Experimentation on EGG Classification Using CNN and RNN

Jackson Zhou, Tianshu Xiao, Liyuan Zhao, Qingyuan Pan
UCLA

Abstract

This goal of the project is to classify the provided electroencephalograph (EEG) dataset[1] with 2,115 training/validation samples. The data will be classified into four different motor imaginary tasks (i.e.of the left hand, right hand, both feet, and tongue) using Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). For this task, we compare performance and accuracy of classification using CNN and RNNs. Along with multiple deep learning architectures, we also apply many data preprocessing techniques. In addition, we are adding Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) modeles to the neural network as a RNN function to show the different performances for these models. Throughout the testing, we found out that our model of CNN+LSTM with data subsampling is able to achieve the highest test accuracy of ***73.76%.***

1. Introduction

The base project provided was to explore datasets collected from electroencephalography (EEG). The dataset used to train the network is the BCI Competition 2008 data set A [1]. It consists of 22 EEG channels from 9 subjects performing 4 motor-imagery tasks, as mentioned in the abstract. There are a total of 2115 trials completed and each train has corresponding EEG data from 22 electrodes. These signals are recorded near non-invasive scalp electrodes and therefore have less accurate resolution compared to other approaches. Classification on these types of dataset is challenging because there are many noises in the dataset. Our approach is to apply deep learning techniques, including CNN and RNN to this dataset. To tackle the task of classification EEG dataset, both the CNN and RNN architectures have been compared and evaluated. We will further discuss details and results on implementing these networks.

1.1. Data Preprocessing

Throughout the testing, we trained and tested on different sets of data to make the network more robust to variations in data. To start with, the network was trained on data without data augmentation. We generated the training and validation indices using random splitting. We then converted the labels to categorical variables for multiclass classification. This was performed with all subjects. In the next procedure, the dataset was first pre-processed with trimming, max pooling and subsampling, and then followed with the previous processes.

Trimming is the process where we first cluster samples into their corresponding labels and examine their EEG signals in their respective channels. The EEG signals reveal distinguishable patterns roughly in the first 500 time steps. The signals in the later 500 time steps are noisier and do not exhibit clear patterns. Prior to data augmentation, the model overfits the training data. Therefore, we augment our data using maxpooling and subsampling. For maxpooling, we maxpool the data with trimmed 500 timeslots to 250 timeslots. We also use the trimmed data to get the average of the signal and add noise with Guassian distribution to it to get another 250 timeslots data. We also do data subsampling for every two timesteps and add noise to these data. By these steps, we can get 4 times more trials with the same 22 electrodes over 250 timesteps for us to train our models. The advantage is that we have more data to train, but the disadvantage is the validation accuracy might be higher because we broke down each sample into 4 samples, thus there will be many overlaps between the training set and validation set. Therefore the accuracy of the validation set is not reliable. We can see that we can get very high validation accuracy and lower test accuracy later in our results.

1.2. CNN

For this task, we built a Convolution Neural Network (CNN) which involved the use of 4 convolutional layers in total. It contained 4 layers of Conv2d neural network and 1 layer of fully connected layers. The techniques of Max Pooling, Batch Normalization and Dropout with $p = 0.5$ were applied in the Conv2d layers to improve the efficiency of the network. Kernel size was regulated to optimize the network performance. We set epochs with 200 and the early stopping function for the entire iteration. Throughout the testing, the Adam optimizer was utilized.

1.3. CNN+LSTM

Long short-term memory (LSTM) is a special kind of RNN which is capable of learning long-term dependencies. LSTM are designed to dodge long-term dependency problems as they are capable of remembering information for longer periods of time[2].

The CNN structure remains the same. The LSTM module is connected along with CNN layers. We introduced CuDNNLSTM as the LSTM layer after the fully connected layers. Adding CNN before LSTM is considered as a preprocessing of data before being input to the LSTM.

Similarly, The CNN+LSTM networks were trained and tested on datasets with the same preprocessing step as CNN network. The datasets were distinguished between without data augmentation and with data subsampling to make the network more robust to variations in data and further improve the classification accuracy and model performances.

1.4. CNN+GRU

The Gated Recurrent Unit (GRU) is also a newer generation of RNN. Unlike LSTM, GRU has only two gates, a reset gate and an update gate and they lack output gate. GRU's got itself free of the cell state and instead uses the hidden state to transfer information[3].

The GRU module is connected after CNN layers. The architecture flowchart of CNN+GRU should be similar to CNN+LSTM, except for using CuDNNGRUas GRU layer at the end.

