ECE C247 Course Project Report

Po-Yen Yang (Paul Yang) Yu-Wei Chen Chen-Wei Huang Sinya Peng

Abstract

This project aims to perform a multi-class classification of electroencephalography (EEG) data provided by the BCI competition, in which we are going to perform classification comprising four classes for different motor imaginary tasks. We compare the performance of multiple deep learning architectures, including convolutional neural networks (CNNs), Recurrent neural networks (RNNs), and Transformer with Attention models. Furthermore, we designed an improved CNN model achieving the best accuracy compared to the baseline models. Through empirical evaluation on a single subject, all subjects, and varying time durations, our model is shown to provide an accuracy of over 75% in all scenarios.

1. Introduction

Starting with visualizing the data and using our insights to perform data preprocessing. We then train 7 architectures, including Naive LSTM, Naive Transformer, EEGNet [1], DeepConvNet [1], CNN+LSTM, CNN+Transformer, and the improved version of CNN (CNNPlus).

The remaining report is organized as follows: Section 2 gives the details of the experimental evaluation, the insight and discussion are written in Section 3, and the appendix contains the architectures and tables summarizing the performance of all models.

1.1. Preprocessing

We observed the first 500 time intervals are more relevant 5 after analyzing the data. Hence, we trimmed the data, applied max pooling, applied average pooling with Gaussian noise, and concatenated the results along the sample axis.

1.2. Naive LSTM

LSTM can be applied to time series forecasting. Therefore, we start with the Naive LSTM model, where the architecture is shown in figure 3.

This model has a simple structure with only one LSTM layer and one fully connected layer.

1.3. Naive Transformer

The transformer model is widely used for sequence classification tasks. Our Naive Transformer architecture is shown in figure 5.

The Naive Transformer is a feedforward neural network comprising a PositionalEncoding, a TransformerEncoder, and a linear decoder. It takes an input sequence of features and applies positional encoding to add information about the position of each feature in the sequence. The TransformerEncoder processes the encoded sequence to extract its features while the decoder maps the extracted features to the output labels. It includes a self-attention mechanism to learn global dependencies between the input features.

1.4. EEGNet

EEGNet is a related work proposed by [2]. Our implemented architecture is shown in figure 7.

It consists of three main components. First, a 1-D convolutional layer with batch normalization. Then a depthwise convolutional layer and a separable convolutional layer both with average pooling and dropout. Lastly, a linear classifier outputting a classification prediction for four different classes.

1.5. DeepConvNet

DeepConvNet is a related work proposed by [3]. Our implemented architecture is shown in figure 2.

DeepConvNet is a deep convolutional neural network with four convolutional layers followed by a fully connected layer. The first batch of the convolutional layers consists of two 1-D convolution layers to extract temporal and spatial features. The remaining three convolutional layers are designed to extract high-level features. The network uses batch normalization, ReLU activation, max pooling, and dropout to prevent overfitting. The output layer has four neurons for classification.

1.6. CNN+LSTM

From previous research, we know CNN models perform well [5] on the EEG data. The design of the CNN+LSTM model is using the output of CNN as the input to LSTM. This allows LSTM to learn features from the input data learned by CNN. Our implemented architecture is shown in figure 4.

From the result of the naive LSTM model, we can observe that it is way overfitting. However, instead of just taking the original data as input, we first pass the data through a CNN model, which makes the input more complicated, and we believe that would lead to LSTM performing better.

1.7. CNN+Transformer

CNN+Transformer is a related work proposed by [4]. Our implementation is shown in figure 6.

It combined CNNs and Transformers to overcome their individual limitations and get the benefits of both: CNNs for local feature learning and Transformers for global correlation capturing. The overall structure of this model includes a convolution module, a self-attention module, and a classifier. All convolutional channels at each time point are served as a token and fed into the self-attention module to learn the global temporal features. Lastly, several fully-connected layers are added to map the results to the final classification.

1.8. CNNPlus

The CNNPlus model is an improved version of the CNN-based model inspired by DeepConvNet. Our implemented architecture is shown in figure 1.

We added another convolutional layer before the second batch normalization and stacked three linear layers to gradually decrease the number of neurons instead of just one linear layer. As for the activation function of the CNN layers, we adopted ReLU as we obtained empirically better accuracies compared to ELU and Leaky ReLU. Additionally, we also changed the first pooling layer to an average pooling layer instead of a max pooling layer.

