EEG 4-class Motor Classification using Convolutional Neural Networks Cascaded with Recurrent Neural Networks

Terri Tsai
UID: 805624292
tctsai@g.ucla.edu

Abstract

In this project, I explored the performances of different Neural Network architectures. I investigated the effects of cascading recurrent layers after convolutional layers, and the effects that windowing data augmentation techniques have on these structures. This report will show that a 70% all subject classification accuracy can be achieved with both CNN and CNN+LSTM networks given the appropriate data augmentation technique.

1. Introduction

Given the 22×1000 data structure and only a mere 2115 samples of training data, notice that first, our neural network architecture needs to be able to extract features in both space and time, and second, that the quantity of training data is small, and ought to be compensated for somehow.

1.1. Network Architecture

Convolutional Neural Networks in the past have proved to perform well on this EEG dataset, but we also know that Recurrent Neural Networks such as the Vanilla RNN, GRU, and LSTM, are commonly used in learning temporal patterns. My aim for this project was to design structures that have CNN cascaded with LSTMs, and to observe how adding a recurrent neural network to a convolutional neural network affects the pefromance of the original CNN structure. I wanted to know if the additional complexity necessarily aids the network in learning, and if so, under what circumstances?

There are two primary designs: DeepConv and Conv1. DeepConv is adapted from Schirrmeister *et al*. [1], and its LSTM-cascaded partner, DeepConv2, shares the same convolutinal layers but has an additional three LSTM layers at at the end. Conv1 is another CNN network; it takes inspiration from DeepConv, but has two less convolutional blocks, an added activation function in the first convolutional block to make up for the subtracted convolutional blocks just mentioned, and larger MaxPooling filters. Its LSTM-cascaded partner, Conv2, has two LSTMs at the end. Structure details can be seen in Tables 1- 4.

1.2. Data Augmentation

Next, we have to address the fact that training on this dataset is prone to overfitting due to the small number of examples and the large number of features. Large features typically require a more complicated network, but increasing the complexity of the network makes the trained network less likely to be general enough to receive new data. Finding a balance between learning and being general is even more difficult with a small number of training samples, thus it is clear that data augmentation is needed to increase performance. My aim was to explore the effects of cropping the given data in order to produce more samples. The goal was to investigate the optimal crop size and at what stride to take the multiple crops.

2. Results

During the investigation process, networks were tested under different hyperparameters. For the purpose of final comparison, the four network architectures were all trained and tested at the end under the same conditions: Adam optimizer with 1e-5 learning rate and 0.001 weight decay, a more aggressive dropout rate of 0.5, and cropped at 900 time bins with stride of 10. These hyperparameters were chosen through empirical observation that structures with LSTMs attached perform better with low weight decay, and that generally higher dropout values result in better generalization. To simulate the early stop mechanism, the final model used for test evaluation was determined to be the model that resulted in the highest validation accuracy during 200 training epochs. DeepConv achieved 70.14% classification accuracy, DeepConv2 achieved 68.66%, Conv1 achieved 68.19%, and Conv2 achieved 70.12%. Training was done on Google Colab. Note a 5-fold cross validation process was included in my code, which for each test ran, would give us an average test accuracy of five training runs.

However this was not implemented due to the usage limits of Colab. Additionally, based on obesrvations that will be mentioned in the Discussion section, I believe a smaller stride of 5 may achieve higher accuracy, but the RAM limits of Colab allows for only 10.

3. Discussion

3.1. Model Architecture

While the DeepConv-DeepConv2 pair and the Conv1-Conv2 pair both contain a CNN structure and a matching CNN+LSTM structure, their performances differ.

3.1.1 DeepConv vs DeepConv2

DeepConv was observed to perform better generally when using full trial lengths and larger windows, compared to its LSTM-cascaded counterpart DeepConv2. This could be because DeepConv already performs well. With its many layers, DeepConv likely already extracts important spatial and temporal features, thus adding the LSTM was an overkill, causing DeepConv2 be more over-fitted.

However, as seen in Figure 1, DeepConv2 does start to perform better when the trials are cropped to be around 500 to 600 or smaller. This suggests that there may be properties in the additional recurrent layers that are able to extract more information given shorter trial lengths. This hypothesis is in line with my early architecture investigations, as I discovered that RNN structures with no convolutional layers in front performed very poorly on the original full-sized dataset, achieving around 30% or less.

