ECE C247 Final Project

Yikai Wang
UID: 905522085
wangyk0817@g.ucla.edu

Shihao Yang
UID: 205945583
syang536@g.ucla.edu

Haoting Ni
UID:905545789
haotinn@g.ucla.edu

Yaochen Li
UID:406073856
yaochenli@g.ucla.edu

A. Abstract

The aim of this project is to optimize the classification accuracy of electroencephalography (EEG) data collected from Brain Computer Interaction (BCI) Competition IV [[2]] using various neural network architectures such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), CRNN, and LSTM. We evaluate each CNN & RNN architecture thoroughly.

B. Introduction

To set a baseline model, we first use a naive CNN architecture with total parameters of 360k and 2 convolutional layers without dropout. Then we fine-tune the naive CNN to see if the EEG data can be modeled well within a smaller scale of parameters. After that, to be able to compare with RNN models, we optimize the CNN architecture with total parameters of 310k and 4 convolutional layers with batch normalization and dropout, where we achieve an accuracy as high as above 70%. Moreover, several post-CNN architectures are built, including RNN, CRNN, and CNN+LSTM [3]. The RNN has 1.4M and 2 bidirectional LSTM layers with batch-norm and dropout layers. CRNN has 620k parameters with 3 convolutional layers and 3 bidirectoinal LSTM layers. And CNN+LSTM model takes 250x1 input shape and has the output shape of 4 classes. Besides, we also build a ResNet to see its performance on the EEG data.

C. Results

C.1. Naive CNN

Before trying other models and optimize the classification accuracy, we decided to train a naive CNN model as our baseline to its general performance on this EGG dataset. The architecture is provided in Figure 4. The test accuracy is 0.459 across all period of time.

The first naive CNN model has 360k trainable parameters in total. On the basis of that, we want to explore CNN models with similar structures but fewer parameters. After fine-tuning, we finally get a structure with 3 layers and 18k parameters. We train 2 models separately on subject_0 dataset and the total dataset. Then we test them across the dataset of other 8 subjects, as figure 1 illustrated [1]. Both models are optimized. As we can see in figure 1, the model trained on the total dataset has much better performance due to the larger scale of training data. During training, we found that the structure of the model trained on only 1 dataset needs to be adjusted to get higher accuracy. To enlarge the capability of the model trained on 1 dataset, we add a layer and get a model with an accuracy of 61% on the total dataset. The final model only has 21k parameters. The detailed structure is listed in figure 10.

C.2. Optimized CNN

We learn from the CNN architecture provided in the discussion coding file(CNN with data preprocessing.ipynb [5]), adding Batch Normalization, Max Pooling and dropout after every each convolutional layers. Most important, we learn from VGG16 architecture [6], adding more filters but smaller filter sizes (3,1) instead of (10,1). Eventually, we get a test accuracy 0.719. The architecture is provided in Figure 5.

We found that if we choose to train all the data across the whole time period, the data accuracy will eventually goes down. The validation accuracy increases while more training data provided and after rising to a peak at time series at 700, then it decreases dramatically till time series at 1000. We obeserved that the new data may be different from the first 700 or the model has reached the optimal performance, therefore no need to add more training data points.

Given the two test accuracy graphs for every single subject, Figure 12 and Figure 13, we concluded that in CNN, optimize the subject 1 classification accuracy will not help train across all other objects. The test accuracy differs starting from subjects 4 from both graphs. And each subjects should be trained seperately if necessary. If we optimize

Subject ID	Train Accuracy	Test Accuracy
0	100%	40%
1	100%	48%
2	100%	30%
3	100%	38%
4	100%	30%
5	100%	37%
6	100%	28%
7	100%	38%
8	100%	45%

Table 1. Accuracy across all subjects with different subjects

Subject ID	Train Accuracy	Test Accuracy
0	35%	20%
1	40%	36%
2	59%	46%
3	71%	42%
4	86%	28%
5	99%	39%
6	98%	56%
7	93%	44%
8	97%	49%

Table 2. Accuracy across all subjects with different subjects

the classification accuracy across all subjects, we notice that subject 4 and subject 9 has the highest test accuracy, which is above 0.7. They are even higher than the test accuracy across all subjects. In conclusion, Optimizing the classification accuracy across all subjects doesn't apply to all the subjects, but has a high test accuracy for all subjects eventually.

