# ECE C247 Final Project Report


Xianxing Jiang
UID: 604958018
xjiang@g.ucla.edu

Yifan Xu
UID: 905432141
yifanxu@g.ucla.edu

Weibo Hong
UID: 405427378
weiboh1@g.ucla.edu

Yong Huang (C147)
UID: 904950577
yhuang137@g.ucla.edu



## Abstract

*Brain-Computer Interaction (BCI) competition provides data of electroencephalograph with four different motor imaginary tasks, including movement of left hand, right hand, both feet and tongue. Our purpose for this project is to analysis and classify the four sets of data using Convolutional Neural Network(CNN) [1] and Recurrent Neural Network(RNN). After classification, we will further analyse the pros and cons of different model architectures with hyper-parameters, aims at finding the best settings. What's more, we will further investigate the influence of training samples on different time periods, and do comparison.*


## 1. Introduction

In this project, we trained data sets on two kinds of network: Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). For each of them, we then built two structures. Then, we trained the data on both of them and made comparisons.

### 1.1. Architecture of CNN

After analysis of the training data, we noticed that the second dimension was referred as channels and the third dimension was refer as time bins. This implied two options: to do convolution on one dimension or on both two dimensions. Thus, we built two kinds of CNN and test them one by one. The first CNN architecture was shown in table 3. There are four 2D convolution layers and connected by a ELU after. We also added Batch norm to prevent overfitting and used Max Pooling after each convolution to reduce dimension size. Dropout was only added on layers three and four. In the end, we use Softmax as an activation function and Adam as our optimizer policy. The second CNN was similar to the previous one but only applied kernel operation on one dimension each time. The first three 2D convolution layers were designed to do convolution on time-bin dimension while the last 2D convolution layer was designed to do convolution on channel dimension.

### 1.2. Architecture of RNN

We built three different RNN networks in total. The first one was a three layers SimpleRNN network, but the result was far from out expectation. We then switched to use LSTM layer as a replacement. The RNN-A network that we shown in table 5, was a three Bidirectional LSTM layers network, each connected with Dropout setting of 0.5 probability. For activation function, we still used ELU like in CNNs. In the end, we kept Softmax to shrink classification into 4 and yield prediction basing on that. As to the RNN-B network, we tried to combine CNN and LSTM together. We kept all structures and settings of CNN-A, but fed the result to RNN-A then. Thus, we got a three Conv2D layer three LSTM layers network. The result was also fed to a output layers using Softmax and yield a class prediction of 0-4.

## 2. Results

first of all, We trained all four neural networks individually to evaluate the classification accuracy as a function of time and it was shown by figure 1 2 3 4. Through the graph, we pick the time bin of highest accuracy from CNN-A and RNN-A and use it to do subjects training. Table 1 is generated by CNN-A and Table 2 is generated by RNN-A.

### 2.1. Evaluate the accuracy as a function of time

For CNN-A, as shown in figure 1, the training accuracy increases as we increases the time t from 200 to 1000. The highest testing accuracy is achieved when t=400, which is 67.269%. After that, the testing accuracy starts to decrease. When t equals to 1000, the testing accuracy is only 60.045%. For RNN-A, as shown in figure 3, the training accuracy also increases as we increases the time t. This time, the highest testing accuracy appears when t=700, which is 65.688%. We also achieves highest testing accuracy at t=600 for both CNN-B and RNN-B. It seems that the testing accuracy does not have a high volatility corresponding to time, but all of our model achieved highest accuracy when t is in range (400, 700).

### 2.2. Train on each subject and test across all subjects

First, we train CNN-A and CNN-B across all subjects and test them across all subjects. We record the highest testing accuracy: for CNN-A is 66.59%, and for RNN-A iS 60.05%.

Then we trained them only on one subject but still test across all subjects. For CNN-A, as shown in table 1, the test accuracy of the models that only trained with one subject data are all below 40%, while, the training accuracy varies a lot. The highest testing accuracy is 37.47% when we using subject 4. For RNN-A, the training accuracy for all the subjects are 100%, which implies an over-fitting. However, even all testing accuracy are higher than 40%, they are still far lower than the accuracy we achieved by training on all subjects. The highest testing accuracy is 48.76% , when we using subject 4.

## 3. Discussion

In this part, we will talk about what we found in this project and how we reflect on the results we got. We will make comparisons on two CNN and RNN designs. We will also talk about how we decide the architectures and how we search the best hyper-parameters.

