Lab2 Report

0856091 蘇邱弘

July 2020

1 Introduction

Brain-Computer Interfaces (BCI) are systems that can translate the brain activity patterns of a user into messages or commands for an interactive application. The brain activity which is processed by the BCI systems is usually measured using Electroencephalography (EEG). In this lab, I implement two CNN models with three different activation functions to do the classification on EEG-data.

In this report, I will first introduce the experiment setups, including the details of the two CNN models and three kinds of activation function used in the neural networks. Next, I will show my testing results and comparison figures of two models with different activation functions. Finally, I will do some discussions.

2 Experiment Setups

In this lab, I implement two CNN models: EEGNet and DeepConvNet, with three different activation functions: ReLU, Leaky ReLU, and ELU. The models' architecture I implement are the same as the ones described in the spec, except for the activation function and the dropout ratio in the dropout layer.

In the initial construction step of the neural network, we can specify the activation function and the dropout ratio used in the model. I set the dropout ratio to 0.5 for both models, which get a better performance comparing to other dropout ratios (the comparing results are in section 4).

In the following sections, I will further introduce the details of the two models and the three activation functions used in the models.

2.1 Model Detail

2.1.1 EEGNet

EEGNet is a compact convolutional neural network for EEG-based BCIs. A visualization of the EEGNet model is shown in Figure 1, and the implementation detail (which use ReLU as the activation function and dropout ratio = 0.5) is shown in Figure 2. The following are descriptions of each layer:

- The convolution layer outputs feature maps containing the EEG signal at different band-pass frequencies.
- A depthwise convolution, which connected to each feature map individually, is used to learn frequency-specific spatial filters.
- The separable convolution is a combination of a depthwise convolution, which learns a temporal summary for each feature map individually.

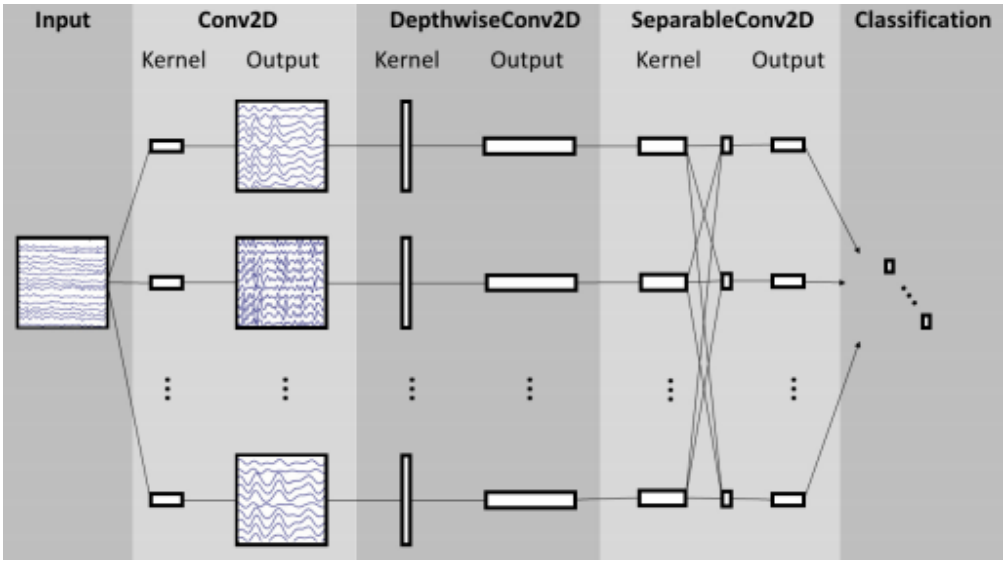


Figure 1: Overall visualization of the EEGNet architecture

```
EEGNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False)
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (depthwiseConv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.5, inplace=False)
  )
  (separableConv): Sequential(
    (0): Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

Figure 2: Implementation detail of EEGNet

2.1.2 DeepConvNet

The DeepConvNet architecture consists of five convolutional layers with a softmax layer for classification. Comparing to EEGNet, which was mainly designed for EEG-based BCIs, DeepConvNet was designed to be a general-purpose architecture that is not restricted to specific feature types. The implementation detail (which use ReLU as the activation function and dropout ratio = 0.5) is shown in Figure 3.