C.3. Recurrent Neural Network (RNN)

RNN structures process sequential data by maintaining a hidden state which captures information from previous inputs. Traditional RNNs face gradient vanishing problems, which may cause the network to stop learning. Thus, We will use bidirectional LSTM to approach this problem.

In our RNN structure, we have two layers of bidirectional LSTM, after each layer, we have batch norm layers and dropout layers to reduce the noise and maximize the performance. Overall, we get an accuracy of 0.3111. The summary of the layers and parameter numbers are attached as Figure3. RNN with respect to each subject in train and test accuracy are listed as table and the chart is attached as Figure8

C.4. Convolutional Recurrent Neural Network (CRNN)

From RNN, we get a low accuracy, so we combine CNN and bidirectional LSTM as our new model to improve the performance accuracy. The structure we build is 3 convolutional layers followed by 3 bidirectional LSTM layers. CNNs are well-suited for processing spatial data, while RNNs are effective at processing sequential data. CNN extract features between the steps which reduce the difficulty and LSTM extract features with highly correlation. The summary of the layers and parameter numbers are attached as Figure6. CRNN with respect to each subject in train and test accuracy are listed as table.

C.5. Convolutional Neural Network+Long Short-Term Memory(CNN+LSTM)

Since the CRNN model did not perform well on the test data and give us a relatively low accuracy score, we theoritically analyze the reason. while CRNN models can be effective in many applications, they may not always be the best choice for EEG-based BCI data due to the complexity of the temporal dynamics in EEG signals, limited sample sizes, and noise and variability in the data.

Then we come up with a more promising model regarding to the EEG dataset - CNN + LSTM (Convolutional Neural Network + Long Short-Term Memory model). The reason we choose to explore this model is because CNN+LSTM models may be more appropriate for tasks where the input sequence length varies or real-time processing is required, such as in EEG-based BCI data which we are working on.

Overall, the test accuracy goes up to 65.52%, and the model architecture is presented in Figure 7.

C.6. Residual neural network (ResNet)

We've also made attemps to build a ResNet to see its performance on EEG data. From previous tuning experience, we found too deep layers could lead to overfitting and thus high validation loss. So we choose a ResNet architecture with few blocks. We first build a ResNet [4] for only subject_0 dataset. We found that overfitting will appear once there are more than 2 residual blocks. However, when we train it on the total 9 datasets, a structure with 4 residual blocks can still work. We think this will happen because the small dataset can bring down the critical depth of the network when overfitting will appear. Figure 2 shows the test accuracies of the two model trained on different datasets. Figure 11 lists the architecture of the ResNet(trained on the total 9 datasets).

From figure2 we can see that the fluctuation of the model trained on single dataset is larger than that trained on the total 9 datasets. One of the reasons may lie in the fine structures of the 2 models. Also, it may also indicate that ResNet

can learn some deeper features of the individual subjects, which other subjects do not have.

Compared with other architectures, ResNet does not show a clear advantage. That is because ResNet structure is usually very deep and can lead to overfitting. Besides, the dataset is not large enough to train a model with large trainable parameters. Also, simple data augmentation has been tested by us, including height shift(time series shift in EEG data) and rescaling. However, no clear improvement was observed. On the contrary, the performance of ResNet gets worse. We think that is because a simple shift or rescaling of the EEG signal is not a scientific way. To accomplish effective data augmentation. Deeper knowledge about EEG data and the experiment setup are important.

D. Discussion

In our final project, we explored different neural network architectures for EEG-based BCI data classification. We started by training a naive CNN model as a baseline and obtained a test accuracy of 0.459 across all 9 datasets. We then tried to build CNN models with similar structures but fewer parameters. After tuning different hyperparameters, we found that a model with 3 layers and 18k parameters trained on the total dataset performed better than a model trained on only one dataset. We also optimized our CNN model by adding Batch Normalization, Max Pooling layers, and drop-out after the convolutional layers, and using smaller filter sizes. Finally, we got a test accuracy of 0.719 (well above 70%).

Next, we tried the recurrent neural network (RNN) structure, but traditional RNNs faced the problem of gradient vanishing, so we used bidirectional LSTM to solve this issue. We built a two-layer bidirectional LSTM model with batch normalization and dropout layers and achieved a test accuracy of 31.11%.

