EC ENGR C147 Final Project Writeup: EEG Classification

Nicholas Chu
UCLA
Los Angeles, CA, 90095
nickchu8@g.ucla.edu

Abstract

The aim of this project is to perform classification on an EEG dataset. Additionally, this project compares the performance training on one subject to training on all subjects as well as how performance varies over time. The dataset used is classified into four different motor imaginary tasks that the subject imagines doing: moving the right hand, left hand, both feet, and tongue. Classification was accomplished by constructing a convolutional neural network (CNN), CNN with LSTM, and CNN with GRU. I found that the best performance occurred with the traditional CNN, with a testing accuracy of 70.6%.

1. Introduction

Convolutional neural networks (CNNs) for EEG classification have become increasingly popular: the CNN that I constructed was heavily based on [1] especially. In addition to the traditional CNN, due to the fact that the model is working on sequential time-series data, I also investigated model architectures incorporating LSTMs and GRUs. In particular, the model architecture for both was inspired by [2], who found that for EEG classification, their best results came with a CNN+GRU combination. Therefore, all in all, the three models I constructed were a CNN model, a CNN+LSTM model, and a CNN+GRU model. Using these models, we answer the questions:

1. How does performance compare when training on one subject as opposed to all subjects?
2. How does the performance vary over time?
3. What kind of architecture is best suited for this task?

2. Dataset: Pre-processing, Augmentation

The EEG dataset is collected from 22 electrodes across 1000 time points across 9 different subjects with a total of 2558 trials. Within these trials, the subject imagines moving their right hand, their left hand, both feet, or their tongue, so these four actions are what the networks will be classifying. Before classification, however, we must perform exploratory data analysis on the dataset to gain insight on how to prepare it for training, then properly pre-process and augment it for our needs.

2.1. Exploratory Data Analysis

After extraction, we find that the X data has a shape of (N, 22, 1000), where N is the number of trials collected. The y data is of shape (N,), with labels from 769 to 772, indicating the four different actions. Plotting the average of the data of each electrode (channel) over time yielded the result that after the time point of around 500, the signals flatten out, become noisy, and do not give distinct insight. Therefore, I trained most of my models only including the first 500 datapoints to see if accuracy would improve by cutting out the unnecessary time points. I additionally found that each of the 9 channels had slightly different shapes in their signals, but not to the point where it was drastic.

2.2. Pre-processing

First, the labels of y are adjusted to be from 0 to 3 instead of 769 to 772. Next, the time point that the data should be included up to, which is a hyperparameter set to 500 as a default, is specified. Next, I split up the training and val datasets into training and validation sets using an 80/20 split. I then converted the labels to categorical variables and reshaped the data to be able to be inserted into the Keras API. For the traditional CNN, this was all the pre-processing I did, but for the LSTM and GRU models, I also added some additional data augmentation.

2.3. Augmentation

To properly give the CNN+LSTM and CNN+GRU architectures enough time-series data to generalize, I augmented the data to create more data points. I implemented maxpooling, added averaging and noise, and subsampled. I also included a flip, as Freer and Yang (2019) found that implementing flip among other methods improved classification accuracy by up to 14.0% [3]. After augmenting the data, we go from having 2115 trials in the training set to 7460 trials, a significant increase that is hoped to yield more impressive results from the LSTM and

GRU architectures.

3. Models

As mentioned before, the three models I constructed were a traditional CNN, a CNN+LSTM, and a CNN+GRU. Detailed model architecture can be found in 6.1, but I will give a brief description for each architecture below. Hyperparameter selection will be discussed in the Discussion section.

3.1. Convolutional Neural Network (CNN)

The CNN consists of 4 convolutional components, each with a 2-dimensional convolutional layer with an ELU activation, a 2-dimensional maxpooling layer, a batch normalization layer, and a dropout layer. For the output layer, I used a fully connected layer with a SoftMax activation to get the scores.

3.2. CNN with LSTM

The CNN+LSTM architecture consists of 4 convolutional components, each with a 2-dimensional convolutional layer with an ELU activation, a 2-dimensional maxpooling layer, a batch normalization layer, and a dropout layer, just like the traditional CNN. Next, I added a fully connected layer, then the LSTM layer. Lastly, I similarly use a fully connected layer with a SoftMax activation to get the scores.