```
DeepConvNet(
  (conv_1): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU()
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (conv_2): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv_3): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv_4): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

Figure 3: Architecture of DeepConvNet

2.2 Activation Function

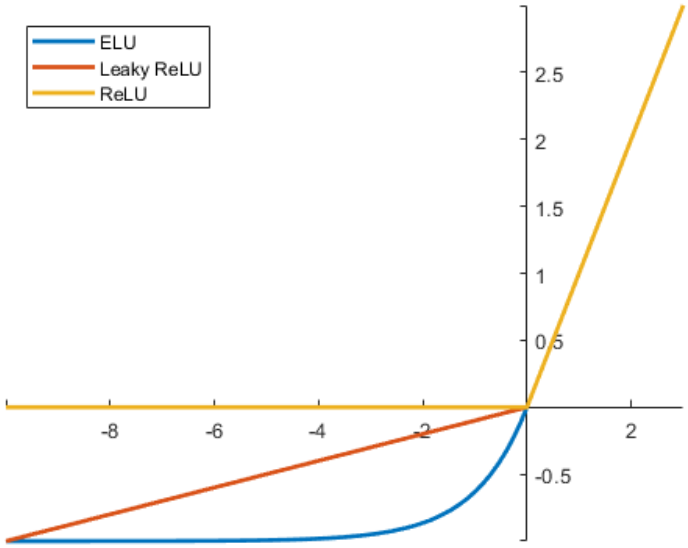


Figure 4: Activation functions

2.2.1 ReLU

ReLU is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. The formula of ReLU is simple:

$$ReLU(x) = max(0, x)$$

ReLU has several advantages when comparing to sigmoid and tanh. For example, it is easier to compute, can avoid the problem of gradient vanishing, and converge faster. But ReLU has some downsides. In the region $x < 0$ of ReLU, gradient will be 0 because of which the weights will not get adjusted during gradient descent. That means, those neurons which go into that state will stop responding to variations in error/input. The problem is known as the "Dead ReLU" problem.

2.2.2 Leaky ReLU

Leaky RelU is a variant of ReLU. Instead of being 0 when $x < 0$, Leaky ReLU allows a negative slope α (which is set to 0.01 by default in PyTorch framework). The negative slope is added in order to fix the "Dead ReLU" problem. But in practice, there is no evidence that Leaky ReLU will perform better than ReLU. The formula of Leaky ReLU is:

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha \times x, & \text{otherwise} \end{cases}$$

2.2.3 ELU

Exponential Linear Unit (ELU) is another variant of ReLU. Unlike to ReLU, ELU becomes smooth slowly until its output equal to $-\alpha$ (which is set to 1.0 by default in PyTorch framework). ELU is also designed for solving the "Dead ReLU" problem, but just like Leaky ReLU, there is no evidence that ELU will perform better than ReLU neither. The formula of ELU is:

$$ELU(x) = max(0, x) + min(0, \alpha \times (exp(x) - 1))$$

3 Experimental Results

The followings are the hyper-parameters I used in both models:

- batch size: 128
- epochs: 800
- learning rate: 0.001
- optimizer: Adam
- loss function: cross entropy

3.1 Highest Testing Accuracy

Figure 5 and 6 are the screenshots of the highest accuracy of two architectures with three kinds of activation functions, and Table 1 shows the overall comparison results of the accuracy. As shown in the table, models using ReLU and Leaky ReLU have similar performance, and get a better result than models using ELU. And also, we can see that EEGNet outperform DeepConvNet for all activation functions.

```
100%|##########| 800/800 [02:00<00:00,  6.63 epochs/s, train_loss=0.0837, train_acc=96.85, test_loss=0.3168, test_acc=88.43]
```

(a) ReLU

```
100%|##########| 800/800 [02:01<00:00,  6.60 epochs/s, train_loss=0.0935, train_acc=96.85, test_loss=0.3566, test_acc=87.96]
```

(b) Leaky ReLU

```
100%|##########| 800/800 [02:00<00:00,  6.61 epochs/s, train_loss=0.1153, train_acc=94.54, test_loss=0.4479, test_acc=85.65]
```