### 3.1. Comparison of two kinds of CNN

For this project, we have two kind of CNN design and the only difference between them are the size of kernels. By observing the results, we can see that both design have a test accuracy over 65%. CNN-A's highest accuracy is 67.269% and CNN-B's highest accuracy is 65.688%, which means they are very close. From the figure 1 and 2, we can also noticed that the accuracy difference at each time bins are very small. This is acceptable. Thus, we can conclude that this two CNN design have similar results. Besides that, we can also conclude that do convolution on one dimension each time and do convolution on two dimensions together will not affect result very much.

### 3.2. Comparison of two kinds of RNN

Since the EEG data sets were collected from 22 electrodes over 1000 time-bins, they are, for sure, highly time related and dependent. Therefore, using RNN instead of CNN should yield better performances. Our initial idea was using a three SimpleRNN layers network. However, after several unsuccessful parameters tuning trials, we decided to introduce the Long Short Term Memory (LSTM) layer instead, which was designed to "remember" past data in memory for long time period.

The test result shows that we achieved similar accuracy on the test data set by both CNNs and RNN-A. Therefore, we make a another decision to try connecting CNN and RNN together: a new network design of 3 Conv2D layers plus 3 bidirectional LSTM layers. Although this network took more time to be trained, from figure 4 we can find out that it achieved better and stable performances than all three previous networks. What's more, we noticed that the training accuracy of RNN-B was only around 75%, which seems to imply a upside potential on the accuracy, if we have more training data to train or set larger epochs during the training stage.

### 3.3. How to pick hyper parameter and architecture

In order to check the difference between "do convolution on one dimension each time" and "do convolution on two dimensions together", we set CNN-A kernel size as (3, 3) and CNN-B as (1, 10). Beside that, during training process, we found the models were easy to be over-fitting. Therefore, we added batch norm and dropout to prevent. We also found that introducing batch norm and dropout together helps increase the accuracy. Finally, we choose ELU as our activation instead of ReLU because The learning rate is faster in ELU than ReLU [2]. As to the RNN part, we used GridSearchCV and find out the best regularizer to be L2 normal of 0.05.

### 3.4. Comparison of time bin

We made a for-loop for tuning time-bins(t) from 200 to 1000 and set the step to be 1000. For all four networks, we found the training accuracy were constantly increasing as expected. However, the testing accuracy achieved peaks when t=400/600/700, but dropped down while we kept increase t from 600 to 1000. This indicates that the model starts to be over-fitting. We assume, from this finding, that the features of data were mainly accumulated in range of t=(0-700). It is wise to focus on this part of the data for training our model to yield the best performance.

### 3.5. Comparison of training one subject and training all subjects

We compared the accuracy of the model trained on one subject with the one trained on all subjects. The result is showing as Table 1 and Table 2. Apparently, the accuracy of the former are extremely low compared to the latter, no matter CNN or RNN. This can be easily explained since data of every single subject we have is only around 200. The model could not be well-trained by that amount of data. Also, from Table 2, we can find that even if we over-fitted the model, the test accuracy are still very bad. This implies that features of one subject data cannot represent the whole. Therefore, we conclude that training on one subject is meaningless, if we only have limited amount of data.

## References


[1] S. Roy, I. Kiral-Kornek, and S. Harrer. Chrononet. *A deep recurrent neural network for abnormal eeg identification.* CoRR, abs/1802.00308, 2018.

[2] Dabal Pedamonti. ,*Comparison of non-linear activation functions for deep neural networks on MNIST classification task.* Department of Computer Science University of Edinburgh, Apr. 8, 2018, https://arxiv.org/pdf/1804.02763.pdf

Table 1. Accuracy of Different Subjects with CNN

| CNN_A: Train One Test All | | |
|---|---|---|
| Subject | train accuracy | test accuracy |
| 0 | 83.97% | 32.51% |
| 1 | 58.47% | 27.77% |
| 2 | 49.58% | 28.22% |
| 3 | 91.45% | 31.83% |
| 4 | 37.02% | 28.44% |
| 5 | 100.00% | 37.47% |
| 6 | 60.08% | 31.83% |
| 7 | 64.66% | 24.15% |
| 8 | 49.78% | 30.47% |
| all | 87.28% | 66.59% |

Table 2. Accuracy of Different Subjects with RNN

| RNN_A: Train One Test All | | |
|---|---|---|
| Subject | train accuracy | test accuracy |
| 0 | 100.0% | 46.95% |
| 1 | 100.0% | 43.79% |
| 2 | 100.0% | 44.02% |
| 3 | 100.0% | 43.12% |
| 4 | 100.0% | 48.76% |
| 5 | 100.0% | 47.86% |
| 6 | 100.0% | 44.02% |
| 7 | 100.0% | 43.34% |
| 8 | 100.0% | 45.37% |
| all | 98.12% | 60.05% |