2. Result

2.1. Classification on One Subject

In this experiment, we compared the performance of models trained on the complete dataset (all subjects) and trained on subject one only as shown in table 3 and table 2 respectively.

From Table 2, we can observe DeepConvNet and CNNPlus architectures perform better than other models when trained only on subject one, with CNNPlus achieving the highest accuracy of 74.00% when testing on subject one.

2.2. Classification on All Subjects

From Table 3, we can see that the best test accuracy on all subjects is achieved by the CNNPlus method with an accuracy of 75.06%, outperforming all other methods. This is a significant improvement compared to the best test accuracy achieved when training only on subject one (46.39% by CNN+LSTM).

Interestingly, we can see that the performance of some methods improves significantly when training on all subjects compared to training only on subject one. For example, the Naive LSTM method improves from a test accuracy of 29.97% to 33.18%, and the Naive Transformer method improves from 33.97% to 47.40%.

2.3. Compare One Subject vs All Subjects

Overall, these results suggest that training on all subjects can improve the performance of the models, and different models have different sensitivity to the inclusion of data from multiple subjects. In particular, DeepConvNet and CNNPlus models continue to perform well, achieving accuracies of 73.5% and 80.5%, respectively, when testing on subject one.

Training across all subjects may help improve the model's generalization ability by learning features that are common across all subjects. This is reflected in the improved test accuracy on subject one when training over all subjects, as shown in Table 3. However, it is crucial to note the performance gain on subject one may come at the cost of decreased performance on other subjects, as the models may overfit the training data. Therefore, it is important to carefully evaluate the trade-off between model complexity and performance on different subjects.

2.4. Classification as A Function of Time

The table 5 shows the performance of training on different time periods of data using CNNPlus, with the best test accuracy for each time period listed.

From the table, it is shown that test accuracy generally increases as the time period of data used for training increases. The test accuracy starts at 56.43% for a time period of 100 ms and increases to a peak of 73.48% for time periods of 500 ms and 600 ms. The test accuracy decreases slightly afterward, with the lowest accuracy of 68.34% for a time period of 1000 ms.

This suggests that longer time periods of data can lead to better classification accuracy up to a certain point. However, beyond a certain time period, additional data may not provide much additional benefit or may even have a negative impact respecting test accuracy. The exact amount of time required to achieve the best test accuracy may depend on various factors, such as the complexity of the classification problem and the amount of noise in the data.

3. Discussion

3.1. Performance of Different Architectures

DeepConvNet and CNNPlus models achieved the best results, outperforming others. EEGNet, CNN+LSTM, and CNN+Transformer also showed promising results. Naive LSTM and Naive Transformer have lower accuracies. Overall, architectures without recurrent layers performed better than those combining convolutional and recurrent layers, and deeper architectures tended to perform better than shallower ones.

3.1.1 Performance of CNN vs RNN

In our experiment, we found the performance of CNN comprehensively outperforms RNN. CNNs are often used for EEG classification tasks since they can extract spatial features from multi-channel EEG signals, meaning CNNs can learn to detect patterns in different parts of the EEG signal that are related to the target class. This is particularly useful in EEG classification tasks, as different types of brain activities are often distributed across different EEG channels.

In contrast, RNNs are typically used for sequential data classification tasks, such as natural language processing or time-series data analysis. While EEG signals are also sequential in nature, the spatial dependencies between different channels are vital for classification, making CNN a more natural fit for EEG classification tasks. It is worth noting that some studies have also explored the use of RNNs or a combination of CNNs and RNNs for this task.