(c) ELU

Figure 5: Screenshots of EEGNet's results

```
100%|##########| 800/800 [03:36<00:00,  3.69 epochs/s, train_loss=0.0343, train_acc=98.80, test_loss=0.7290, test_acc=82.13]
```

(a) ReLU

```
100%|##########| 800/800 [03:36<00:00,  3.69 epochs/s, train_loss=0.0356, train_acc=98.70, test_loss=0.6808, test_acc=82.22]
```

(b) Leaky ReLU

```
100%|##########| 800/800 [03:37<00:00,  3.68 epochs/s, train_loss=0.0389, train_acc=98.98, test_loss=1.3174, test_acc=80.65]
```

(c) ELU

Figure 6: Screenshots of DeepConvNet's results

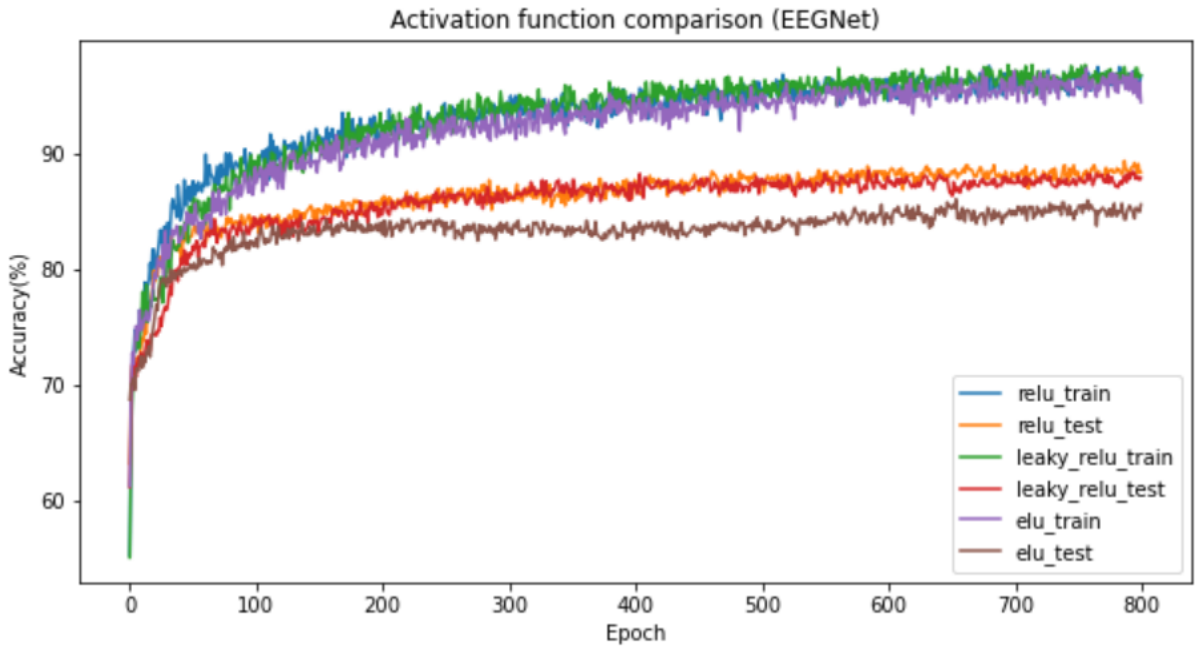
	EEGNet	DeepConvNet
ReLU	88.43%	82.13%
Leaky ReLU	87.96%	82.22%
ELU	85.65%	80.65%