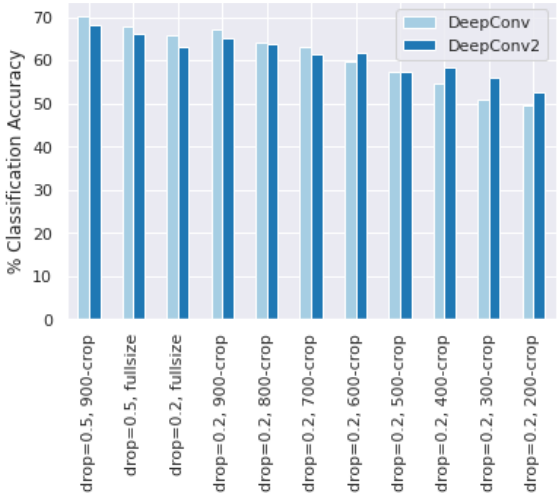

Figure 1: All Subject Classification Accuracy Comparison between DeepConv and DeepConv2 across different hyperparameters.

3.1.2 Conv1 vs Conv2

The opposite pattern was observed in the Conv1-Conv2 pair. Conv2's addition of LSTM seemed to be helpful, observed in the final test, and throughout most stages of investigation. This could be because Conv1 was not fully extracting information due to its shallower nature, thus adding the LSTM assisted in further learning.

It is worth noting that although a possibility, we cannot be certain that it is exactly the LSTM that allows Conv2 to out-perform Conv1, but we can postulate that the added LSTM added just enough complexity to balance learning and generalizing, while the LSTM added in DeepConv2 was too much.

3.2. Data Augmentation

Before discussion of the effects of stride and window length in cropping the trials, it is important to note that I have chosen to augment both the training and test dataset. Accuracy is computed on the test set by treating each crop as its individual sample. Alternatively, a "voting method" was considered, and implemented in code, where an original test sample's crops are not treated as individual samples, but rather the most popular label prediction amongst the crops is assigned to their original full-size test sample as its prediction label. This voting method would also make the cropping method an ensemble method, but ultimately was not pursued due to the long validation and test time that looping through each original sample one by one would take. Also note that during the investigation phase, a more moderate dropout value was used to evaluate the effects of stride and windowing, so the accuracy in these investigations ought not to be directly compared to the final test.

3.2.1 Stride

In testing trial lengths of 500 time bins with varying strides on Conv1 and Conv2, we observe that the test accuracy increases with smaller strides, as seen in Figure 2. This is most likely because smaller strides produces more crops, increasing the number of training samples. When training Neural Networks, it is usually better to have more data, but a concern was that small strides would cause too much overlapping between crops, and that the close correlation between the samples would be detrimental, so this stride test needed to be done. Note the stride test was done before window test, and because cropping increases the dataset and therefore the train time, I chose to implement the stride test on the Conv1-Conv2 pair due to their shorter training times.

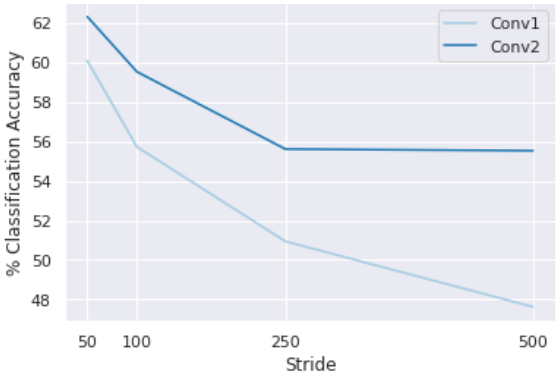

Figure 2: All Subject Classification Accuracy of Conv1 and Conv2, cropped at 500 time bins, across different strides.

3.2.2 Window Length

Since the above concluded small strides were beneficial, small strides were also implemented on DeepConv and DeepConv2. To keep the tests consistent, $stride = 0.1 * windowsize$ for all window sizes, so that the percentage of overlapping time bins with respect to the window size is consistent across all window sizes. Note that DeepConv and DeepConv2 had to be altered for small window sizes 400 and under, and I did so by removing the forth convolutional block. As seen in Figure 3, larger windows produced better results, suggesting that crucial information spans longer time lengths, or that the networks are too complex for smaller crops and thereby causing over-fitting.

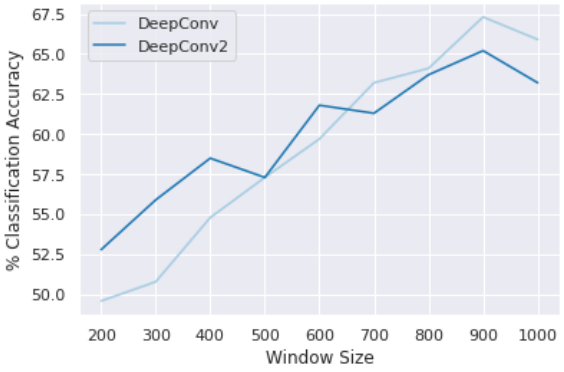

Figure 3: All Subject Clasificaiton Accuracy of DeepConv and DeepConv2 across different window sizes.