3.3. CNN with GRU

The CNN+GRU architecture consists of 3 convolutional components instead of 4 to follow [2]. Each convolutional block is the same as the previous models' convolutional blocks. Next, I implement a fully connected layer followed by the GRU layer. Lastly, I once again use a fully connected layer with a SoftMax activation to get the scores.

4. Results

After testing every single architecture, I found that I consistently got the best accuracy with the traditional CNN architecture. Therefore, to answer the questions regarding single subject accuracy vs. all subject accuracy, I chose to focus on using the CNN to make my evaluations.

4.1. Single Subject Accuracy

After training the CNN on each subject, I found that the testing accuracies on each subject's own testing set were all around 30-50%. When testing on the entire testing set, the accuracies tended to be quite similar, with most of them dropping a bit.

4.2. Accuracy Across All Subjects

The performance of the CNN when training on the entire dataset, meaning all subjects, achieved a peak accuracy of 70.6%. For the CNN+LSTM architecture, it achieved an accuracy of 66.6%. For the CNN+GRU architecture, it achieved an accuracy of 60.9%.

4.3. Accuracy Across Time

After separately training the CNN across time segments 0:50, 0:100, 0:150, …, 0:1000, the performance peaked in accuracy at time point 500. At the small time points, the testing accuracy was very low, with accuracies below 60%, but then at time point 250, we achieve a 66% accuracy. After time point 500 however, the accuracy begins to taper off, going back down to a low of 61%.

5. Discussion

5.1. Hyperparameter Selection

First, the shared hyperparameters for all models I experimented with are: time (for preprocessing), filter number, kernel size, pool size, activation, dropout rate, learning rate, epochs, and batch size. As mentioned before, a time of 500 for the CNNs yielded the best results, but for the LSTM and GRU models, I chose a time of 800 to include as much time-series data as possible so the RNN layers could capture more temporal dependencies. For the other hyperparameters, I incorporated values that minimized overfitting and improved generalizability. I found that, for example, some choices such as having a dropout of 0.2 or an activation of Relu would lead to overfitting, so I chose hyperparameters that led to minimal overfitting, like dropout of 0.5 and activation of Elu instead.

For the CNN+LSTM and CNN+GRU models, additional hyperparameters for data augmentation were chosen to give the models an optimal amount of datapoints. Additionally, due to the GRU and LSTM layers, these models also had a recurrent dropout hyperparameter, which I set to 0.1 to prevent overfitting. A detailed briefing of the hyperparameters used can be found in 6.2.

5.2. Comparison Between Models

All of the architectures followed the same layout for their convolutional blocks. This is because I found success using the traditional CNN, so I hypothesized that utilizing the same architecture again for the CNN's with LSTM and GRU would yield good results.

However, I clearly was mistaken, as the traditional CNN yielded the best results. This leads us to believe that the LSTM layer might be too complex for the limited amount of data at our disposal. Adding the LSTM layer to the base CNN model gave a drastically longer training time and reduced performance. The increase in training time is likely due to the increased interconnectivity due to the LSTM layer. It is notable that the validation accuracy was higher than the training accuracy, indicative of the data augmentation not being up to task for this architecture, as the validation set is too similar to the training set.

We address this problem in the GRU model, where I incorporated a flip, which then causes the model to not have these issues. However, this does not mean that the model had better performance: it in fact performed the worst out of all the models. Similar to the LSTM, the GRU layer might be too complex for the limited data we have even after subsampling. If we were given more trials to work with, then I would assume that the CNN+LSTM and CNN+GRU architectures would have better performance. But for the data we have, I would have to conclude based on my results that traditional CNNs are the most optimal architecture for classification.

5.3. Accuracy Across Single Subjects vs. All Subjects

As mentioned before, the testing accuracies on both each subject's testing set and the whole testing set after training the CNN on each subject individually were not great. This makes sense, as training the CNN on each subject individually would not generalize to the whole dataset and would also not yield good results overall due to the very limited amount of data per subject.

On the other hand, we obviously prefer to train the network on the entire dataset so that it is able to generalize to the unseen data better, as we have a significantly greater amount of data points.