Table 1: Accuracy comparison

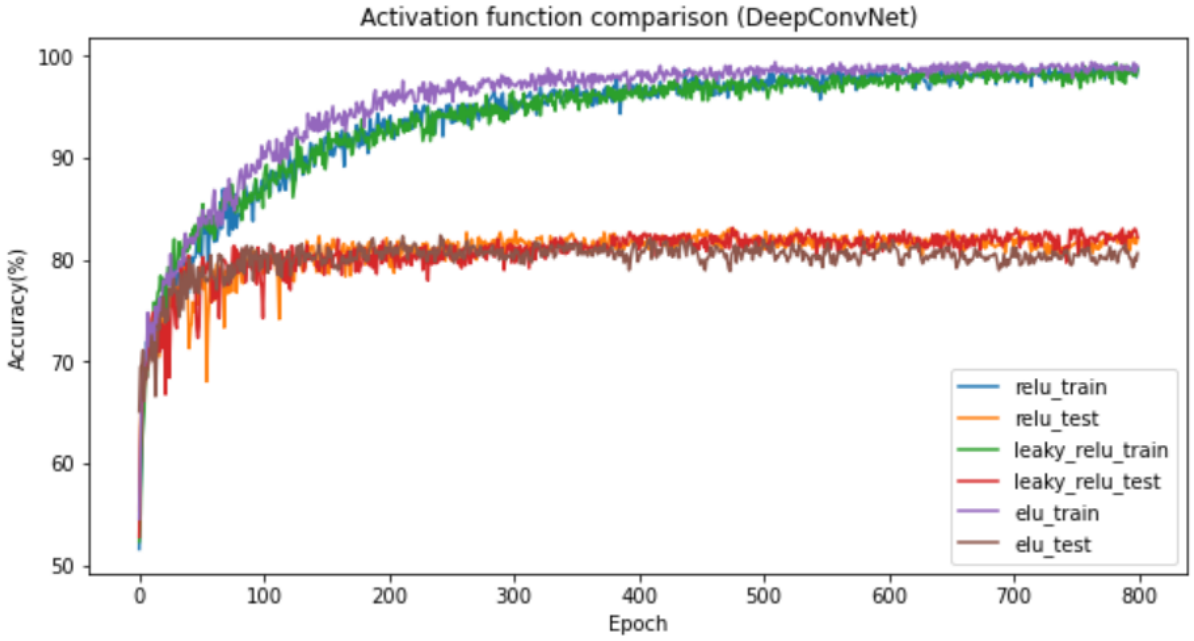
3.2 Comparison Figures

Figure 7 shows the comparison figures of the two models with three kinds of activation functions. As shown in Figure 7(a), for EEGNet, ReLU and Lealy ReLU converge faster and get a better performance than ELU in both training and testing set, but in Figure 7(b), for DeepConvNet, although ELU converge faster than the other two in the training set, it still get the lowest accuracy in testing set.

In addition, we can notice that although DeepConvNet has better accuracy (about 98%) than EEGNet (about 94%) for all activation functions in training set, in testing set it doesn't actually perform better. The reason may be that DeepConvNet uses too much parameters in the neural network, so it becomes easier to be overfitting.



(a) EEGNet



(b) DeepConvNet

Figure 7: Accuracy trend during training phase and testing phase

4 Discussion

4.1 Different Learning Rates

In this section, I use EEGNet with ReLU, which gets the highest accuracy among all models, to do the experiment with different learning rates while the other settings remain the same as Section 3. The results are shown in Figure 8.

As shown in the figure, with learning rate 0.001, the model gets the best performance in testing set. And we can notice that, in both training and testing, if the learning rate is too large (0.1), the training becomes unstable; if the learning rate is too small (0.0001), the curves converge slowly. In both cases, the final results are not good enough, thus it is important to choose an appropriate learning rate.

