTM after the CNN implementation. We achieved the best performance of these using a CNN+GRU depicted in Table 3.

3.0.5 TSMixer

We experimented with TSMixer, a state of the art model for multivariate time series classification [1]. We did not modify the TSMixer, except for it to accept our input data. Our results show that TSMixer performed worse than both CNN and CNN+LSTM, coming in at 30.6% accuracy using the AdamW optimizer in PyTorch.

3.0.6 Ensemble Model

We used a majority vote ensembling of three Keras models (CNN, LSTM, CNN+GRU), that had test accuracies of 0.7156, 0.5395, and 0.7088 respectively. Figure 1 shows this visually. The majority vote ensembling achieved an improved accuracy of 0.7223.

4. Results

4.0.1 Optimizing for All Subjects

The results for training on the whole dataset with all the subjects are shown in Table 1.

4.0.2 Optimizing for One Subject

We trained a model using the data from only a single subject out of the 9 total and evaluated its performance on its own test data as well as the rest of the subjects' test data to see how well the model would generalize to other subjects. We repeated this process for all 9 subjects and the results can be seen in Figure 5. The architecture we used was a CNN only EEGNet architecture with an increased dropout rate of 0.7 in order to account for the large reduction in training data. The architecture was kept constant across all trials. The model performed significantly better when trained on some subjects, while struggled on others. The minimum test accuracy was 0.34 for subject 3 and the maximum test accuracy was 0.79 for subject 8.

4.0.3 Accuracy as a Function of Channels

Based on our hypothesis from data exploration, we conducted an experiment where the channels fed into the EEG-net CNN model was reduced to only the first 20 channels. Our results show that it is possible to reach an accuracy of 69.09% using only the first 20 channels using an Adam optimizer with starting learning-rate 1e-3 in 60 epochs, shown in Table 1.

This supports our theory that channels 20 - 22 do not contain significant information which can be used to discriminate between classes. This finding shows that we can further reduce the size of our model thanks to the lower channel count.

4.0.4 Accuracy as a Function of Time

We experimented with several time "cutoffs" between 100 to 1000, at intervals of 100. At each cutoff we trained a PyTorch implementation of the EEGnet CNN using Adam optimizer at learning rate 1e-3 for 30 epochs with 22 channels. We found that the best validation result occurred for time-step 600 as can be see n in Figure 3.

This supports our hypothesis from data exploration, when we theorized that information past time-step 600 is unnecessary for discriminating between classes.

Curiously, when we reduced the number of channels used based on our findings from the previous subsection while using a "cutoff" of 600, we found that the model performed worse. This leads us to believe that while our intuition from data exploration is correct, another hidden factor impacted by our choice of channel count and choice of time cutoff is the neurons present in the final FC layer of the model. Perhaps in order for the model to reach accuracy of above 70%, it is also necessary that there be at least 144+ neurons in the final FC.

On the CNN-LSTM model we notice that there was diminsing returns after a "cutoff" of 800. After 800, the model performance starts suffering.

For the LSTM-GRU model, we found that there was no appreciable difference between time series after 100 due to over fitting from the RNNs as can be seen in Figure 4. The validation accuracy over different time cutoffs are quite similar after time step 200.

5. Discussion

The CNN model we experimented with performed quite well, with accuracy similar to our ensemble model. Upon further visualization of our CNN model's kernel, we discover several interesting features learned by our model.

In convolution layer 1, the convolution layer focused on convolution in the time-domain, and we discover that filter 6 resembles a time-shifted band pass filter, with symmetry across its middle with varying amplitude reminiscent of sinc (Figure 8).

In Convolution layer 2, we see the relationship between different channels. While the mean across different kernel would seem to indicate that all channels were useful in learning (Figure 11, Figure 9), it's important to point out that there is also a 20 channel version of the kernel capable of 70% accuracy. It's possible the filter happening in Convolution Layer 1 discovered features within Channel 20-22 that were hidden from cosine-similarity.

Surprisingly, the CNN+GRU model performed slightly worse than the CNN alone, despite using the same CNN implementation. We hypothesize that the CNN+GRU model may be overly complex since due to the short length of our time steps, leading to slight overfitting.

Despite the LSTM having a lower accuracy of 0.5395, the ensembling of three models led to better accuracy than any of the individual models alone. We suspect that the CNN and CNN+GRU model accurately classified the LSTM's shortcomings to make up for its lower performance.

Our result show that out of all none-CNN models, the TSMixer performed the worst. After 20 epochs, the model started showing significant over fitting on the training set. This is a common problem among none-CNN architectures, where the complexity of the model often leads to over fitting on smaller data-sets, obscuring the true underlying structure of the data.

Overall, our study show that deep learning based methods perform significantly better than classical methods such as ROCKET. Among the deep learning methods, there were interesting results between CNN and Post-CNN model. While Post-CNN models can still perform worse than CNN due their complexity, a intelligent ensemble of CNN and Post-CNN models can lead to boosted results.