3.2. Choice of Architectures and Parameters

- **Pooling**: For our self-built CNNPlus model and CNN+LSTM model, we adopted the average pooling layer after the first two convolutional layers since we want the model to learn more in the first batch, not just remember the maximum number in the kernel size. We chose max pooling for other layers, as when passing through more layers, the number that matters most to the output will be the maximum number.
- **Kernel size**: For our CNN+LSTM model, we chose most kernel sizes to be a bigger number, (1, 11), since we don't want the model to overfit when passing into LSTM. We chose most of the kernel sizes to be a smaller number for CNNPlus, (1, 5) since we don't want the model to shift too much at a time. While the second convolution layer for both CNN+LSTM and CNNPlus models, we chose the kernel size to be (22, 1) as we want to change the output shape from 22 to 1.
- **Epoch**: The epoch size we set is 100. If the epoch size is too small, then the model might still be learning and we cannot get the best testing accuracy. While if the epoch size is too large, the model stops learning after the training accuracy reaches 1, and it increases both the computational cost and time consumption.
- **Learning rate**: A high learning rate has a higher chance to step in the wrong direction, while a low learning rate may take more time to reach the optimal point. The learning rate we used for our CNNPlus model is 0.001 and the learning rate for other models is 0.0005. Both are higher than the initial setting (0.0001) as we have complicated models, hence we adopted a larger learning rate to take a bigger step each time.
- **Batch size**: A large batch size may include too much information and lose the purpose of generalization, while a small batch size may include too less information and cause noise or bias. The batch size we ended up setting is 64, as it performed empirically better than other settings that we tested such as 16, 32, 128, and 256.
- **Optimizer**: We adopted Adam as our optimizer for all our models as it performs empirically better than other optimizers we tested, for instance, SGD, RMSprop, and Adagrad.
- **Depth**: depth can be a crucial factor for determining the fitting ability of end-to-end models. Testing on different datasets showed that the CNN+Transformer is insensitive to self-attention depth; on the other hand, parameters increase proportionally with depth increase and complicated computation.
- **Number of heads**: changes in the number of heads have not shown significant effects on feature learning.
- **Token**: the kernel size is especially important when it comes to token generation. If a chosen kernel is too large, we might not be able to capture useful details, while a small kernel can be easily affected by local noise. The CNN+Transformer performs best when the kernel size is in the range of 15 to 45. After this point, the performance no longer rises significantly. Therefore, we also experimented with kernel sizes within this range.

References

[1] Bixbeat Alex. How to understand the densenet implementation, 2017. 1

[2] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. Eegnet: A compact convolutional network for eeg-based brain-computer interfaces. *arXiv preprint arXiv:1611.08024*, 2016. 1

[3] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for eeg decoding and visualization. *Human brain mapping*, 38(11):5391–5420, 2017. 1

[4] Yonghao Song, Qingqing Zheng, Bingchuan Liu, and Xiaorong Gao. Eeg conformer: Convolutional transformer for eeg decoding and visualization. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 2022. 2

[5] Zhichuan Tang, Chao Li, and Shouqian Sun. Single-trial eeg classification of motor imagery using deep convolutional neural networks. *Optik*, 130:11–18, 2017. 1

A. Model Architecture

```
==========================================================================================
Layer (type (var_name):depth-idx)        Input Shape          Output Shape         Param #
==========================================================================================
CNNPlus (CNNPlus)                        [64, 1, 22, 250]     [64, 4]              --
+ Sequential (cnn): 1-1                  [64, 1, 22, 250]     [64, 256, 1, 28]     --
│    + Conv2d (0): 2-1                   [64, 1, 22, 250]     [64, 16, 22, 250]    96
│    + Conv2d (1): 2-2                   [64, 16, 22, 250]    [64, 32, 1, 254]     11,296
│    + BatchNorm2d (2): 2-3              [64, 32, 1, 254]     [64, 32, 1, 254]     64
│    + ReLU (3): 2-4                     [64, 32, 1, 254]     [64, 32, 1, 254]     --
│    + AvgPool2d (4): 2-5                [64, 32, 1, 254]     [64, 32, 1, 127]     --
│    + Dropout (5): 2-6                  [64, 32, 1, 127]     [64, 32, 1, 127]     --
│    + Conv2d (6): 2-7                   [64, 32, 1, 127]     [64, 64, 1, 127]     10,304
│    + Conv2d (7): 2-8                   [64, 64, 1, 127]     [64, 128, 1, 123]    41,088
│    + BatchNorm2d (8): 2-9              [64, 128, 1, 123]    [64, 128, 1, 123]    256
│    + ReLU (9): 2-10                    [64, 128, 1, 123]    [64, 128, 1, 123]    --
│    + MaxPool2d (10): 2-11              [64, 128, 1, 123]    [64, 128, 1, 61]     --
│    + Dropout (11): 2-12                [64, 128, 1, 61]     [64, 128, 1, 61]     --
│    + Conv2d (12): 2-13                 [64, 128, 1, 61]     [64, 256, 1, 57]     164,096
│    + BatchNorm2d (13): 2-14            [64, 256, 1, 57]     [64, 256, 1, 57]     512
│    + ReLU (14): 2-15                   [64, 256, 1, 57]     [64, 256, 1, 57]     --
│    + MaxPool2d (15): 2-16              [64, 256, 1, 57]     [64, 256, 1, 28]     --
│    + Dropout (16): 2-17                [64, 256, 1, 28]     [64, 256, 1, 28]     --
+ Sequential (classify): 1-2             [64, 7168]           [64, 4]              --
│    + Linear (0): 2-18                  [64, 7168]           [64, 3584]           25,693,696
│    + Linear (1): 2-19                  [64, 3584]           [64, 1792]           6,424,320
│    + Linear (2): 2-20                  [64, 1792]           [64, 4]              7,172
==========================================================================================
Total params: 32,352,900
Trainable params: 32,352,900
Non-trainable params: 0
Total mult-adds (G): 3.28
==========================================================================================
Input size (MB): 1.41
Forward/backward pass size (MB): 91.36
Params size (MB): 129.41
Estimated Total Size (MB): 222.18
==========================================================================================
```