Similarly, The CNN+GRU networks were trained and tested on datasets with the same preprocessing step as CNN and CNN+LSTM.

1.5. RNN (GRU based)

For this experiment, we did not combine CNN structures as previous parts. Instead, we tested on 2 layer GRU which is a completely RNN model, using the preprocessed data[5].

2. Overall Results and Discussion

Dissemination of results is an important component of any project. In this section, we summarize the results we obtained and present an analysis. We tested in a total of 7 cases. Figure 1 shows the summarized results.

We tested through CNN, RNN (GRU base) and also combined CNN+LSTM and CNN+GRU to perform better classification accuracy. Finally, CNN+LSTM with data subsampling achieved the highest accuracy with 73.76%. We observed that CNN and hybrid models perform better than RNN for the given EEG data. Additionally, these networks could be further enhanced by using data pre-processing techniques, such as trimming and data sampling. By comparison, we realized that the data with subsampling can attain a higher classification accuracy than the data without preprocessing.

In the spirit of discovery, we evaluated the CNN, RNN and hybrid architectures with and without data sampling. We also found out and analyzed the confusion matrices for each test case. The classification accuracies indicate that the CNN, CNN +LSTM and CNN+GRU model with data subsampling perform better at classification compared to the RNN (GRU base) model as shown in Figure 2.

Dataset and Models	Test Accuracy
CNN w/o data augmentation	0.7065462470054626
CNN with data augmentation	0.7353273034095764
CNN+LSTM w/o data augmentation	0.6297968626022339
CNN+LSTM with data augmentation	0.73758465051651
CNN+GRU w/o data augmentation	0.6252821683883667
CNN+GRU with data augmentation	0.705981969833374
RNN with data augmentation (GRU)	0.3504514694213867

Figure 1. results for all 7 test cases

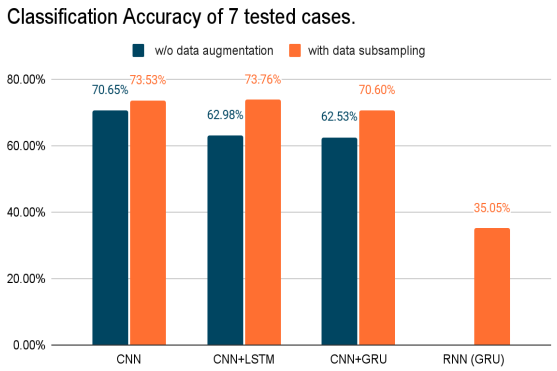


Figure 2. bar charts for all 7 test accuracies

2.1. CNN results

For the CNN model alone, it already achieved a decent test accuracy of 70.65% without any data augmentation and using 4 layers with 2d convolution neural network along with techniques like batchnorm and dropout. However, we were able to obtain a better test accuracy result of 73.53% with data subsampling to randomize the data to reduce the reliance of precise positioning within feature maps that are produced and improve learning result by preventing overfit of the model through toning all the hyper parameters.

2.2. CNN+LSTM results

For the CNN+LSTM model, we kept the same CNN structure and added a LSTM layer at the end for detection using the extracted features which are fed into LSTM after data was processed by CNN. Without any data augmentation, we were only able to achieve a test accuracy of 62.98%. However, we were able to achieve the best test accuracy for our entire experiment by using subsampling for this hybrid model, which is 73.76%. With this combination, the CNN model can find both effective features from the data along with the interdependence of data after being fed into the LSTM.

2.3. CNN+GRU Results

For the CNN+GRU model, we kept the same CNN structure. Similarly to the CNN+LSTM model, we added a Gated Recurrent Unit layer at the end after data was processed by CNN. With any data augmentation, we were able to obtain a test accuracy of 62.53%. With subsampling, we were able to improve the test accuracy to 70.60%. Even though the result we have from this hybrid model is a bit less than CNN+LSTM, the results are quite similar because GRU is similar to LSTM with one less gate. GRU has two gates while LSTM has 3 gates and they are both dedicated to process time sequence data. Due to the gate difference, GRU uses less memory and is faster than LSTM. However, GRU is less accurate than LSTM when using datasets with longer sequences. Therefore, CNN+GRU with data subsampling resulted in a lower test accuracy compared to CNN+LSTM.