We then combined CNN and bidirectional LSTM to create a convolutional recurrent neural network (CRNN) model. The CRNN model consists of 3 convolutional layers and 3 bidirectional LSTM layers and test accuracy of 28%-59% across different subjects is got.

In order to further improve the performance, we then explored a CNN+LSTM model, which may be more appropriate for tasks where the input sequence length varies or real-time processing is required. The CNN+LSTM model has a test accuracy of 65.52% . Compared with other architectures, this model works fairly well. That indicates that this type of data indeed can be well modeled with the CNN+LSTM model.

Finally, we attempted to build a residual neural network (ResNet) but found that deeper layers may lead to overfitting. We were able to train a ResNet with 4 residual blocks on the total dataset and achieved a test accuracy of 49.06%. However, compared to other architectures, ResNet did not show a clear advantage.

All in all, in this final project, we have explored various neural networks with different architectures for EEG-based BCI data classification. We found that the optimized CNN model had the best performance among our models explorations. We also found that training each subject separately may be necessary to optimize classification accuracy, and that deeper models may not always perform better due to overfitting problems. Future work can focus on exploring more sophisticated data augmentation techniques and incorporating domain-specific knowledge to further improve the performance of these models.

E. Acknowledgements

We gratefully appreciate the help and guidance from our professor Jonathan Kao and all the TAs. Some of our works also reference from the lecture slides and TA's discussion code/documents.

References

[1] Methods of training (see attached). 1

[2] G.R.Muller-Putz A.Schlogl and G.Pfurtscheller C.Brunner, R.Leeb. Bci competition 2008 – graz data set a. https://www.bbci.de/competition/iv/desc_2a.pdf. 1

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 1

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[5] Haoting Ni. Jupyter notebook file within the project: Cnn with data preprocessing.ipynb. 1

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

F. Method

F.1. Performance of all algorithms

Model	Accuracy
Naive CNN	60.64%
Optimized CNN	**71.9%**
RNN	31.11%
CRNN	51.47%
CNN + LSTM	**65.52%**
ResNet	49.06%

F.2. Architectures + Figures

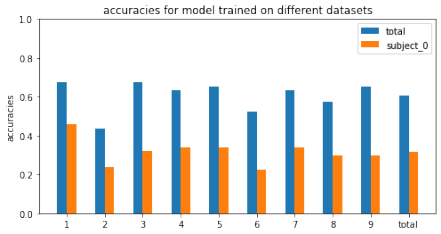

Figure 1. Accuracies of fine-tuned CNN models trained on different datasets

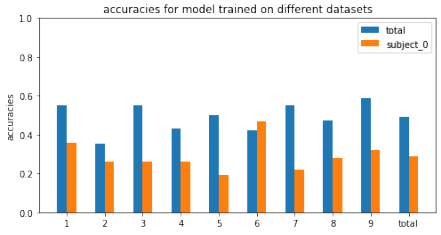

Figure 2. Accuracies of ResNet models trained on different datasets

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectiona  (None, 22, 256)           1156096
l)

batch_normalization (BatchN  (None, 22, 256)           1024
ormalization)

dropout (Dropout)            (None, 22, 256)           0

bidirectional_1 (Bidirectio  (None, 22, 128)           164352
nal)

batch_normalization_1 (Batc  (None, 22, 128)           512
hNormalization)

