EEG Motion Classification with Convolutional and Recurrent Neural Networks

Christopher Yang
UID: 806302417
chrisy810@g.ucla.edu

Fengzhou Pan
UID: 806152050
panfengzhou@g.ucla.edu

Zhan Yu
UID: 306153151
yuzhan3232@g.ucla.edu

Wenkai Gong
UID: 406183261
mga19wg@g.ucla.edu

Abstract

This project explores the classification of EEG data provided by Brain-Computer Interaction (BCI) [1] Competition using both convolutional and recurrent neural networks. We compared the performance of four architectures, EEGNet [2], ShallowConvNet [3], DeepConvNet [3], and CRNN (shallow CNN + bidirectional LSTM), over three tasks: optimizing the classification accuracy for subject 1, optimizing that of all subjects, and the exploring the impact of data trimming in terms of time series. For the first task, the results showed a higher test accuracy when the models were trained with subject 1 only; we also observed a positive correlation between training data size and model performance. For the second task, we observed that the deep DeepConvNet architecture achieved a higher testing accuracy at around 74.49%, while CRNN only achieved around 65.46%. For the third task, we observed an increase in test accuracy of around 2-5% at time periods of 600, 800, and 1000, corresponding to different models.

1. Introduction

We evaluated the performance of three CNN model architectures and one RNN model architecture on the EEG data from BCI competition. We first experimented with the DeepConvNet architecture provided by the TA, which we believe was originally proposed in [3]. We discovered that the model showed signs of overfitting, therefore we employed regularization techniques such as L2 regularization, increasing dropout strength, adjusted optimizer parameters, and early stopping to improve model generalization. Because the results continued to show signs of overfitting, we switched to ShallowConvNet [3] and later EEGNet, another well-known CNN model architecture for classifying EEG data [2], with the intuition that fewer convolutional layers may work better with our small training dataset. However, we continued to observe overfitting for both models regardless of the further adjustments we made to each architecture, which led us to explore a more direct approach, which is data augmentation.

For all three CNN models, we performed data augmentation referencing code from the TA. Using averaging with added noise and subsampling techniques, we quadrupled our training dataset. In conjunction with data trimming to remove noise data points, we observed a significant improvement over previous iterations of testing for all three architectures.

To further improve test accuracy and with the intuition that RNN architecture should work better with time series data, we tested an RNN architecture, specifically a CNN + Bidirectional LSTM architecture. This architecture models DeepConvNet with the addition of a bidirectional LSTM layer added after four convolutional layers. We used data augmentation and data trimming for this model as well.

2. Methods

2.1. Models

The four models we utilized in this project are ShallowConvNet, EEGNet, DeepConvNet, and CRNN. The detailed architectures of the four models are shown in Appendix B.

2.2. Optimization

We performed two iterations of hyperparameter tuning and model optimization by hand and with KerasTuner—an automatic hyperparameter optimization framework. For automatic hyperparameter optimization, we applied random search, in which not all parameter values are tried out but rather a fixed number of parameter settings is sampled from the specified distributions, as our main searching method, because random search is more computationally efficient, which allows us to expand our searching space given our limited computational resources. For all four models, we conduct searching for learning_rate and weight_decay for the optimizers. For ShallowConvNet and DeepConvNet, dropout_rate is searched; for EEGNet, dropout_rate, kernLength, F1, D, F2, norm_rate are searched; for CRNN, kernel_regularizer, dropout_rate, and recurrent_dropout_rate are searched. The results of both manual and automatic optimizations are shown in Appendix A.

3. Discussion

3.1. Overall observations

During the process of training and optimizing various models, we observed the impact of hyperparameter tuning on inference accuracy. Specifically, we frequently encountered U-shaped validation loss, such as in the CRNN model for task 1, which we resolved by a combination of learning rate annealing and early stopping to reduce overfitting. To further combat overfitting in the CRNN model for task 2, we also increased the dropout rate and added L2 regularization. In addition, we also explored the performance of different optimizers, such as RMSprop and Adagrad, but we did not observe a significant performance increase over Adam.

To account for the reduced dataset size for subject 1 data only, we included an additional iteration of training that combines the train set and validation set into a "final" train set before evaluating the model using the test set. In this setup, we made sure to adjust hyperparameters solely based on validation results and not test results in order to prevent bias.

3.2. Optimize classification accuracy for subject 1

Based on these results, when performing optimization by hand, we observed a clear pattern in the test accuracy between training on subject 1 data and training on all subject data. When performing classification on subject 1 only, training on subject 1 data had a significantly higher accuracy across all four model architectures, despite having nearly 1/10 the training data size. This result led us to conclude that there exists significant differences in the EEG signals of different subjects from this dataset, and this subject difference had a more significant impact on model accuracy than training data size.