Figure 1. Model architecture of CNNPlus.

```
==========================================================================================
Layer (type (var_name):depth-idx)        Input Shape          Output Shape         Param #
==========================================================================================
DeepConvNet (DeepConvNet)                [64, 1, 22, 250]     [64, 4]              --
+ Sequential (conv1): 1-1                [64, 1, 22, 250]     [64, 25, 1, 127]     --
│    + Conv2d (0): 2-1                   [64, 1, 22, 250]     [64, 25, 22, 250]    150
│    + Conv2d (1): 2-2                   [64, 25, 22, 250]    [64, 25, 1, 254]     13,775
│    + BatchNorm2d (2): 2-3              [64, 25, 1, 254]     [64, 25, 1, 254]     50
│    + ReLU (3): 2-4                     [64, 25, 1, 254]     [64, 25, 1, 254]     --
│    + MaxPool2d (4): 2-5                [64, 25, 1, 254]     [64, 25, 1, 127]     --
│    + Dropout (5): 2-6                  [64, 25, 1, 127]     [64, 25, 1, 127]     --
+ Sequential (conv2): 1-2                [64, 25, 1, 127]     [64, 50, 1, 63]      --
│    + Conv2d (0): 2-7                   [64, 25, 1, 127]     [64, 50, 1, 127]     6,300
│    + BatchNorm2d (1): 2-8              [64, 50, 1, 127]     [64, 50, 1, 127]     100
│    + ReLU (2): 2-9                     [64, 50, 1, 127]     [64, 50, 1, 127]     --
│    + MaxPool2d (3): 2-10               [64, 50, 1, 127]     [64, 50, 1, 63]      --
│    + Dropout (4): 2-11                 [64, 50, 1, 63]      [64, 50, 1, 63]      --
+ Sequential (conv3): 1-3                [64, 50, 1, 63]      [64, 100, 1, 31]     --
│    + Conv2d (0): 2-12                  [64, 50, 1, 63]      [64, 100, 1, 63]     25,100
│    + BatchNorm2d (1): 2-13             [64, 100, 1, 63]     [64, 100, 1, 63]     200
│    + ReLU (2): 2-14                    [64, 100, 1, 63]     [64, 100, 1, 63]     --
│    + MaxPool2d (3): 2-15               [64, 100, 1, 63]     [64, 100, 1, 31]     --
│    + Dropout (4): 2-16                 [64, 100, 1, 31]     [64, 100, 1, 31]     --
+ Sequential (conv4): 1-4                [64, 100, 1, 31]     [64, 200, 1, 15]     --
│    + Conv2d (0): 2-17                  [64, 100, 1, 31]     [64, 200, 1, 31]     100,200
│    + BatchNorm2d (1): 2-18             [64, 200, 1, 31]     [64, 200, 1, 31]     400
│    + ReLU (2): 2-19                    [64, 200, 1, 31]     [64, 200, 1, 31]     --
│    + MaxPool2d (3): 2-20               [64, 200, 1, 31]     [64, 200, 1, 15]     --
│    + Dropout (4): 2-21                 [64, 200, 1, 15]     [64, 200, 1, 15]     --
+ Sequential (classify): 1-5             [64, 3000]           [64, 4]              --
│    + Linear (0): 2-22                  [64, 3000]           [64, 4]              12,004
==========================================================================================
Total params: 158,279
Trainable params: 158,279
Non-trainable params: 0
Total mult-adds (M): 628.75
==========================================================================================
Input size (MB): 1.41
Forward/backward pass size (MB): 96.21
Params size (MB): 0.63
Estimated Total Size (MB): 98.25
==========================================================================================
```