flatten (Flatten)            (None, 2816)              0

dropout_1 (Dropout)          (None, 2816)              0

dense (Dense)                (None, 32)                90144

dropout_2 (Dropout)          (None, 32)                0

dense_1 (Dense)              (None, 4)                 132

=================================================================
Total params: 1,412,260
Trainable params: 1,411,492
Non-trainable params: 768
```

Figure 3. Architecture of RNN.

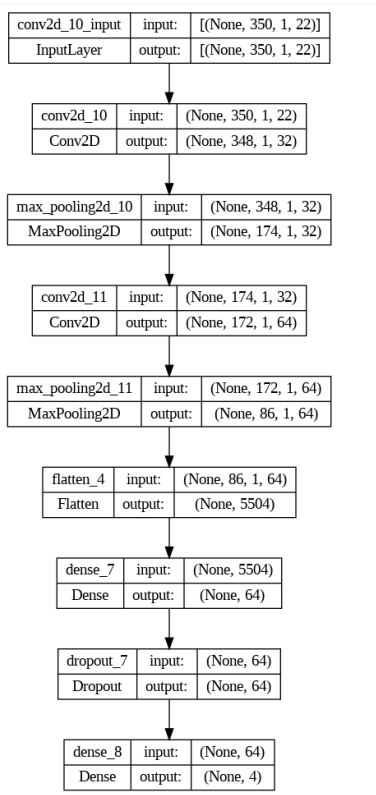

Figure 4. Architecture of naive CNN.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 350, 1, 200)       13400

max_pooling2d_14 (MaxPoolin  (None, 117, 1, 200)       0
g2D)

batch_normalization_4 (Batc  (None, 117, 1, 200)       800
hNormalization)

dropout_9 (Dropout)          (None, 117, 1, 200)       0

conv2d_15 (Conv2D)           (None, 117, 1, 100)       180100

max_pooling2d_15 (MaxPoolin  (None, 39, 1, 100)        0
g2D)

batch_normalization_5 (Batc  (None, 39, 1, 100)        400
hNormalization)

dropout_10 (Dropout)         (None, 39, 1, 100)        0

conv2d_16 (Conv2D)           (None, 39, 1, 100)        50100

max_pooling2d_16 (MaxPoolin  (None, 13, 1, 100)        0
g2D)

batch_normalization_6 (Batc  (None, 13, 1, 100)        400
hNormalization)

dropout_11 (Dropout)         (None, 13, 1, 100)        0

conv2d_17 (Conv2D)           (None, 13, 1, 200)        60200

max_pooling2d_17 (MaxPoolin  (None, 5, 1, 200)         0
g2D)

batch_normalization_7 (Batc  (None, 5, 1, 200)         800
hNormalization)

dropout_12 (Dropout)         (None, 5, 1, 200)         0

flatten_6 (Flatten)          (None, 1000)              0

dense_11 (Dense)             (None, 4)                 4004

=================================================================
Total params: 310,204
Trainable params: 309,004
Non-trainable params: 1,200
```

Figure 5. Architecture of optimized CNN.

```
[ ]  model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)            (None, 22, 991, 25)       275

 elu_3 (ELU)                  (None, 22, 991, 25)       0

 batch_normalization_3 (Batc  (None, 22, 991, 25)       100
 hNormalization)

 conv2d_4 (Conv2D)            (None, 2, 991, 50)        26300

 elu_4 (ELU)                  (None, 2, 991, 50)        0

 batch_normalization_4 (Batc  (None, 2, 991, 50)        200
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 2, 247, 50)        0
 2D)

 conv2d_5 (Conv2D)            (None, 2, 238, 100)       50100

 elu_5 (ELU)                  (None, 2, 238, 100)       0

 batch_normalization_5 (Batc  (None, 2, 238, 100)       400
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 2, 59, 100)        0
 2D)

 dropout_2 (Dropout)          (None, 2, 59, 100)        0

 permute_1 (Permute)          (None, 59, 100, 2)        0

 time_distributed_1 (TimeDis  (None, 59, 200)           0
 tributed)

 bidirectional_3 (Bidirectio  (None, 59, 256)           336896
 nal)

 bidirectional_4 (Bidirectio  (None, 59, 128)           164352
 nal)

 bidirectional_5 (Bidirectio  (None, 64)                41216
 nal)

 dropout_3 (Dropout)          (None, 64)                0

 dense_1 (Dense)              (None, 4)                 260

=================================================================
Total params: 620,099
Trainable params: 619,749
Non-trainable params: 350
```

Figure 6. Architecture of CRNN.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 250, 1, 25)        5525

 max_pooling2d (MaxPooling2D  (None, 84, 1, 25)         0
 )

 batch_normalization (BatchN  (None, 84, 1, 25)         100
 ormalization)

 dropout (Dropout)            (None, 84, 1, 25)         0

 conv2d_1 (Conv2D)            (None, 84, 1, 50)         12550

 max_pooling2d_1 (MaxPooling  (None, 28, 1, 50)         0
 2D)

 batch_normalization_1 (Batc  (None, 28, 1, 50)         200
 hNormalization)

 dropout_1 (Dropout)          (None, 28, 1, 50)         0

 conv2d_2 (Conv2D)            (None, 28, 1, 100)        50100

 max_pooling2d_2 (MaxPooling  (None, 10, 1, 100)        0
 2D)