References

[1] Chen, S.-A., Li, C.-L., Yoder, N., Arik, S. O., and Pfister, T. (2023). Tsmixer: An all-mlp architecture for time series forecasting. 2

[2] Dempster, A., Petitjean, F., and Webb, G. I. (2020). Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495. 1

[3] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. 2

[4] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization. 2

[5] Lawhern, V. J., Solon, A. J., Waytowich, N. R., Gordon, S. M., Hung, C. P., and Lance, B. J. (2018). Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 15(5):056013. 2

[6] Middlehurst, M., Schäfer, P., and Bagnall, A. (2023). Bake off redux: a review and experimental evaluation of recent time series classification algorithms. 1

[7] Schirrmeister, R. T., Springenberg, J. T., Fiederer, L. D. J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W., and Ball, T. (2017). Deep learning with convolutional neural networks for eeg decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420. 1

A. Appendix: Performance of Algorithms

Test Accuracy of Various Models	
Architecture	Test Accuracy
sktime	
ROCKET	0.2009
PyTorch Models	
CNN 600/22	0.7168
CNN 1000/20	0.690
CNN	0.7156
TSMixer	0.3064
Keras Models	
EEGNet (CNN)	0.7156
LSTM	0.5395
CNN+GRU	0.7088
L-EEG/Ensemble	0.7223