Upon performing further optimization using an automated tool, KerasTuner, we observed similar results to those of manual tuning, with an exception being the DeepConvNet model. The DeepConvNet model achieved a higher performance when trained on all subject data compared to only on subject 1 data. We believe this is because DeepConvNet has multiple convolutional layers, which greatly benefit from having a larger training data in order to improve generalization. In the other models, the subject differences may have outweighed the effect of a more generalized and larger training data, resulting in worse performance.

Looking more closely at the performance associated with each model, we observe a more significant increase in test accuracy for auto tuning over manual tuning for training on all subject data, while the test accuracy for training on subject 1 data is not significantly improved and even decreased for DeepConvNet. We hypothesize that these networks have more headroom for improvements when paired with sufficiently large training data; as for training on subject 1 data, due to how small the training data set is, the models could not fully learn the underlying features and thus resulted in relatively less performance increase.

3.3. Optimize classification accuracy for all subjects

In this section, we compared the CRNN mode and the best performing model of the three CNN models, which is the DeepConvNet model. When training and inferencing on all subjects, the DeepConvNet model achieved a higher test accuracy of 73.5% compared to 70% from the CRNN model. This was contrary to our expectation, since our intuition suggested that time series data, such as the EEG data used in this experiment, perform better with RNN models. We believe that this particular data set maybe less compatible with our CRNN model architecture and other post-CNN architectures, such as a conformer architecture, may result in a better performance.

3.4. Determine best time period

To determine the best time period for training, we compared the test accuracy of all four models when trained with data of time periods from [0-300] to [0-1000] in increments of 50. All four models were kept at a similar level of optimization to the best of our ability, with each time period data trained for 50 epochs with data augmentation enabled. The results are shown in Figure 1 of Appendix A.

We observe a clear peak in accuracy at 600 for the three CNN models, 800 for all but ShallowConvNet, and 1000 for ShallowConvNet and EEGNet. By manually inspecting an average of the training data, we were able to determine that the data appears to approximate noise after around 500, which agrees with our experimental findings. That said, the test accuracy improvements are at a maximum only between 2 to 5 percents, which is not overly significant compared to the improvements from hyper-parameter tuning and model optimization.

References

[1] BCI Competition IV. www.bbci.de/competition/iv/. 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 neural network for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 15(5):056013, Jul 2018. 1

[3] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique 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, Aug 2017. 1

A. Performance Report

Table 1. Question 1 (Manual Search)

	T_auc_on_S1	T_auc_on_all
ShallowConvNet	54.00%	36.00%
EGGNet	54.00%	36.00%
DeepconvNet	57.99%	43.99%
CRNN	47.99%	37.99%

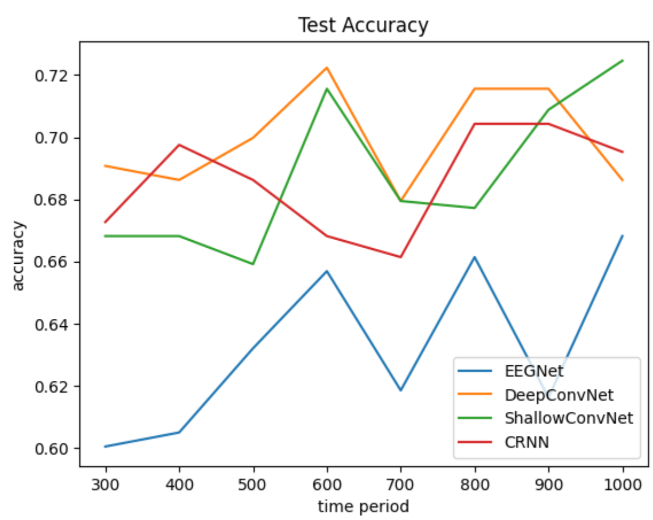
Table 2. Question 1 (KerasTuner Random Search)

	T_auc_on_S1	T_auc_on_all
ShallowConvNet	60.00%	47.99%
EGGNet	57.99%	40.00%
DeepconvNet	54.00%	62.00%
CRNN	47.99%	43.99%

Table 3. Question 2

	Accuracy
DeepconvNet	74.49%
CRNN	65.46%

Figure 1. Question 3



B. Architecture Report

B.1. Data Augmentation

Two data augmentation techniques are used in this project: max pooling and average pooling. For both techniques we applied a window size of 2. For average pooling, we introduced random noises to help reduce overfitting.

B.2. Optimizer

We applied Adam optimizer with loss of categorical_crossentropy and single metric of accuracy. The values for learning_rate and weight_decay are tuned during hyperparameter optimizations.