Figure 2. Model architecture of DeepConvNet.

```
==========================================================================================
Layer (type (var_name):depth-idx)        Input Shape          Output Shape         Param #
==========================================================================================
NaiveLSTM (NaiveLSTM)                    [64, 1, 22, 250] [64, 4]                  --
+ Sequential (lstm): 1-1                 [64, 250, 22]    [64, 250, 512]           --
│    + LSTM (0): 2-1                     [64, 250, 22]    [64, 250, 512]           573,440
+ Sequential (classify): 1-2             [64, 512]        [64, 4]                  --
│    + Dropout (0): 2-2                  [64, 512]        [64, 512]                --
│    + Linear (1): 2-3                   [64, 512]        [64, 4]                  2,052
==========================================================================================
Total params: 575,492
Trainable params: 575,492
Non-trainable params: 0
Total mult-adds (G): 9.18
==========================================================================================
Input size (MB): 1.41
Forward/backward pass size (MB): 65.54
Params size (MB): 2.30
Estimated Total Size (MB): 69.25
==========================================================================================
```

Figure 3. Model architecture of Naive LSTM.

```
==========================================================================================
Layer (type (var_name):depth-idx)        Input Shape          Output Shape         Param #
==========================================================================================
CNNLSTM (CNNLSTM)                        [64, 1, 22, 250]     [64, 4]              --
+ Sequential (cnn): 1-1                  [64, 1, 22, 250]     [64, 128, 1, 4]      --
│    + Conv2d (0): 2-1                   [64, 1, 22, 250]     [64, 16, 22, 240]    192
│    + Conv2d (1): 2-2                   [64, 16, 22, 240]    [64, 32, 1, 240]     11,296
│    + AvgPool2d (2): 2-3                [64, 32, 1, 240]     [64, 32, 1, 80]      --
│    + Dropout (3): 2-4                  [64, 32, 1, 80]      [64, 32, 1, 80]      --
│    + Conv2d (4): 2-5                   [64, 32, 1, 80]      [64, 64, 1, 70]      22,592
│    + BatchNorm2d (5): 2-6              [64, 64, 1, 70]      [64, 64, 1, 70]      128
│    + ELU (6): 2-7                      [64, 64, 1, 70]      [64, 64, 1, 70]      --
│    + MaxPool2d (7): 2-8                [64, 64, 1, 70]      [64, 64, 1, 23]      --
│    + Dropout (8): 2-9                  [64, 64, 1, 23]      [64, 64, 1, 23]      --
│    + Conv2d (9): 2-10                  [64, 64, 1, 23]      [64, 128, 1, 13]     90,240
│    + BatchNorm2d (10): 2-11            [64, 128, 1, 13]     [64, 128, 1, 13]     256
│    + ELU (11): 2-12                    [64, 128, 1, 13]     [64, 128, 1, 13]     --
│    + MaxPool2d (12): 2-13              [64, 128, 1, 13]     [64, 128, 1, 4]      --
│    + Dropout (13): 2-14                [64, 128, 1, 4]      [64, 128, 1, 4]      --
+ Linear (fc1): 1-2                      [64, 4, 128]         [64, 4, 128]         16,512
+ LSTM (lstm): 1-3                       [4, 64, 128]         [4, 64, 256]         264,192
+ Dropout (dropout): 1-4                 [64, 256]            [64, 256]            --
+ Linear (classify): 1-5                 [64, 256]            [64, 4]              1,028
==========================================================================================
Total params: 406,436
Trainable params: 406,436
Non-trainable params: 0
Total mult-adds (M): 483.46
==========================================================================================
Input size (MB): 1.41
Forward/backward pass size (MB): 54.27
Params size (MB): 1.63
Estimated Total Size (MB): 57.30
==========================================================================================
```

