Classification of electroencephalography (EEG) data

Hanfeng Zhong hanfengz0811@ucla.edu
Wayne Zhang waynezhang99@ucla.edu
Wenbo Zhao wenboz@ucla.edu
Yuxin Yin yyxyy999@ucla.edu

Abstract

Machine-learning techniques allow us to extract information from EEG recordings of brain activity. We implemented Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to conduct 4-class classification based on EEG data. We tested various model architectures and training setups, the best model achieves ***71.78% Accuracy*** *using Shallow CNN (SCNN) on the test set. We studied how different subjects and sampling time windows affect the model performance. Data preprocessing techniques including low pass filter (LPF), cropping and normalization are also adopted.*

1. Introduction

The EEG dataset contains synchronized activity of millions of neurons close to a scalp electrode. In BCI Competition 2008 [1], researchers recorded EEG data from 9 subjects while they imagine performing four different actions. The original dataset contains 2115 trials, each with electrical signal from 22 electrodes over a 1000-bin time period. Due to the small number of trials and inherent noise within the data, it is challenging to create a model that does not overfit during training. In this article, we implemented several CNN and RNN architecture to achieve the best testing accuracy on the EEG dataset.

1.1. Shallow Convolution Network

We implemented two Shallow CNN (SCNN) in this project, both yielding an accuracy higher than 0.7. We used SCNN here since there are only 2115 trials in the BCI dataset, with a 0.2 validation split, we only have 1692 trials for the training process. A deep convolution network could be more vulnerable to overfitting on such a small dataset.

The architecture for the two SCNNs is shown in Figure 1. Both SCNNs implement the first CNN layer using 40 filters with size 1×25. Here our goal is to extract time series features related to each separate channel.

For our first SCNN model, we permuted and reshaped the 3D kernel from Layer one to a 2D plain, with 22×40 features, each feature with a slightly reduced feature size. We then used a fully connected (FC) network to extract 40 features from the previous output. This step is critical as we learn the correlations between different channels, information embedded within their relative spatial positions can then be extracted. Finally, after math layers like `square` and `log` we use the last FC layer and softmax function to generate the scores for each of the four directions. This architecture is inspired by the prior research on SCNN. [4]

The second SCNN uses another CNN layer after the first one to further extract spatial features. After the two CNN layers, a squaring non-linearity, a mean pooling layer, and a logarithmic activation function followed. As the first SCNN, this model also has several pooling regions within one trial, enabling it to learn a temporal structure of the band power changes within the trial. [3]

1.2. Deep Convolution Network

We also implemented two Deep CNN (DCNN) in TensorFlow and Keras consists of five convolutional layers (one temporal and four spatial) and one fully-connected layer. After each convolutional layer, we inserted a batch normalization layer and a ELU activation layer. To increase the receptive fields in the convolutional layers, we introduced a 2x2 max pooling layer to increase the performance of validation accuracy in the EEG dataset. Finally we applied a softmax layer to obtain the classification scores for the four labels.

With more non-linear layers, DCNN tends to learn more information from the dataset and thus yielding a higher training accuracy. Similar to GoogLeNet [5], we used 3×3 filters across multiple layers. However, DCNN also suffers from problems like vanishing or exploding gradients, which severely degrade its learning capability. Our remedy to this problem in the second DCNN is to add a gradient highway like ResNet [2] which increases the possibility of the appearance of an identity layer, thus allowing the model to outperform SCNNs. However, in practice, SCNN converges faster and yields a higher test accuracy while trained with limited number of epochs. The architecture for the better DCNN is shown in Figure 1.