 batch_normalization_2 (Batc  (None, 10, 1, 100)        400
 hNormalization)

 dropout_2 (Dropout)          (None, 10, 1, 100)        0

 conv2d_3 (Conv2D)            (None, 10, 1, 200)        200200

 max_pooling2d_3 (MaxPooling  (None, 4, 1, 200)         0
 2D)

 batch_normalization_3 (Batc  (None, 4, 1, 200)         800
 hNormalization)

 dropout_3 (Dropout)          (None, 4, 1, 200)         0

 flatten (Flatten)            (None, 800)               0

 dense (Dense)                (None, 100)               80100

 reshape (Reshape)            (None, 100, 1)            0

 lstm (LSTM)                  (None, 10)                480

 dense_1 (Dense)              (None, 4)                 44

=================================================================
Total params: 350,499
Trainable params: 349,749
Non-trainable params: 750
```

Figure 7. Architecture of CNN + LSTM.

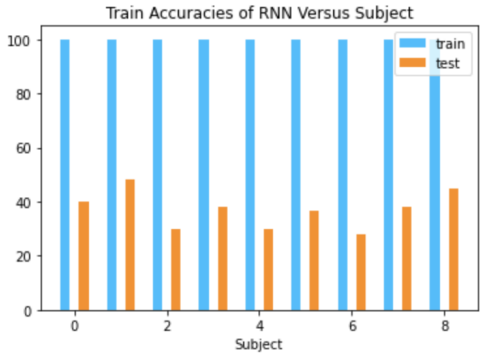

Figure 8. Architecture of CRNN.

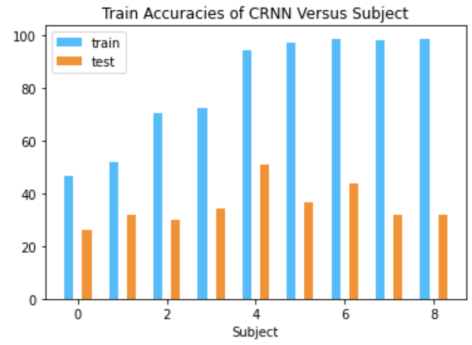

Figure 9. Architecture of CRNN.

```
=================================================================
 conv1 (Conv2D)               (None, 1000, 22, 6)       90

 average_pooling2d (AverageP  (None, 334, 22, 6)        0
 ooling2D)

 batch_normalization (BatchN  (None, 334, 22, 6)        24
 ormalization)

 dropout (Dropout)            (None, 334, 22, 6)        0

 conv2 (Conv2D)               (None, 334, 22, 12)       516

 average_pooling2d_1 (Averag  (None, 111, 22, 12)       0
 ePooling2D)

 batch_normalization_1 (Batc  (None, 111, 22, 12)       48
 hNormalization)

 conv6 (Conv2D)               (None, 105, 22, 24)       2040

 average_pooling2d_2 (Averag  (None, 35, 22, 24)        0
 ePooling2D)

 batch_normalization_2 (Batc  (None, 35, 22, 24)        96
 hNormalization)

 conv2d (Conv2D)              (None, 29, 22, 24)        4056

 average_pooling2d_3 (Averag  (None, 9, 22, 24)         0
 ePooling2D)

 batch_normalization_3 (Batc  (None, 9, 22, 24)         96
 hNormalization)