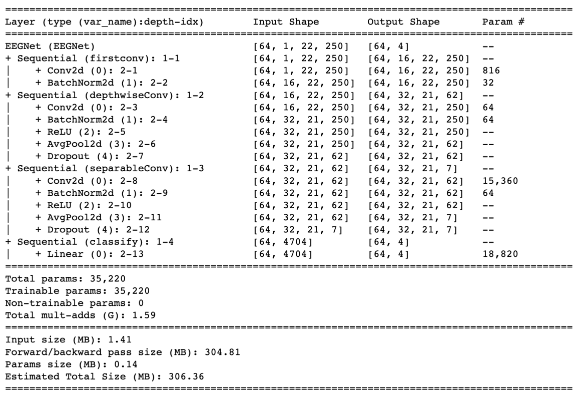
Figure 4. Model architecture of CNN+LSTM.

```
=============================================================================================================
Layer (type (var_name):depth-idx)                           Input Shape      Output Shape     Param #
=============================================================================================================
NaiveTransformer (NaiveTransformer)                         [64, 1, 22, 250] [64, 4]          508,762
+ PositionalEncoding (pos_encoder): 1-1                     [64, 22, 250]    [64, 22, 250]    --
+ TransformerEncoder (transformer_encoder): 1-2             [64, 22, 250]    [64, 22, 250]    --
│    + ModuleList (layers): 2-1                             --               --               --
│    │    + TransformerEncoderLayer (0): 3-1                [64, 22, 250]    [64, 22, 250]    --
│    │    │    + MultiheadAttention (self_attn): 4-1        [64, 22, 250]    [64, 22, 250]    251,000
│    │    │    + Dropout (dropout1): 4-2                    [64, 22, 250]    [64, 22, 250]    --
│    │    │    + LayerNorm (norm1): 4-3                     [64, 22, 250]    [64, 22, 250]    500
│    │    │    + Linear (linear1): 4-4                      [64, 22, 250]    [64, 22, 512]    128,512
│    │    │    + Dropout (dropout): 4-5                     [64, 22, 512]    [64, 22, 512]    --
│    │    │    + Linear (linear2): 4-6                      [64, 22, 512]    [64, 22, 250]    128,250
│    │    │    + Dropout (dropout2): 4-7                    [64, 22, 250]    [64, 22, 250]    --
│    │    │    + LayerNorm (norm2): 4-8                     [64, 22, 250]    [64, 22, 250]    500
│    │    + TransformerEncoderLayer (1): 3-2                [64, 22, 250]    [64, 22, 250]    --
│    │    │    + MultiheadAttention (self_attn): 4-9        [64, 22, 250]    [64, 22, 250]    251,000
│    │    │    + Dropout (dropout1): 4-10                   [64, 22, 250]    [64, 22, 250]    --
│    │    │    + LayerNorm (norm1): 4-11                    [64, 22, 250]    [64, 22, 250]    500
│    │    │    + Linear (linear1): 4-12                     [64, 22, 250]    [64, 22, 512]    128,512
│    │    │    + Dropout (dropout): 4-13                    [64, 22, 512]    [64, 22, 512]    --
│    │    │    + Linear (linear2): 4-14                     [64, 22, 512]    [64, 22, 250]    128,250
│    │    │    + Dropout (dropout2): 4-15                   [64, 22, 250]    [64, 22, 250]    --
│    │    │    + LayerNorm (norm2): 4-16                    [64, 22, 250]    [64, 22, 250]    500
+ Linear (decoder): 1-3                                     [64, 5500]       [64, 4]          22,004
=============================================================================================================
Total params: 1,548,290
Trainable params: 1,548,290
Non-trainable params: 0
Total mult-adds (M): 34.40
=============================================================================================================
Input size (MB): 1.41
Forward/backward pass size (MB): 28.43
Params size (MB): 2.15
Estimated Total Size (MB): 31.99
=============================================================================================================
```