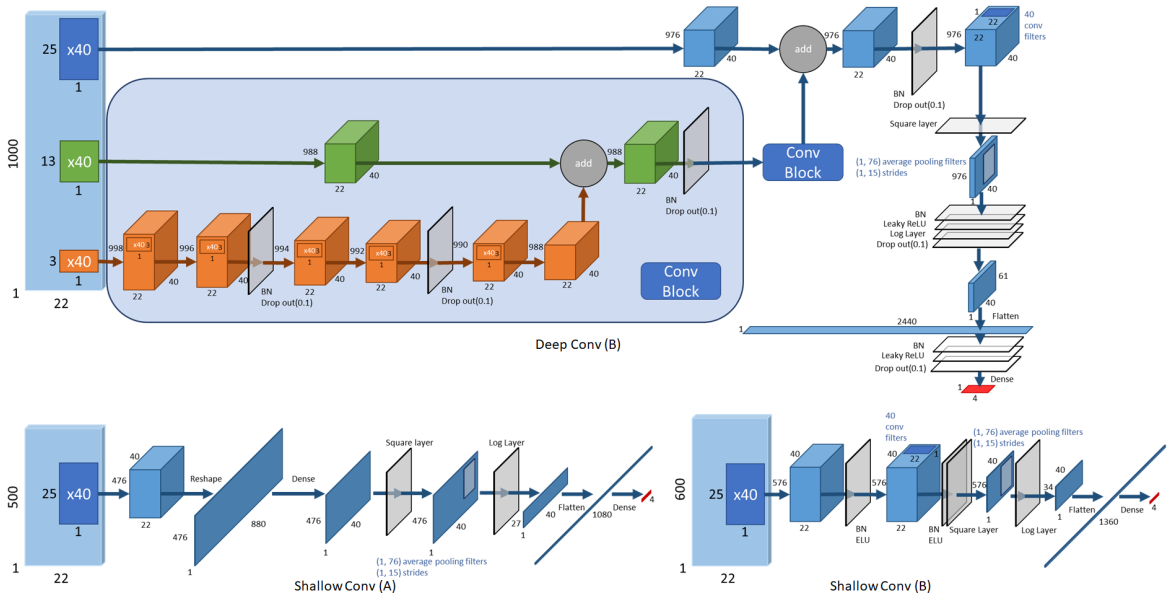

Figure 1. Model Architecture

1.3. RNN/LSTM

Theoretically, RNN can be applied to this dataset for the dataset is time-related and RNN is designed to focus on sequential data. However, RNN is excellent at predicting the next scenario not classifying the sequential signal, it is hard to obtain high accuracy by RNN. Meanwhile, RNN has fundamental vanishing and exploding gradient issues. Hence instead of using RNN, LSTM (Long Short Term Memory), which is designed to overcome the gradient-related issue, is applied to our project. Through experiment, we built a network with three bidirectional LSTM layers, but that network only can gain 50% accuracy. With the experience in CNN network, the low accuracy of the LSTM network might be caused by the overfit. Therefore, we simplified the network into a shallow one by only applying one bidirectional layer. But the accuracy is also around 50%. Consequently, we wish by combining CNN with LSTM, the accuracy of the network will improve. We established a CNN network before the LSTM network to provide a blurry classification and applied a one layer bidirectional LSTM after the CNN network. Unfortunately, the improvement is not as good as we expected. The accuracy only reached to 54.8%.

2. Results

2.1. Training Setup and Data Pre-processing

Before the training process, we have to look at the dataset carefully and get familiar with its property of it. By averaging the signal in the same classification and plotting the average in the time domain. From appendix Figure 5, we could find that there is no more efficient information after a specific time. Hence, we could crop the signal and only train the useful information in the dataset to guarantee the efficiency of our model. Moreover, if we look at the reciprocal domain of the signal in Figure 5, most of the energy is contained in the low frequency. As a result, we could utilize a low pass filter to eliminate the high frequent thermal noise. By comparing the signal before and after the filter, it is clear in Figure 5 that the signal becomes more smooth and the noise inside the signal influences the training and classifying process less than without the filter.

Additionally, there are only 2115 training data contained in the dataset, which is pretty small for training a model to accomplish the classification task. Because of this fact, our team applied a data augmentation technique to enlarge the training dataset. The methodology our team utilized is to add additional Gaussian noise into the original signal. And by doing this augmentation, the training dataset was doubled. We only applied this augmentation when training the Deep ConvNet(B) for this method largely increases the test accuracy.