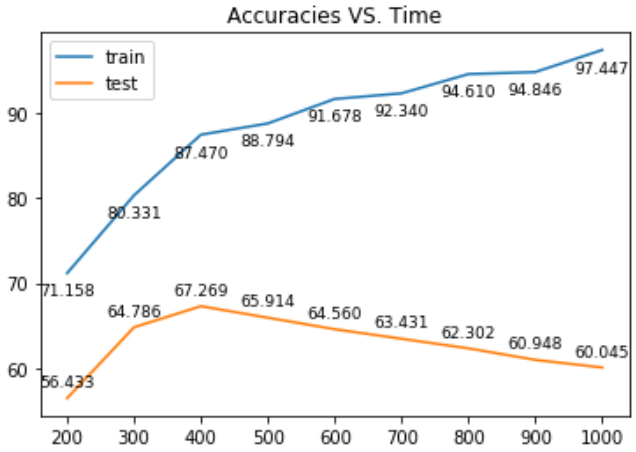


Figure 1. Accuracy of CNN_A

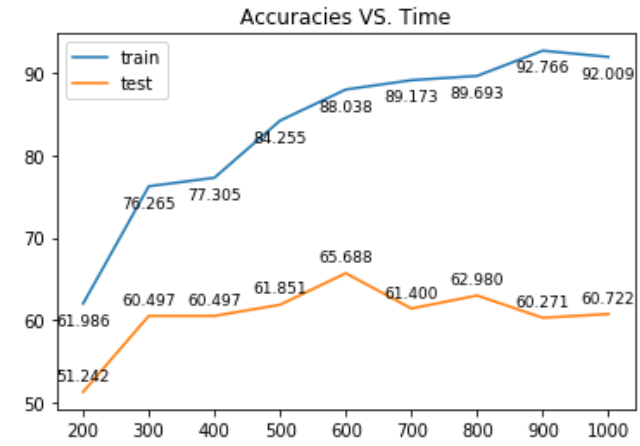


Figure 2. Accuracy of CNN_B

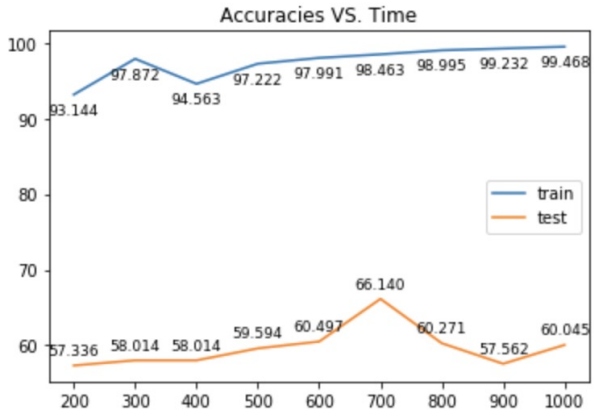


Figure 3. Accuracy of RNN_A

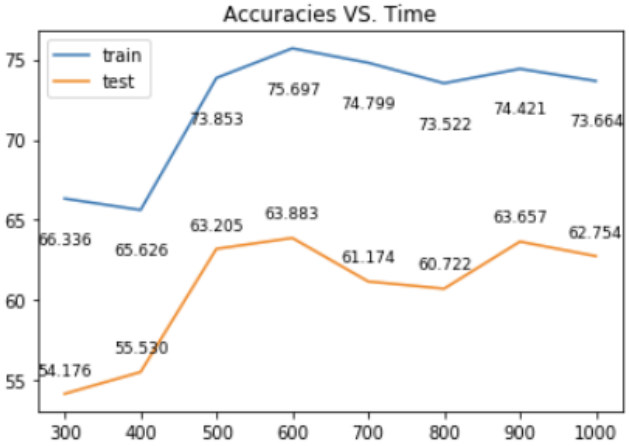


Figure 4. Accuracy of RNN_B

Table 3. Architecture of CNN_A

| CNN | | |
|---|---|---|
| | Kernel size | Output size |
| INPUT | | [22 x 1000 x 1] |
| CONV2D | [3 x 3 x 64] | [20 x 998 x 64] |
| ELU | | [20 x 998 x 64] |
| Batchnorm | | [20 x 998 x 64] |
| MaxPool | [1 x 2] | [20 x 499 x 64] |
| CONV2D | [3 x 3 x 64] | [18 x 497 x 64] |
| ELU | | [18 x 497 x 64] |
| Batchnorm | | [18 x 497 x 64] |
| MaxPool | [1 x 2] | [18 x 248 x 64] |
| CONV2D | [3 x 3 x 32] | [16 x 246 x 32] |
| ELU | | [16 x 246 x 32] |
| Batchnorm | | [16 x 246 x 32] |
| MaxPool | [1 x 2] | [16 x 123 x 32] |
| Dropout | | [16 x 123 x 32] |
| CONV2D | [3 x 3 x 16] | [14 x 121 x 16] |
| ELU | | [14 x 121 x 16] |
| Batchnorm | | [14 x 121 x 16] |
| MaxPool | [1 x 2] | [14 x 60 x 16] |
| Dropout | | [14 x 60 x 16] |
| Flatten | | [13440] |
| Softmax | [13440 x 4] | [4] |