2.4. GRU Results

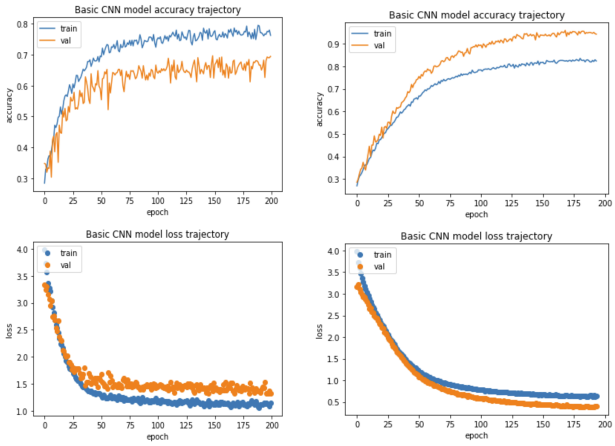
Lastly, we tested 2 layers GRU based RNN model on its own using data augmentation method (preprocessed data) without combining it with a CNN model. The best test accuracy we observed with data augmentation was 35.0%, which is not as appealing compared to other models we have tested. The reasons were because GRU alone has less gesture compatibility when compared to CNN and it did not adaptively learn spatial hierarchies of features, which caused the model to miss lots of important features during the learning process. Overall, GRU did not perform too well as it is predicted because of its disadvantages.

2.5. Conclusions

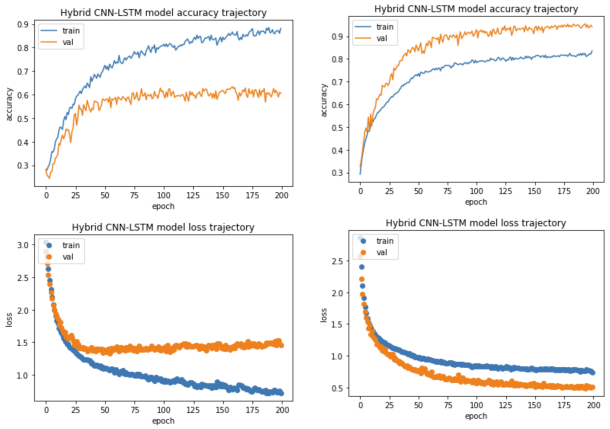
In this project, seven models have been fitted. Three basic neutral networks, CNN with two different dataset, with data augmentation and without data augmentation. and RNN. And four neural networks with combination layers, CNN + LSTM and CNN + GRU, aslo fitted with original data and augmented data. Finally, CNN+LSTM is the best performance model with the highest accuracy, 73.76%.

Algorithm Performance

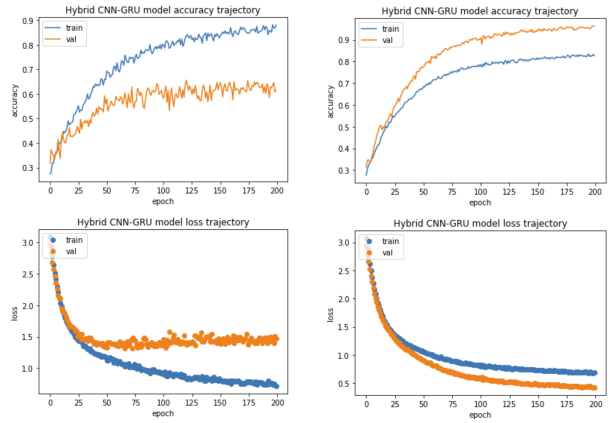
Accuracy and Loss for Train/validation over epoch



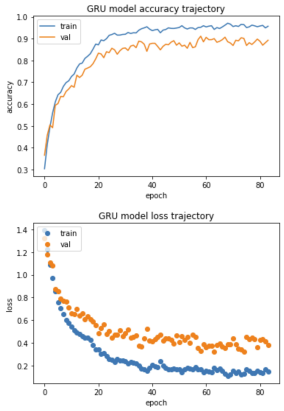
CNN without data augmentation (left) and with data subsampling (right)



CNN+LSTM without data augmentation (left) and with data subsampling (right)



CNN+GRU without data augmentation (left) and with data subsampling (right)