Table 1. Test Accuracies of Various Model Architectures

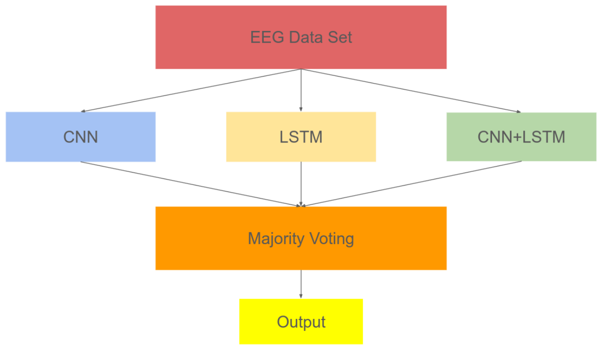

Figure 1. Ensemble Model majority voting scheme

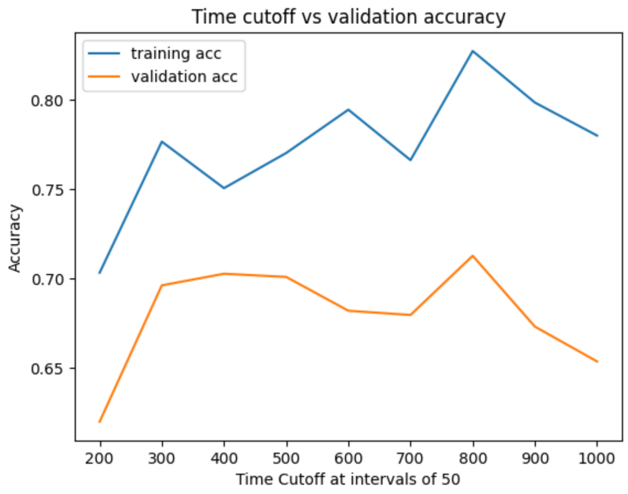

Figure 2. Time cutoff vs Validation accuracy for CNN

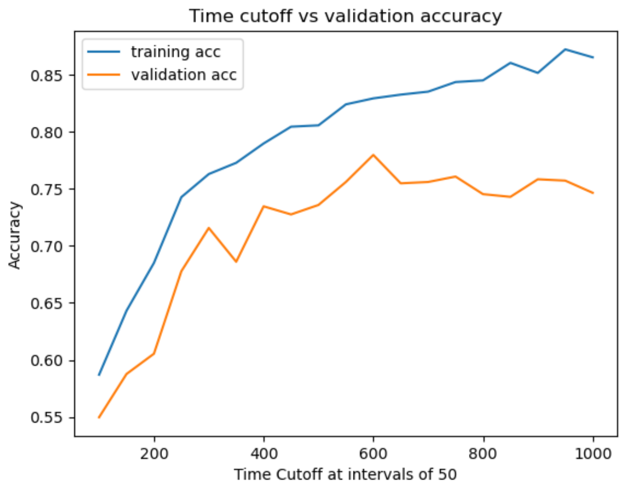

Figure 3. Time cutoff vs Validation accuracy for CNN

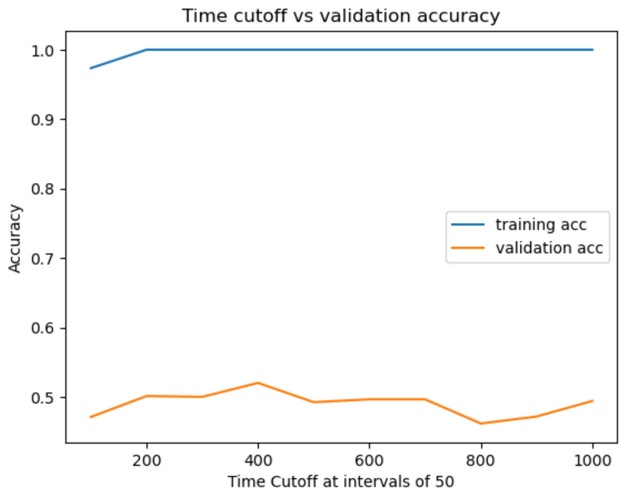

Figure 4. Time cutoff vs Validation accuracy for LSTM

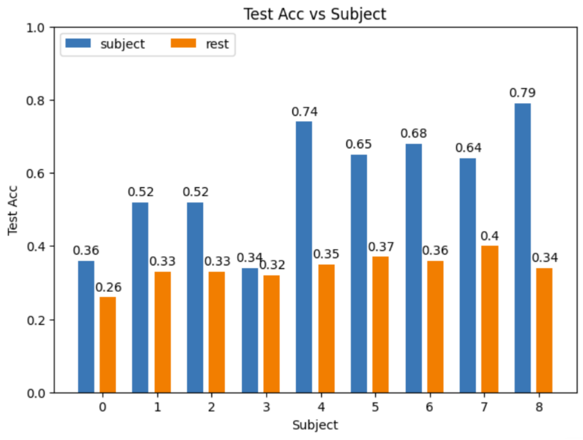

Figure 5. Test Accuracy vs Subject for the optimize for one subject experiment. The test accuracy for the subject the model was trained on as well as for the rest of the subjects is shown.

B. Appendix: Visualization

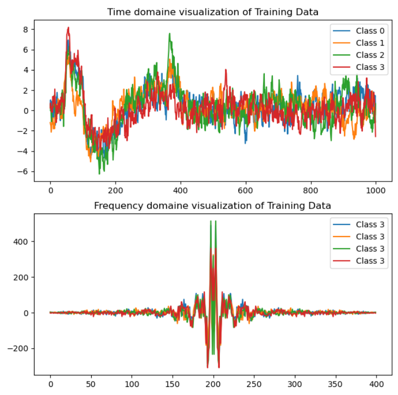

Figure 6. Comparison of Different classes within Channel 1 Data

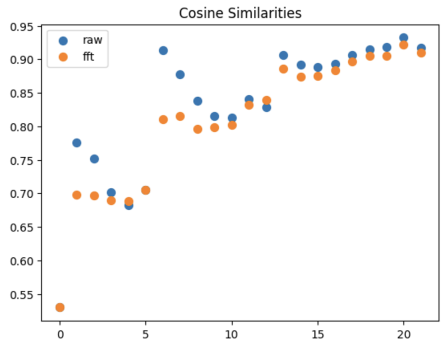

Figure 7. Cosine similarity across channels 1-22 in Frequency domain and time domain

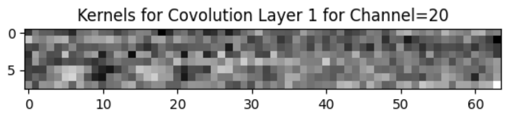

Figure 8

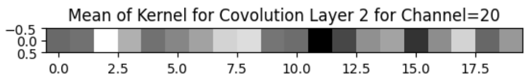

Figure 9

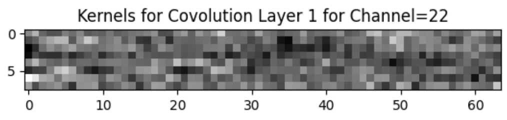

Figure 10

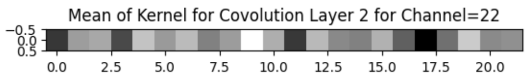

Figure 11

Figure 12. X-axis is kernel size, Y-axis is kernel number

C. Appendix: Model Architecture

Layer	**Output Shape**
Conv2D	8 filters, kernel shape=(1,64) ,
BatchNorm	
Conv2D	16 filters, kernel shape=(18,1)
BatchNorm	
ELU	
AvgPool2d	kernel shape=(1,4)
Dropout	dropout=0.5
Conv2D	16 filters, kernel shape=(1,16)
Conv2D	16 filters, kernel shape=(1,1)
BatchNorm	
ELU	
AvgPool2d	kernel shape = (1,8)
Dropout	dropout = 0.5
Flatten	dim=1
Classifier	1200 $\rightarrow$ 4

Table 2. Summary of Keras/Torch CNN Model

Layer	**Output Shape**
Conv2D	8 filters, kernel shape=(1,64) ,
BatchNorm	
Conv2D	16 filters, kernel shape=(18,1)
BatchNorm	
ELU	
AvgPool2d	kernel shape=(1,4)
Dropout	dropout=0.5
Conv2D	16 filters, kernel shape=(1,16)
Conv2D	16 filters, kernel shape=(1,1)
BatchNorm	
ELU	
AvgPool2d	kernel shape = (1,8)
Dropout	dropout = 0.5
Time Distributed Flatten	dim=1
Bidirectional GRU	units = 32, return_sequences = True
Flatten	dim = 1
Classifier	1200 $\rightarrow$ 4

Table 3. Keras CNN+GRU Model

Layer	**Output Shape**
Bidirectional LSTM	32 units ,
Flatten	
Softmax Classifier	4 units

Table 4. Keras LSTM Model