Table 4. Architecture of CNN_B

| CNN | | |
|---|---|---|
| | Kernel size | Output size |
| INPUT | | [22 x 1000 x 1] |
| CONV2D | [1 x 10 x 64] | [22 x 991 x 64] |
| ELU | | [22 x 991 x 64] |
| Batchnorm | | [22 x 991 x 64] |
| MaxPool | [1 x 2] | [22 x 495 x 64] |
| CONV2D | [1 x 10 x 64] | [22 x 486 x 64] |
| ELU | | [22 x 486 x 64] |
| Batchnorm | | [22 x 486 x 64] |
| MaxPool | [1 x 2] | [22 x 243 x 64] |
| CONV2D | [1 x 10 x 32] | [22 x 234 x 32] |
| ELU | | [22 x 234 x 32] |
| Batchnorm | | [22 x 234 x 32] |
| MaxPool | [1 x 2] | [22 x 117 x 32] |
| Dropout | | [22 x 117 x 32] |
| CONV2D | [10 x 1 x 16] | [13 x 117 x 16] |
| ELU | | [13 x 117 x 16] |
| Batchnorm | | [13 x 117 x 16] |
| MaxPool | [1 x 2] | [13 x 58 x 16] |
| Dropout | | [13 x 58 x 16] |
| Flatten | | [12064] |
| Softmax | [12064 x 4] | [4] |

Table 5. Architecture of RNN_A

| LSTM(RNN) | | |
|---|---|---|
| | Kernel size | Output size |
| INPUT | | [22 x 1000 ] |
| PERMUTE | | [1000 x 22 ] |
| Bidirectional LSTM | | [1000 x 64] |
| Dropout | (0.5) | [1000 x 64] |
| Bidirectional LSTM | | [1000 x 128] |
| Dropout | (0.5) | [1000 x 128] |
| Bidirectional LSTM | | [1000 x 256] |
| Dropout | (0.5) | [1000 x 256] |
| Flatten | | [25600] |
| Dense | | [4] |

Table 6. Architecture of RNN_B

| CNN+LSTM(CRNN) | | |
|---|---|---|
| | Kernel size | Output size |
| INPUT | | [22 x 1000 x 1] |
| CONV2D | [3 x 3 x 64] | [20 x 998 x 64] |
| ELU | | [20 x 998 x 64] |
| Batchnorm | | [20 x 998 x 64] |
| MaxPool | [1 x 2] | [20 x 499 x 64] |
| Dropout | (0.4) | [20 x 499 x 64] |
| CONV2D | [3 x 3 x 64] | [18 x 497 x 64] |
| ELU | | [18 x 497 x 64] |
| Batchnorm | | [18 x 497 x 64] |
| MaxPool | [1 x 2] | [18 x 248 x 64] |
| Dropout | (0.4) | [18 x 248 x 64] |
| CONV2D | [3 x 3 x 32] | [16 x 246 x 32] |
| ELU | | [16 x 246 x 32] |
| Batchnorm | | [16 x 246 x 32] |
| MaxPool | [1 x 2] | [16 x 123 x 32] |
| Dropout | (0.4) | [16 x 123 x 32] |
| CONV2D | [3 x 3 x 16] | [14 x 121 x 16] |
| ELU | | [14 x 121 x 16] |
| Batchnorm | | [14 x 121 x 16] |
| MaxPool | [1 x 2] | [14 x 60 x 16] |
| Dropout | (0.4) | [14 x 60 x 16] |
| Permute | | [60 x 16 x 14] |
| Time Distributed Flatten | | [60 x 224] |
| Bidirection LSTM | | [60 x 128] |
| Dropout | (0.5) | [60 x 128] |
| Bidirection LSTM | | [60 x 64] |
| Dropout | (0.5) | [60 x 64] |
| Bidirection LSTM | | [60 x 32] |
| Dropout | (0.5) | [60 x 32] |
| Flatten | | [1920] |
| ELU | | [128] |
| Dropout | | [128] |
| Softmax | [128 x 4] | [4] |