Figure 5. Model architecture of Naive Transformer.

```
====================================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
====================================================================================================
Conformer                                          [64, 2440]                --
├─PatchEmbedding: 1-1                              [64, 61, 40]              --
│    └─Sequential: 2-1                             [64, 40, 1, 61]           --
│    │    └─Conv2d: 3-1                            [64, 40, 22, 976]         1,040
│    │    └─Conv2d: 3-2                            [64, 40, 1, 976]          35,240
│    │    └─BatchNorm2d: 3-3                       [64, 40, 1, 976]          80
│    │    └─ELU: 3-4                               [64, 40, 1, 976]          --
│    │    └─AvgPool2d: 3-5                         [64, 40, 1, 61]           --
│    │    └─Dropout: 3-6                           [64, 40, 1, 61]           --
│    └─Sequential: 2-2                             [64, 61, 40]              --
│    │    └─Conv2d: 3-7                            [64, 40, 1, 61]           1,640
│    │    └─Rearrange: 3-8                         [64, 61, 40]              --
├─TransformerEncoder: 1-2                          [64, 61, 40]              --
│    └─TransformerEncoderBlock: 2-3                [64, 61, 40]              --
│    │    └─ResidualAdd: 3-9                       [64, 61, 40]              6,640
│    │    └─ResidualAdd: 3-10                      [64, 61, 40]              13,080
│    └─TransformerEncoderBlock: 2-4                [64, 61, 40]              --
│    │    └─ResidualAdd: 3-11                      [64, 61, 40]              6,640
│    │    └─ResidualAdd: 3-12                      [64, 61, 40]              13,080
│    └─TransformerEncoderBlock: 2-5                [64, 61, 40]              --
│    │    └─ResidualAdd: 3-13                      [64, 61, 40]              6,640
│    │    └─ResidualAdd: 3-14                      [64, 61, 40]              13,080
│    └─TransformerEncoderBlock: 2-6                [64, 61, 40]              --
│    │    └─ResidualAdd: 3-15                      [64, 61, 40]              6,640
│    │    └─ResidualAdd: 3-16                      [64, 61, 40]              13,080
│    └─TransformerEncoderBlock: 2-7                [64, 61, 40]              --
│    │    └─ResidualAdd: 3-17                      [64, 61, 40]              6,640
│    │    └─ResidualAdd: 3-18                      [64, 61, 40]              13,080
│    └─TransformerEncoderBlock: 2-8                [64, 61, 40]              --
│    │    └─ResidualAdd: 3-19                      [64, 61, 40]              6,640
│    │    └─ResidualAdd: 3-20                      [64, 61, 40]              13,080
├─ClassificationHead: 1-3                          [64, 2440]                244
│    └─Sequential: 2-9                             [64, 4]                   --
│    │    └─Linear: 3-21                           [64, 256]                 624,896
│    │    └─ELU: 3-22                              [64, 256]                 --
│    │    └─Dropout: 3-23                          [64, 256]                 --
│    │    └─Linear: 3-24                           [64, 32]                  8,224
│    │    └─ELU: 3-25                              [64, 32]                  --
│    │    └─Dropout: 3-26                          [64, 32]                  --
│    │    └─Linear: 3-27                           [64, 4]                   132
====================================================================================================
Total params: 789,816
Trainable params: 789,816
Non-trainable params: 0
Total mult-adds (G): 3.68
====================================================================================================
Input size (MB): 5.63
Forward/backward pass size (MB): 563.57
Params size (MB): 3.16
Estimated Total Size (MB): 572.37
====================================================================================================
```