2.2. Comparison between training on Subject 1 and across all subjects

The EEG Dataset contains 9 training subjects which refer to 9 people who help to collect the data. We trained our model with different training strategies regarding subject 1 to see if one single sample person can affect the model performance. Specially, we extracted all data belonging to Subject 1 and used it to train the Shallow ConvNetB for 35

epochs, tested it on the subject 1 data and the whole dataset. We also used the aforementioned model as the pre-trained model and trained it on the whole dataset for another 40 epochs. Results are shown in Table 1

Train Set	Test Set	Test accuracy
Subject1	Subject1	0.56
Subject1	All Data	0.37
Sunject1-Pretrain + All Data	All Data	0.38
All Data	All Data	0.71

Table 1. Strategies regarding Subject 1 and Test Accuracy

2.3. Comparison on Accuracy across all models

During the training experiment, we tested how the data preprocessing affects the test accuracy. The result of SCNN(A) with and without the low pass filter data preprocessing were both collected by our team. We plotted the confusion matrix of both SCNN(A) respectively in Figure 3 and Figure 4. The experiment result shows that high-frequency noise doesn't influence the training process much for the accuracy of these two experiments can reach to 70% shown in Table 2. This result might lead us to the conclusion that the deep-learning method can automatically eliminate the influence of noise without the help of an additional filter.

The difference between applying Batch Normalization layers and not applying them is included in our team consideration as well. Shown in Table 2. When we utilized the BN layers, the improvement in accuracy is obvious. The efficiency of BN layers acting like a regularization is high in the SCNN(B)

The results of the nine proposed networks are shown in Table 2. The architecture of the Shallow ConvNet(A), Shallow ConvNet(B) and Deep ConvNet(B) are plotted in Figure 1. To elaborate each network we used in the research, the rest of architectures are shown in the appendix.

2.4. Comparison on Accuracy of different Time Periods

In experiments, we crop the dataset on the dimension of Time Periods to demonstrate how much time is needed to get a reasonable classification accuracy and robust model. We tested 10 different time lengths, the results are shown in Figure 2. It indicates that the model achieved the best performance at the time periods of 600, and the performance basically remains stable if time periods are larger than 300.

3. Discussion

3.1. Analysis about data strategy

It can be seen from the Table 1 that the classification performance are significantly superior when the model is trained on the whole dataset. This makes sense since the EEG Dataset only has 237 Subject 1 samples. The model is more likely to perform overfitting when the number of data is small.

Another interesting fact is that if we only use one single subject to pre-train the model, the performance will drop off as well. Considering the data are collected with electrodes on different person, training or pre-training on one single subject can make the model to be less generalized since the signal of different people could be very various. Thus, the model will be more robust if we trained with the data collected from more people.

Method	Accuracy
Shallow ConvNet(A without filter)	0.7111
Shallow ConvNet(A with filter)	0.7065
Shallow ConvNet(B with BN)	0.7178
Shallow ConvNet(B without BN)	0.6524
Deep ConvNet(A)	0.5782
Deep ConvNet(B)	0.6568
Shallow LSTM	0.4605
Deep LSTM	0.5011
CNN LSTM	0.5485

Table 2. Test Accuracy across All Models

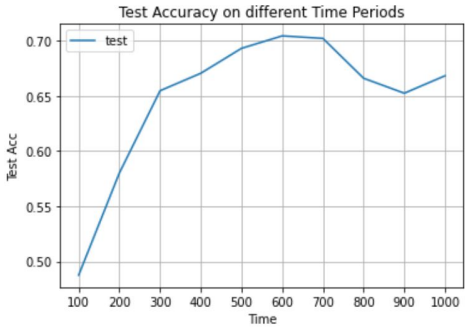

Figure 2. Time Periods vs Test Accuracy

3.2. Analysis about Time Periods of the data

Figure 2 demonstrates that the longer time period does not necessarily yield better performance, the performance becomes stable after the time period is larger than 300. The corresponding data visualization Figure 5 also indicates that as the Time Period increases, more data becomes noisy that brings less useful information for classification. One reasonable explanation can be the error accumulation of instruments for collecting the data. Thus, we cropped the data and applied only the first 600 times to train the model.