RNN (GRU base) with data subsampling

The classification reports with threshold=0.5 for each label

Model with Accuracy	Label	Precision	Recall	F1-score
CNN+GRU **Accuracy=0.625**	left	0.66	0.51	0.58
	right	0.54	0.72	0.62
	foot	0.7	0.51	0.59
	tongue	0.75	0.56	0.64
CNN+GRU with data preprocessing **Accuracy=0.625**	left	0.78	0.66	0.72
	right	0.71	0.7	0.7
	foot	0.69	0.62	0.66
	tongue	0.78	0.73	0.72
CNN+LSTM **Accuracy=0.630**	left	0.6	0.69	0.64
	right	0.72	0.54	0.61
	foot	0.7	0.5	0.58
	tongue	0.63	0.66	0.65
CNN+LSTM with data preprocessing **Accuracy=0.738**	left	0.85	0.69	0.76
	right	0.74	0.75	0.75
	foot	0.71	0.62	0.67
	tongue	0.74	0.74	0.74
CNN **Accuracy=0.707**	left	0.81	0.66	0.73
	right	0.84	0.49	0.62
	foot	0.66	0.58	0.62
	tongue	0.72	0.71	0.71
CNN with data preprocessing **Accuracy=0.735**	left	0.78	0.72	0.75
	right	0.76	0.78	0.77
	foot	0.7	0.51	0.59
	tongue	0.79	0.72	0.75
GRU **Accuracy=0.350**	left	0.37	0.3	0.33
	right	0.3	0.39	0.34
	foot	0.31	0.31	0.31
	tongue	0.47	0.38	0.42

Model Architecture

CNN: The architecture of CNN model is 4 layers of Conv2D with 4 different filter sizes 16, 32, 64, 128. The kernel sizes are all (10,1), paddings all use zero padding, the activation function we used are all ELU, and we added L2 regularizer with lambda= 0.01 for all these convolution layers. Then the MaxPooling2D layer with pool size (10,1) and zero padding, the BatchNormalization layer , and the Dropout layer with p=0.5 are added to each convolution layer. At the end of the network, the extracted features are flattened and output the FC layer with softmax activation and L2 regularizer with lambda= 0.01. The total parameters are 163974, and we use Adam optimizer with learning rate=1e-4 and batch_size=32.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 250, 1, 16)	3536
max_pooling2d (MaxPooling2D)	(None, 84, 1, 16)	0
batch_normalization (BatchNo	(None, 84, 1, 16)	64
dropout (Dropout)	(None, 84, 1, 16)	0
conv2d_1 (Conv2D)	(None, 84, 1, 32)	5152
max_pooling2d_1 (MaxPooling2	(None, 28, 1, 32)	0
batch_normalization_1 (Batch	(None, 28, 1, 32)	128
dropout_1 (Dropout)	(None, 28, 1, 32)	0
conv2d_2 (Conv2D)	(None, 28, 1, 64)	20544
max_pooling2d_2 (MaxPooling2	(None, 10, 1, 64)	0
batch_normalization_2 (Batch	(None, 10, 1, 64)	256
dropout_2 (Dropout)	(None, 10, 1, 64)	0
conv2d_3 (Conv2D)	(None, 10, 1, 128)	82048
max_pooling2d_3 (MaxPooling2	(None, 4, 1, 128)	0
batch_normalization_3 (Batch	(None, 4, 1, 128)	512
dropout_3 (Dropout)	(None, 4, 1, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 4)	2052

Total params: 114,292
Trainable params: 113,812
Non-trainable params: 480

CNN+LSTM: The architecture uses the same 4 convolution blocks in CNN above, and then it added a flatten layer to the output of the last convolution block, and added an FC layer with 100 units. After reshaping the output of the FC layer to (100,1) shape, we added the CuDNNLSTM layer with 10 units. The reason why we use CuDNNLSTM instead of LSTM is that we can get over hundreds times faster speed to train the network. At the end of the network, the extracted features are flattened and output the FC layer with softmax activation and L2 regularizer with lambda= 0.01. The total parameters are 164104, and we use Adam optimizer with learning rate=3e-4 and batch_size=64.

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 250, 1, 16)	3536
max_pooling2d_36 (MaxPooling	(None, 84, 1, 16)	0
batch_normalization_36 (Batc	(None, 84, 1, 16)	64
dropout_36 (Dropout)	(None, 84, 1, 16)	0
conv2d_37 (Conv2D)	(None, 84, 1, 32)	5152
max_pooling2d_37 (MaxPooling	(None, 28, 1, 32)	0
batch_normalization_37 (Batc	(None, 28, 1, 32)	128
dropout_37 (Dropout)	(None, 28, 1, 32)	0
conv2d_38 (Conv2D)	(None, 28, 1, 64)	20544
max_pooling2d_38 (MaxPooling	(None, 10, 1, 64)	0
batch_normalization_38 (Batc	(None, 10, 1, 64)	256
dropout_38 (Dropout)	(None, 10, 1, 64)	0
conv2d_39 (Conv2D)	(None, 10, 1, 128)	82048
max_pooling2d_39 (MaxPooling	(None, 4, 1, 128)	0
batch_normalization_39 (Batc	(None, 4, 1, 128)	512
dropout_39 (Dropout)	(None, 4, 1, 128)	0
flatten_9 (Flatten)	(None, 512)	0
dense_15 (Dense)	(None, 100)	51300
reshape_9 (Reshape)	(None, 100, 1)	0
cu_dnnlstm_6 (CuDNNLSTM)	(None, 10)	520
dense_16 (Dense)	(None, 4)	44