Figure 6. Model architecture of CNN+Transformer.



```
===========================================================================================
Layer (type (var_name):depth-idx)        Input Shape        Output Shape       Param #
===========================================================================================
EEGNet (EEGNet)                          [64, 1, 22, 250]   [64, 4]            --
+ Sequential (firstconv): 1-1            [64, 1, 22, 250]   [64, 16, 22, 250]  --
|    + Conv2d (0): 2-1                   [64, 1, 22, 250]   [64, 16, 22, 250]  816
|    + BatchNorm2d (1): 2-2              [64, 16, 22, 250]  [64, 16, 22, 250]  32
+ Sequential (depthwiseConv): 1-2        [64, 16, 22, 250]  [64, 32, 21, 62]   --
|    + Conv2d (0): 2-3                   [64, 16, 22, 250]  [64, 32, 21, 250]  64
|    + BatchNorm2d (1): 2-4              [64, 32, 21, 250]  [64, 32, 21, 250]  64
|    + ReLU (2): 2-5                     [64, 32, 21, 250]  [64, 32, 21, 250]  --
|    + AvgPool2d (3): 2-6                [64, 32, 21, 250]  [64, 32, 21, 62]   --
|    + Dropout (4): 2-7                  [64, 32, 21, 62]   [64, 32, 21, 62]   --
+ Sequential (separableConv): 1-3        [64, 32, 21, 62]   [64, 32, 21, 7]    --
|    + Conv2d (0): 2-8                   [64, 32, 21, 62]   [64, 32, 21, 62]   15,360
|    + BatchNorm2d (1): 2-9              [64, 32, 21, 62]   [64, 32, 21, 62]   64
|    + ReLU (2): 2-10                    [64, 32, 21, 62]   [64, 32, 21, 62]   --
|    + AvgPool2d (3): 2-11               [64, 32, 21, 62]   [64, 32, 21, 7]    --
|    + Dropout (4): 2-12                 [64, 32, 21, 7]    [64, 32, 21, 7]    --
+ Sequential (classify): 1-4             [64, 4704]         [64, 4]            --
|    + Linear (0): 2-13                  [64, 4704]         [64, 4]            18,820
===========================================================================================
Total params: 35,220
Trainable params: 35,220
Non-trainable params: 0
Total mult-adds (G): 1.59
===========================================================================================
Input size (MB): 1.41
Forward/backward pass size (MB): 304.81
Params size (MB): 0.14
Estimated Total Size (MB): 306.36
===========================================================================================
```

Figure 7. Model architecture of EEGNet.

B. Experimental Result

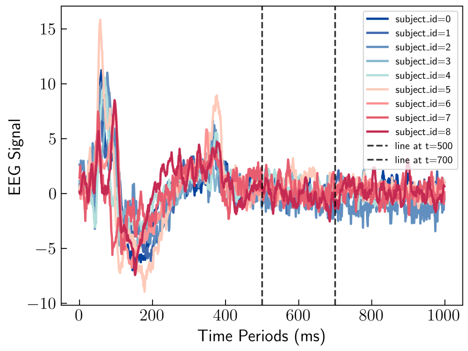


Figure 8. Data observation.

Table 1. General training setting.

Settings	Details
Dataset Augmentation	first 500 with gaussian noise
Optimizer	Adam
Epochs	100
Weight Decay	0 to 0.02
Learning Rate	5e-4 to 1e-3

Table 2. Performance of training on subject one.

Method	Best Test Accuracy on Subject One	Best Test Accuracy on All Subjects
NaiveLSTM	0.3600	0.2997
NaiveTransformer	0.4350	0.3397
CNN+Transformer	0.6850	0.4261
EEGNet	0.6300	0.4391
DeepConvNe	0.6500	0.4153
CNNLSTM	0.6900	**0.4639**
CNNPlus	**0.7400**	0.4052

Table 3. Performance of training on all subjects.

Method	Best Test Accuracy on Subject One	Best Test Accuracy on All Subjects
NaiveLSTM	0.3700	0.3318
NaiveTransformer	0.5250	0.4740
CNN+Transformer	0.6400	0.6727
EEGNet	0.6850	0.6388
DeepConvNe	0.7350	0.7223
CNNLSTM	0.6750	0.6992
CNNPlus	**0.8050**	**0.7506**

Table 4. Performance of training on all subjects across different subjects using CNNPlus.

Subject	Best Test Accuracy
Subject 0	0.7400
Subject 1	0.7400
Subject 2	0.8200
Subject 3	0.8000
Subject 4	**0.8511**
Subject 5	0.7143
Subject 6	0.8200
Subject 7	0.7800
Subject 8	0.7872

Table 5. Performance of training on different time periods of data using CNNPlus.

Time Period	Best Test Accuracy
100 ms	0.5643
200 ms	0.6569
300 ms	0.7088
400 ms	0.7291
500 ms	**0.7348**
600 ms	**0.7348**
700 ms	0.7049
800 ms	0.7235
900 ms	0.7071
1000 ms	0.6834