B.3. Model Architecture

B.3.1 ShallowConvNet

Figure 2. ShallowConvNet Architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 22, 400, 1)]	0
conv2d (Conv2D)	(None, 22, 388, 40)	560
conv2d_1 (Conv2D)	(None, 1, 388, 40)	35200
batch_normalization (BatchNormalization)	(None, 1, 388, 40)	160
activation (Activation)	(None, 1, 388, 40)	0
average_pooling2d (AveragePooling2D)	(None, 1, 51, 40)	0
activation_1 (Activation)	(None, 1, 51, 40)	0
dropout (Dropout)	(None, 1, 51, 40)	0
flatten (Flatten)	(None, 2040)	0
dense (Dense)	(None, 4)	8164
activation_2 (Activation)	(None, 4)	0

B.3.2 EEGNet

Figure 3. EEGNet Architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 22, 400, 1)]	0
conv2d (Conv2D)	(None, 22, 400, 8)	512
batch_normalization (BatchNormalization)	(None, 22, 400, 8)	32
depthwise_conv2d (DepthwiseConv2D)	(None, 1, 400, 16)	352
batch_normalization_1 (BatchNormalization)	(None, 1, 400, 16)	64
activation (Activation)	(None, 1, 400, 16)	0
average_pooling2d (AveragePooling2D)	(None, 1, 100, 16)	0
dropout (Dropout)	(None, 1, 100, 16)	0
separable_conv2d (SeparableConv2D)	(None, 1, 100, 16)	512
batch_normalization_2 (BatchNormalization)	(None, 1, 100, 16)	64
activation_1 (Activation)	(None, 1, 100, 16)	0
average_pooling2d_1 (AveragePooling2D)	(None, 1, 12, 16)	0
dropout_1 (Dropout)	(None, 1, 12, 16)	0
flatten (Flatten)	(None, 192)	0
dense (Dense)	(None, 4)	772
softmax (Activation)	(None, 4)	0

B.3.3 DeepConvNet

Figure 4. DeepConvNet Architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 22, 400, 1)]	0
conv2d (Conv2D)	(None, 22, 396, 25)	150
conv2d_1 (Conv2D)	(None, 1, 396, 25)	13775
batch_normalization (BatchNormalization)	(None, 1, 396, 25)	100
activation (Activation)	(None, 1, 396, 25)	0
max_pooling2d (MaxPooling2D)	(None, 1, 198, 25)	0
dropout (Dropout)	(None, 1, 198, 25)	0
conv2d_2 (Conv2D)	(None, 1, 194, 50)	6300
batch_normalization_1 (BatchNormalization)	(None, 1, 194, 50)	200
activation_1 (Activation)	(None, 1, 194, 50)	0
max_pooling2d_1 (MaxPooling2D)	(None, 1, 97, 50)	0
dropout_1 (Dropout)	(None, 1, 97, 50)	0
conv2d_3 (Conv2D)	(None, 1, 93, 100)	25100
batch_normalization_2 (BatchNormalization)	(None, 1, 93, 100)	400
activation_2 (Activation)	(None, 1, 93, 100)	0
max_pooling2d_2 (MaxPooling2D)	(None, 1, 46, 100)	0
dropout_2 (Dropout)	(None, 1, 46, 100)	0
conv2d_4 (Conv2D)	(None, 1, 42, 200)	100200
batch_normalization_3 (BatchNormalization)	(None, 1, 42, 200)	800
activation_3 (Activation)	(None, 1, 42, 200)	0
max_pooling2d_3 (MaxPooling2D)	(None, 1, 21, 200)	0
dropout_3 (Dropout)	(None, 1, 21, 200)	0
flatten (Flatten)	(None, 4200)	0
dense (Dense)	(None, 4)	16804
activation_4 (Activation)	(None, 4)	0

B.3.4 CRNN

Figure 5. CRNN Architecture

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 250, 1, 25)	13775
max_pooling2d_8 (MaxPooling2D)	(None, 84, 1, 25)	0
batch_normalization_8 (BatchNormalization)	(None, 84, 1, 25)	100
dropout_8 (Dropout)	(None, 84, 1, 25)	0
conv2d_9 (Conv2D)	(None, 84, 1, 50)	31300
max_pooling2d_9 (MaxPooling2D)	(None, 28, 1, 50)	0
batch_normalization_9 (BatchNormalization)	(None, 28, 1, 50)	200
dropout_9 (Dropout)	(None, 28, 1, 50)	0
conv2d_10 (Conv2D)	(None, 28, 1, 100)	125100
max_pooling2d_10 (MaxPooling2D)	(None, 10, 1, 100)	0
batch_normalization_10 (BatchNormalization)	(None, 10, 1, 100)	400
dropout_10 (Dropout)	(None, 10, 1, 100)	0
conv2d_11 (Conv2D)	(None, 10, 1, 200)	500200
max_pooling2d_11 (MaxPooling2D)	(None, 4, 1, 200)	0
batch_normalization_11 (BatchNormalization)	(None, 4, 1, 200)	800
dropout_11 (Dropout)	(None, 4, 1, 200)	0
flatten_4 (Flatten)	(None, 800)	0
dense_6 (Dense)	(None, 40)	32040
reshape_2 (Reshape)	(None, 40, 1)	0
bidirectional_2 (Bidirectional)	(None, 40, 20)	960
time_distributed_2 (TimeDistributed)	(None, 40, 10)	210
flatten_5 (Flatten)	(None, 400)	0
dense_8 (Dense)	(None, 4)	1604