 flatten (Flatten)            (None, 4752)              0

 Dropout2 (Dropout)           (None, 4752)              0

 class-probs (Dense)          (None, 4)                 19012

=================================================================
Total params: 25,978
Trainable params: 25,846
Non-trainable params: 132
_________________________________________________________________
```

Figure 10. Architecture of fine-tune CNN.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 1000, 22, 1)]	0	[]
f1 (Conv2D)	(None, 500, 22, 12)	84	['input_1[0][0]']
batch_normalization (BatchNorm alization)	(None, 500, 22, 12)	48	['f1[0][0]']
activation (Activation)	(None, 500, 22, 12)	0	['batch_normalization[0][0]']
average_pooling2d (AveragePool ing2D)	(None, 500, 22, 12)	0	['activation[0][0]']
f2 (Conv2D)	(None, 250, 22, 24)	888	['average_pooling2d[0][0]']
batch_normalization_1 (BatchNo rmalization)	(None, 250, 22, 24)	96	['f2[0][0]']
activation_1 (Activation)	(None, 250, 22, 24)	0	['batch_normalization_1[0][0]']
f3 (AveragePooling2D)	(None, 250, 22, 24)	0	['activation_1[0][0]']
conv2d (Conv2D)	(None, 250, 22, 24)	2904	['f3[0][0]']
batch_normalization_2 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d[0][0]']
activation_2 (Activation)	(None, 250, 22, 24)	0	['batch_normalization_2[0][0]']
conv2d_1 (Conv2D)	(None, 250, 22, 24)	2904	['activation_2[0][0]']
batch_normalization_3 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d_1[0][0]']
add (Add)	(None, 250, 22, 24)	0	['batch_normalization_3[0][0]', 'f3[0][0]']
activation_3 (Activation)	(None, 250, 22, 24)	0	['add[0][0]']
conv2d_2 (Conv2D)	(None, 250, 22, 24)	2904	['activation_3[0][0]']
batch_normalization_4 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d_2[0][0]']
activation_4 (Activation)	(None, 250, 22, 24)	0	['batch_normalization_4[0][0]']
conv2d_3 (Conv2D)	(None, 250, 22, 24)	2904	['activation_4[0][0]']
batch_normalization_5 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d_3[0][0]']
add_1 (Add)	(None, 250, 22, 24)	0	['batch_normalization_5[0][0]', 'activation_3[0][0]']
activation_5 (Activation)	(None, 250, 22, 24)	0	['add_1[0][0]']
conv2d_4 (Conv2D)	(None, 250, 22, 24)	2904	['activation_5[0][0]']
batch_normalization_6 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d_4[0][0]']
activation_6 (Activation)	(None, 250, 22, 24)	0	['batch_normalization_6[0][0]']
conv2d_5 (Conv2D)	(None, 250, 22, 24)	2904	['activation_6[0][0]']
batch_normalization_7 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d_5[0][0]']
add_2 (Add)	(None, 250, 22, 24)	0	['batch_normalization_7[0][0]', 'activation_5[0][0]']
activation_7 (Activation)	(None, 250, 22, 24)	0	['add_2[0][0]']
conv2d_6 (Conv2D)	(None, 250, 22, 24)	2904	['activation_7[0][0]']
batch_normalization_8 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d_6[0][0]']
activation_8 (Activation)	(None, 250, 22, 24)	0	['batch_normalization_8[0][0]']
conv2d_7 (Conv2D)	(None, 250, 22, 24)	2904	['activation_8[0][0]']
batch_normalization_9 (BatchNo rmalization)	(None, 250, 22, 24)	96	['conv2d_7[0][0]']
add_3 (Add)	(None, 250, 22, 24)	0	['batch_normalization_9[0][0]', 'activation_7[0][0]']
activation_9 (Activation)	(None, 250, 22, 24)	0	['add_3[0][0]']
conv2d_8 (Conv2D)	(None, 250, 22, 4)	292	['activation_9[0][0]']
batch_normalization_10 (BatchN ormalization)	(None, 250, 22, 4)	16	['conv2d_8[0][0]']
activation_10 (Activation)	(None, 250, 22, 4)	0	['batch_normalization_10[0][0]']
flatten (Flatten)	(None, 22000)	0	['activation_10[0][0]']
dropout (Dropout)	(None, 22000)	0	['flatten[0][0]']
dense (Dense)	(None, 4)	88004	['dropout[0][0]']

Total params: 113,428
Trainable params: 112,964
Non-trainable params: 464

Figure 11. Architecture of ResNet.

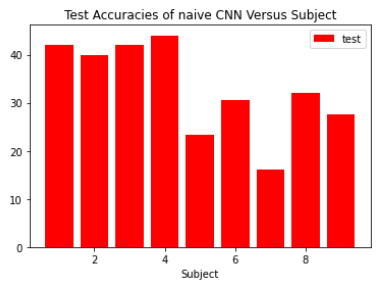

Figure 12. Test Accuracy of Naive CNN.

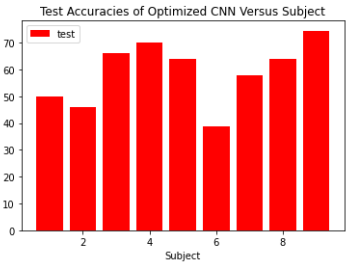

Figure 13. Test Accuracy of Optimized CNN.