3.2.3 Individual Subject Accuracy

The four models optimized for all-subjects classification was used to train and test individual subject accuracies. As seen in Table 1 under Performance Summary, the individual subject accuracies were quite inconsistent across different subjects. With the exception of subject 5, 7, and 9, none of the four models could get a better accuracy score on training just individual subjects compared to training with all subjects. This intuitively makes sense because given 2115 samples is already very little, training deep networks on a even smaller subset of data is more challenging. The benefit of training across all subjects is that the network would be able to learn more features and be more general for incoming unseen test samples. An interesting note is that across the individual subjects, DeepConv2 is often the worst performing one, which is in line with our above speculations that DeepConv2 may be too complex of a structure.

References

[1] R. Schirrmeister, J. Springenberg, L. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human eeg. 2018.

4. Performance Summary

Table 1: % Classification Accuracy on Final Test

	DeepConv	DeepConv2	Conv1	Conv2
Subject 1	45.09	42.18	60.36	66.36
Subject 2	48.73	30.91	45.45	52.55
Subject 3	54.18	55.82	66.18	77.45
Subject 4	18.18	14.73	44.91	36.73
Subject 5	63.44	60.93	74.08	65.96
Subject 6	44.71	35.99	59.93	40.82
Subject 7	61.27	46.73	73.09	77.45
Subject 8	45.45	40.00	61.45	69.27
Subject 9	72.15	62.67	79.88	79.30
All Subjects	70.14	68.16	68.19	70.12

Hyperparameters used on are those specificed in Results section.

5. Network Architecture

Table 2: DeepConv(Small) Network Architecture

(Pytorch) Function	Parameters	Note
Conv2d	kernel $(1 \times 10) \times 10$	-
BatchNorm2d	-	-
Conv2d	kernel $(22 \times 1) \times 25$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×3)	-
Dropout	0.5	-
Conv2d	kernel $(1 \times 10) \times 50$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×3)	-
Dropout	0.5	-
Conv2d	kernel $(1 \times 10) \times 100$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×3)	-
Dropout	0.5	1
Conv2d	kernel $(1 \times 10) \times 200$	1
BatchNorm2d	-	1
ELU	-	1
MaxPool2d	kernel (1×3)	1
Linear	Input: $inputdim$, Output: 4	2

Note 1: Remove when implementing DeepConv with input trial sizes 200-400.
Note 2: "inputdim" is dependent on input trial size.

Table 3: DeepConv2(Small) Network Architecture

(Pytorch) Function	Parameters	Note
Conv2d	kernel $(1 \times 10) \times 10$	-
BatchNorm2d	-	-
Conv2d	kernel $(22 \times 1) \times 25$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×3)	-
Dropout	0.5	-
Conv2d	kernel $(1 \times 10) \times 50$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×3)	-
Dropout	0.5	-
Conv2d	kernel $(1 \times 10) \times 100$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×3)	-
Dropout	0.5	
Conv2d	kernel $(1 \times 10) \times 200$	1
BatchNorm2d	-	1
ELU	-	1
MaxPool2d	kernel (1×3)	1
Dropout	0.5	1
LSTM	64 hidden units, bidirectional	-
LSTM	32 hidden units, bidirectional	-
LSTM	32 hidden units, bidirectional	-
Linear	Input: 32, Output: 4	-

Note 1: Remove when implementing DeepConv2 with input trial sizes 200-400.

Table 4: Conv1 Network Architecture

(Pytorch) Function	Parameters	Note
Conv2d	kernel $(1 \times 10) \times 16$	-
BatchNorm2d	-	-
ELU	-	-
Conv2d	kernel $(22 \times 1) \times 32$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×6)	-
Dropout	0.5	-
Conv2d	kernel $(1 \times 10) \times 64$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×6)	-
Dropout	0.5	-
Linear	Input: $inputdim$, Output:832	1
BatchNorm1d	-	-
RELU	-	-
Dropout	0.5	
Linear	Input: 832, Output: 416	-
BatchNorm1d	-	-
RELU	-	-
Linear	Input: 416, Output: 4	-

Note 1: "inputdim" is dependent on input trial size

Table 5: Conv2 Network Architecture

(Pytorch) Function	Parameters	Note
Conv2d	kernel $(1 \times 10) \times 16$	-
BatchNorm2d	-	-
ELU	-	-
Conv2d	kernel $(22 \times 1) \times 32$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×6)	-
Dropout	0.5	-
Conv2d	kernel $(1 \times 10) \times 64$	-
BatchNorm2d	-	-
ELU	-	-
MaxPool2d	kernel (1×6)	-
Dropout	0.5	-
LSTM	64 hidden units	-
LSTM	32 hidden units	-
Linear	Input: 32, Output: 4	-