Total params: 164,104
Trainable params: 163,624
Non-trainable params: 480

CNN+GRU: The overall architecture of CNN+GRU is the same as the CNN+LSTM model. The only difference is that we use CuDNNGRU instead of CuDNNLSTM, the reason is that we can train faster with GPU for this model. The total parameters are 163974, and we use Adam optimizer with learning rate=1e-4 and batch_size=32.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 250, 1, 16)	3536
max_pooling2d (MaxPooling2D)	(None, 84, 1, 16)	0
batch_normalization (BatchNo	(None, 84, 1, 16)	64
dropout (Dropout)	(None, 84, 1, 16)	0
conv2d_1 (Conv2D)	(None, 84, 1, 32)	5152
max_pooling2d_1 (MaxPooling2	(None, 28, 1, 32)	0
batch_normalization_1 (Batch	(None, 28, 1, 32)	128
dropout_1 (Dropout)	(None, 28, 1, 32)	0
conv2d_2 (Conv2D)	(None, 28, 1, 64)	20544
max_pooling2d_2 (MaxPooling2	(None, 10, 1, 64)	0
batch_normalization_2 (Batch	(None, 10, 1, 64)	256
dropout_2 (Dropout)	(None, 10, 1, 64)	0
conv2d_3 (Conv2D)	(None, 10, 1, 128)	82048
max_pooling2d_3 (MaxPooling2	(None, 4, 1, 128)	0
batch_normalization_3 (Batch	(None, 4, 1, 128)	512
dropout_3 (Dropout)	(None, 4, 1, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 100)	51300
reshape (Reshape)	(None, 100, 1)	0
cu_dnngru (CuDNNGRU)	(None, 10)	390
dense_1 (Dense)	(None, 4)	44

Total params: 163,974
Trainable params: 163,494
Non-trainable params: 480

GRU: This is the model that contains 3 layers of CuDNNGRU. We first flatten the input and then we added

3 GRU blocks to the model. The first block is FC layer with 1100 units, and reshape to (1100,1) for the GRU layer with 256 units. And the next block is FC layer with 220 units, and reshape to (220,1) for the GRU layer with 128 units. The last GRU block is FC layer with 44 units, and reshape to (44,1) for the GRU layer with 64 units. The total parameters are 6375656, and we use Adam optimizer with learning rate=1e-3 and batch_size=32.

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 5500)	0
dense_1 (Dense)	(None, 1100)	6051100
reshape_1 (Reshape)	(None, 1100, 1)	0
cu_dnngru (CuDNNGRU)	(None, 256)	198912
dropout (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 220)	56540
reshape_2 (Reshape)	(None, 220, 1)	0
cu_dnngru_1 (CuDNNGRU)	(None, 128)	50304
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 44)	5676
reshape_3 (Reshape)	(None, 44, 1)	0
cu_dnngru_2 (CuDNNGRU)	(None, 64)	12864
dropout_2 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 4)	260

Total params: 6,375,656
Trainable params: 6,375,656
Non-trainable params: 0

For all of these models, we use 200 epochs to train to make sure the model is converged, and we added L2 regularization to each output block to avoid overfitting. We also use EarlyStopping with monitor='val_accuracy' and patience=25 to stop training when the validation accuracy has stopped improving in 25 epochs.

References

[1] C. Brunner, R. Leeb, G. Muller-Putz, A. Schl ¨ ogl, and ¨ G. Pfurtscheller. Bci competition 2008–graz data set a. Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology, 16, 2008.

[2] Understanding LSTM Networks -- colah's blog, 2015.

[3] Introduction to CNN, LSTM, GRU, OBJECT RECOGNITION & DATASETS. (2018). Introduction to CNN and GRU.

[4] Dar, Muhammad N., Muhammad U. Akram, Sajid G. Khawaja, and Amit N. Pujari. 2020. "CNN and LSTM-Based Emotion Charting Using Physiological Signals" Sensors 20, no. 16: 4551. https://doi.org/10.3390/s20164551

[5] Structure of two layers GRU. | download scientific diagram. (n.d.), 2022.