5.4 Accuracy Over Time

As predicted, the accuracy of the CNN's classifications peaked at time point 500, meaning that my hypothesis that time point 500 would be the optimal hyperparameter choice was correct. Up until time point 200, the accuracy was below 60%: this makes sense as these were the datasets with the least amount of dimensions, leading to the model not being able to discern the simpler data. After the 250th time point, the model seems to hover around 65-67%, as we now have enough data to make reasonable classifications, and each time point has the signal making distinct shapes. The tapering of the accuracy back down to the low 60s after time point 500 is indicative of the fact that the signals themselves taper off and become very flat to the point that taking into account more time points actually hinders our predictions.

References

[1] Schirrmeister, R. T., Springenberg, J. T., Fiederer, L. D., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W., & Ball, T. (2017). Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping*, *38*(11), 5391–5420. https://doi.org/10.1002/hbm.23730

[2] Hussein, A., Djandji, M., Mahmoud, R. A., Dhaybi, M., & Hajj, H. (2020). Augmenting DL with adversarial training for robust prediction of epilepsy seizures. *ACM Transactions on Computing for Healthcare*, *1*(3), 1–18. https://doi.org/10.1145/3386580

[3] He, C., Liu, J., Zhu, Y., & Du, W. (2021). Data augmentation for deep neural networks model in EEG Classification Task: A Review. *Frontiers in Human Neuroscience*, *15*. https://doi.org/10.3389/fnhum.2021.765525

6. Appendix

6.1. Detailed Model Architecture

6.1.1 CNN

The CNN takes input shape (time, 1, 22) and has 4 convolutional blocks, each with a Conv2D layer, MaxPooling2D layer, Batch Normalization layer, and Dropout layer. The Conv2D layers use 25, 50, 100, and 200 filters with kernel size (8,8) and an Elu activation, the MaxPooling2D layers use pool size of (3,1), and the dropout layers use 0.5 dropout. After the convolutional blocks, we use a flatten layer then a dense layer with 4 units with a SoftMax activation.

6.1.2 CNN with LSTM

The CNN with LSTM uses the exact same 4 convolutional blocks as the traditional CNN. After these blocks, we add a flatten layer, then a dense layer with 40 units, then a reshape layer to reshape to (40,1), then the LSTM layer, which has 10 units, dropout of 0.4, and a recurrent dropout of 0.2. We finally add the same output layers as the CNN.

6.1.3 CNN with GRU

The CNN with GRU uses 3 convolutional blocks, all the same as the traditional CNN. We then add the same layers as the CNN with LSTM but instead of an LSTM layer, we add a GRU layer with 20 units and the same dropout and recurrent dropout as the LSTM. We then simply add a dense layer with a SoftMax activation.

6.2. Other Hyperparameters

Learning rate was 1e-3, models were trained on 100 epochs, and the batch size was 64. For the CNN, I used time of 500. For the CNN with LSTM and CNN with GRU, for data augmentation, I used sub_sample of 2, average of 2, and time of 800. All models used a Categorical Cross-entropy loss function and an Adam optimizer.

Now include the architectures.

Table 1. Architecture of CNN

Layer Type	Output Shape	Parameters
Conv2D	(None, 500, 1, 25)	35225
MaxPooling2D	(None, 167, 1, 25)	0
BatchNormalization	(None, 167, 1, 25)	100
Dropout	(None, 167, 1, 25)	0
Conv2D	(None, 167, 1, 50)	80050
MaxPooling2D	(None, 56, 1, 50)	0
BatchNormalization	(None, 56, 1, 50)	200
Dropout	(None, 56, 1, 50)	0
Conv2D	(None, 56, 1, 100)	320100
MaxPooling2D	(None, 19, 1, 100)	0
BatchNormalization	(None, 19, 1, 100)	400
Dropout	(None, 19, 1, 100)	0
Conv2D	(None, 19, 1, 200)	1280200
MaxPooling2D	(None, 7, 1, 200)	0
BatchNormalization	(None, 7, 1, 200)	800
Dropout	(None, 7, 1, 200)	0
Flatten	(None, 1400)	0
Dense	(None, 4)	5604
Total params	1722679	
Trainable params	1721929	
Non-trainable params	750	