References

[1] Clemens Brunner, Robert Leeb, Gernot Müller-Putz, Alois Schlögl, and Gert Pfurtscheller. Bci competition 2008–graz data set a. *Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology*, 16:1–6, 2008. 1

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1

[3] 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, 2018. 1

[4] 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

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. 1

Appendix

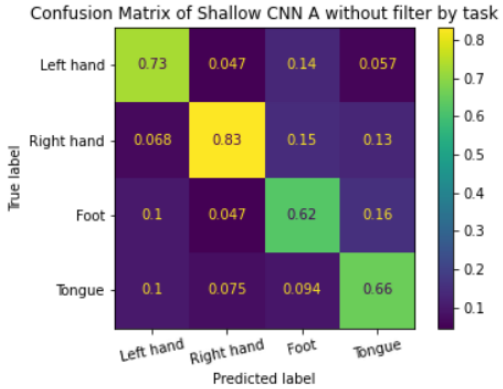

Figure 3. Confusion Matrix for Shallow CNN(A without filter)

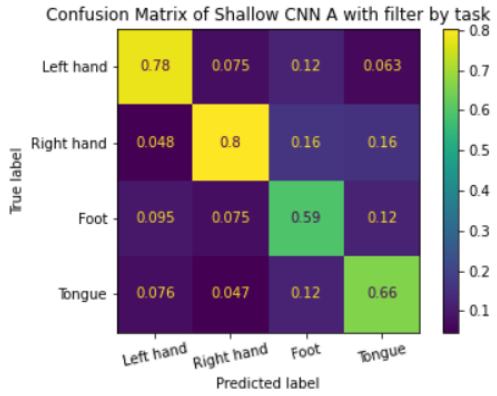

Figure 4. Confusion Matrix for Shallow CNN(A with filter)

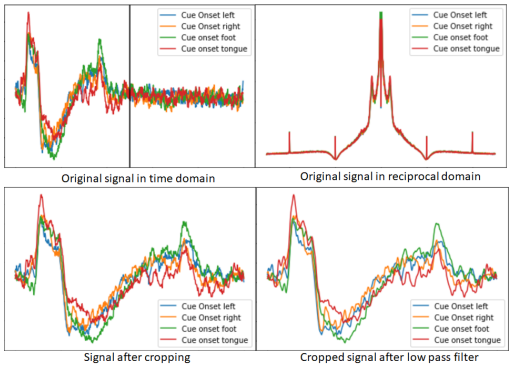

Figure 5. Data Visualization

Layer (type)	Output Shape	Param #
EEG (InputLayer)	[(None, 22, 500, 1)]	0
Temp-conv-side-1-2 (Conv2D)	(None, 22, 476, 40)	1040
permute-0 (Permute)	(None, 476, 22, 40)	0
tf.reshape_6 (TFOpLambda)	(None, 476, 880)	0
dense_6 (Dense)	(None, 476, 40)	35240
bidirectional_3 (Bidirectional)	(None, 476, 32)	7296
tf.math.square_3 (TFOpLambda)	(None, 476, 32)	0
tf.reshape_7 (TFOpLambda)	(None, 1, 476, 32)	0
Ave-Pool (AveragePooling2D)	(None, 1, 27, 32)	0
tf.math.log_3 (TFOpLambda)	(None, 1, 27, 32)	0
flatten_3 (Flatten)	(None, 864)	0
dense_7 (Dense)	(None, 4)	3460

Total params: 47,036
Trainable params: 47,036
Non-trainable params: 0

Figure 6. Architecture of CNN LSTM

Layer (type)	Output Shape	Param #
EEG (InputLayer)	[(None, 22, 1000)]	0
permute-0 (Permute)	(None, 1000, 22)	0
bidirectional (Bidirectional)	(None, 1000, 64)	14080
BN-1 (BatchNormalization)	(None, 1000, 64)	256
leaky_re_lu (LeakyReLU)	(None, 1000, 64)	0
Dropout-1 (Dropout)	(None, 1000, 64)	0
flatten (Flatten)	(None, 64000)	0
BN-3-1 (BatchNormalization)	(None, 64000)	256000
leaky_re_lu_1 (LeakyReLU)	(None, 64000)	0
Dropout-3-1 (Dropout)	(None, 64000)	0
dense (Dense)	(None, 100)	6400100
BN-4-1 (BatchNormalization)	(None, 100)	400
leaky_re_lu_2 (LeakyReLU)	(None, 100)	0
Dropout-4-1 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 4)	404