Table 2. Architecture of CNN+LSTM

Layer Type	Output Shape	Parameters
Conv2D	(None, 400, 1, 25)	35225
MaxPooling2D	(None, 134, 1, 25)	0
BatchNormalization	(None, 134, 1, 25)	100
Dropout	(None, 134, 1, 25)	0
Conv2D	(None, 134, 1, 50)	80050
MaxPooling2D	(None, 45, 1, 50)	0
BatchNormalization	(None, 45, 1, 50)	200
Dropout	(None, 45, 1, 50)	0
Conv2D	(None, 45, 1, 100)	320100
MaxPooling2D	(None, 15, 1, 100)	0
BatchNormalization	(None, 15, 1, 100)	400
Dropout	(None, 15, 1, 100)	0
Conv2D	(None, 15, 1, 200)	1280200
MaxPooling2D	(None, 5, 1, 200)	0
BatchNormalization	(None, 5, 1, 200)	800
Dropout	(None, 5, 1, 200)	0
Flatten	(None, 1000)	0
Dense	(None, 40)	40040
Reshape	(None, 40, 1)	0
LSTM	(None, 10)	480
Flatten	(None, 10)	0
Dense	(None, 4)	5604
Total params	1757639	
Trainable params	1756889	
Non-trainable params	750	

Table 3. Architecture of CNN+GRU

Layer Type	Output Shape	Parameters
Conv2D	(None, 400, 1, 25)	35225
MaxPooling2D	(None, 134, 1, 25)	0
BatchNormalization	(None, 134, 1, 25)	100
Dropout	(None, 134, 1, 25)	0
Conv2D	(None, 134, 1, 50)	80050
MaxPooling2D	(None, 45, 1, 50)	0
BatchNormalization	(None, 45, 1, 50)	200
Dropout	(None, 45, 1, 50)	0
Conv2D	(None, 45, 1, 100)	320100
MaxPooling2D	(None, 15, 1, 100)	0
BatchNormalization	(None, 15, 1, 100)	400
Dropout	(None, 15, 1, 100)	0
Flatten	(None, 1500)	0
Dense	(None, 40)	60040
Reshape	(None, 40, 1)	0
GRU	(None, 20)	1380
Dense	(None, 4)	84
Total params	497579	
Trainable params	497229	
Non-trainable params	350	

Figure 1. Performance of CNN over time

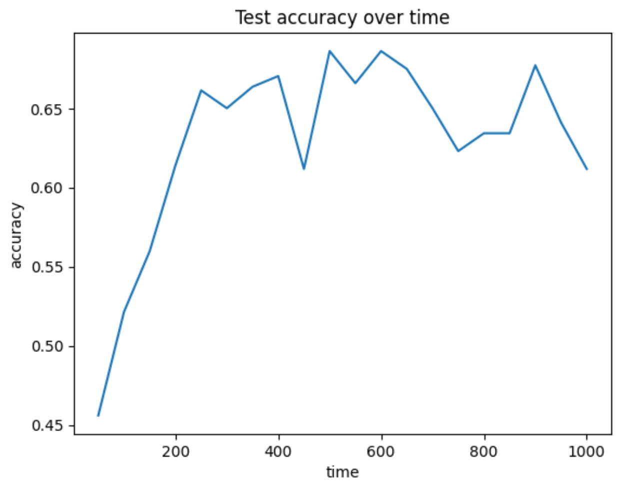

Table 4. Performance of all models on all subjects

Model	**Classification Accuracy**
CNN	70.65%
CNN with LSTM	66.59%
CNN with GRU	60.94%

Table 5. Performance of CNN training on each subject

	Subject Number								
	1	2	3	4	5	6	7	8	9
Accuracy on Subject	0.3800	0.3800	0.4400	0.4000	0.255	0.3469	0.5200	0.3600	0.3617
Accuracy on Entire Set	0.3454	0.3567	0.3928	0.4153	0.4086	0.3725	0.4312	0.3612	0.3905

Table 6. Performance of CNN over time

	Time Point								
	50	100	150	200	250	300	350	400	450
Classification Accuracy	0.4568	0.5214	0.5598	0.6140	0.6614	0.6501	0.6637	0.6704	0.6117
	500	550	600	650	700	750	800	850	900
Classification Accuracy	0.6862	0.6659	0.6862	0.6749	0.6501	0.6230	0.6343	0.6342	0.6772
	950	1000							
Classification Accuracy	0.6411	0.6117							