Total params: 6,671,240
Trainable params: 6,542,912
Non-trainable params: 128,328

Figure 7. Architecture of Shollow LSTM

Layer (type)	Output Shape	Param #
EEG (InputLayer)	[(None, 1000, 22, 1)]	0
conv11 (Conv2D)	(None, 1000, 22, 50)	300
conv12 (Conv2D)	(None, 1000, 22, 50)	55050
BN-11 (BatchNormalization)	(None, 1000, 22, 50)	200
ELU-11 (ELU)	(None, 1000, 22, 50)	0
MP-11 (MaxPooling2D)	(None, 500, 22, 50)	0
dp-11 (Dropout)	(None, 500, 22, 50)	0
conv21 (Conv2D)	(None, 500, 22, 100)	25100
BN-21 (BatchNormalization)	(None, 500, 22, 100)	400
ELU-21 (ELU)	(None, 500, 22, 100)	0
MP-21 (MaxPooling2D)	(None, 250, 22, 100)	0
dp-21 (Dropout)	(None, 250, 22, 100)	0
conv31 (Conv2D)	(None, 250, 22, 200)	100200
BN-31 (BatchNormalization)	(None, 250, 22, 200)	800
ELU-31 (ELU)	(None, 250, 22, 200)	0
MP-31 (MaxPooling2D)	(None, 125, 22, 200)	0
dp-31 (Dropout)	(None, 125, 22, 200)	0
conv41 (Conv2D)	(None, 125, 22, 200)	200200
BN-41 (BatchNormalization)	(None, 125, 22, 200)	800
ELU-41 (ELU)	(None, 125, 22, 200)	0
MP-41 (MaxPooling2D)	(None, 62, 22, 200)	0
dp-41 (Dropout)	(None, 62, 22, 200)	0
flatten_11 (Flatten)	(None, 272800)	0
fully-connected-1 (Dense)	(None, 32)	8729632
class-probs (Dense)	(None, 4)	132

Total params: 9,112,814
Trainable params: 9,111,714
Non-trainable params: 1,100

Figure 8. Architecture of Deep CNN

Layer (type)	Output Shape	Param #
EEG (InputLayer)	[(None, 22, 500)]	0
permute-0 (Permute)	(None, 500, 22)	0
bidirectional_27 (Bidirectional)	(None, 500, 32)	4992
BN-1 (BatchNormalization)	(None, 500, 32)	128
elu_45 (ELU)	(None, 500, 32)	0
Dropout-1 (Dropout)	(None, 500, 32)	0
bidirectional_28 (Bidirectional)	(None, 500, 64)	16640
BN-2 (BatchNormalization)	(None, 500, 64)	256
elu_46 (ELU)	(None, 500, 64)	0
Dropout-2 (Dropout)	(None, 500, 64)	0
bidirectional_29 (Bidirectional)	(None, 500, 128)	66048
BN-3 (BatchNormalization)	(None, 500, 128)	512
elu_47 (ELU)	(None, 500, 128)	0
Dropout-3 (Dropout)	(None, 500, 128)	0
flatten_9 (Flatten)	(None, 64000)	0
BN-3-1 (BatchNormalization)	(None, 64000)	256000
elu_48 (ELU)	(None, 64000)	0
Dropout-3-1 (Dropout)	(None, 64000)	0
dense_18 (Dense)	(None, 100)	6400100
BN-4-1 (BatchNormalization)	(None, 100)	400
elu_49 (ELU)	(None, 100)	0
Dropout-4-1 (Dropout)	(None, 100)	0
dense_19 (Dense)	(None, 4)	404

Total params: 6,745,480
Trainable params: 6,616,832
Non-trainable params: 128,648

Figure 9. Architecture of Deep LSTM