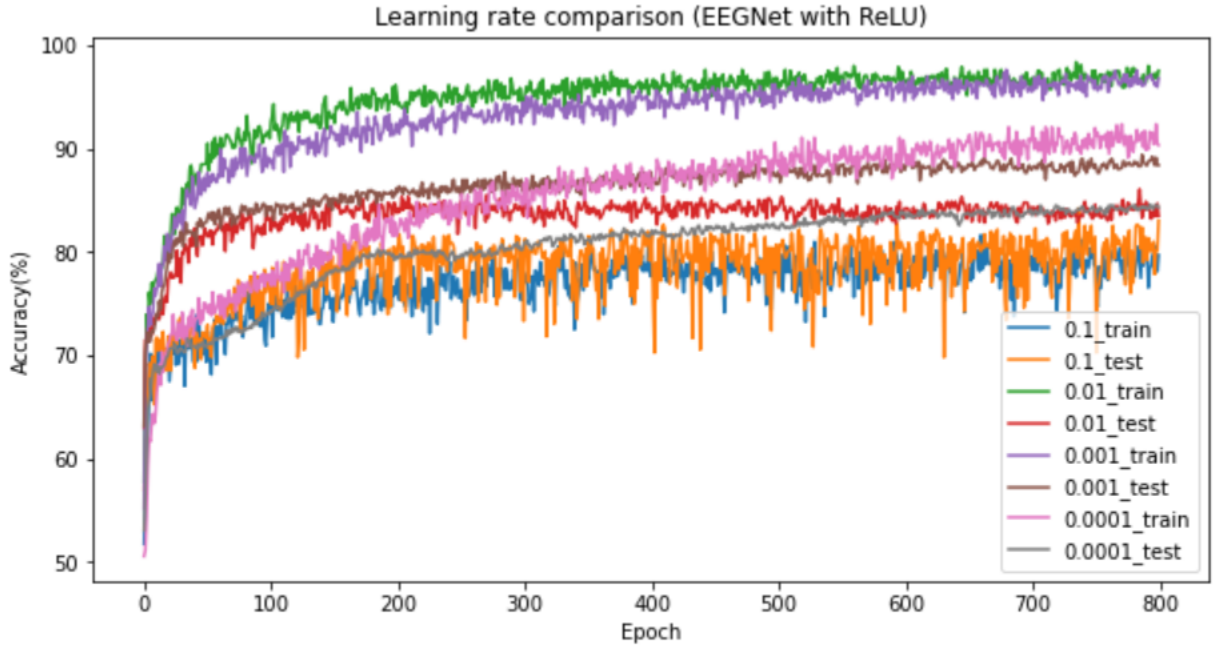
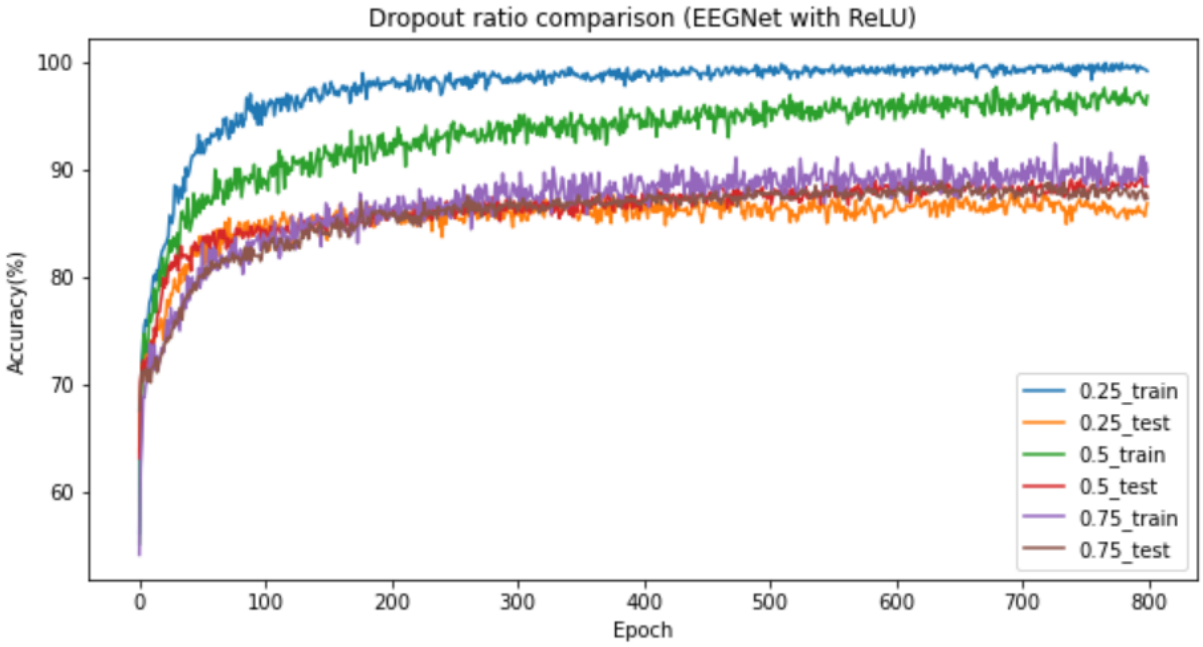


Figure 8: Learning rate comparison

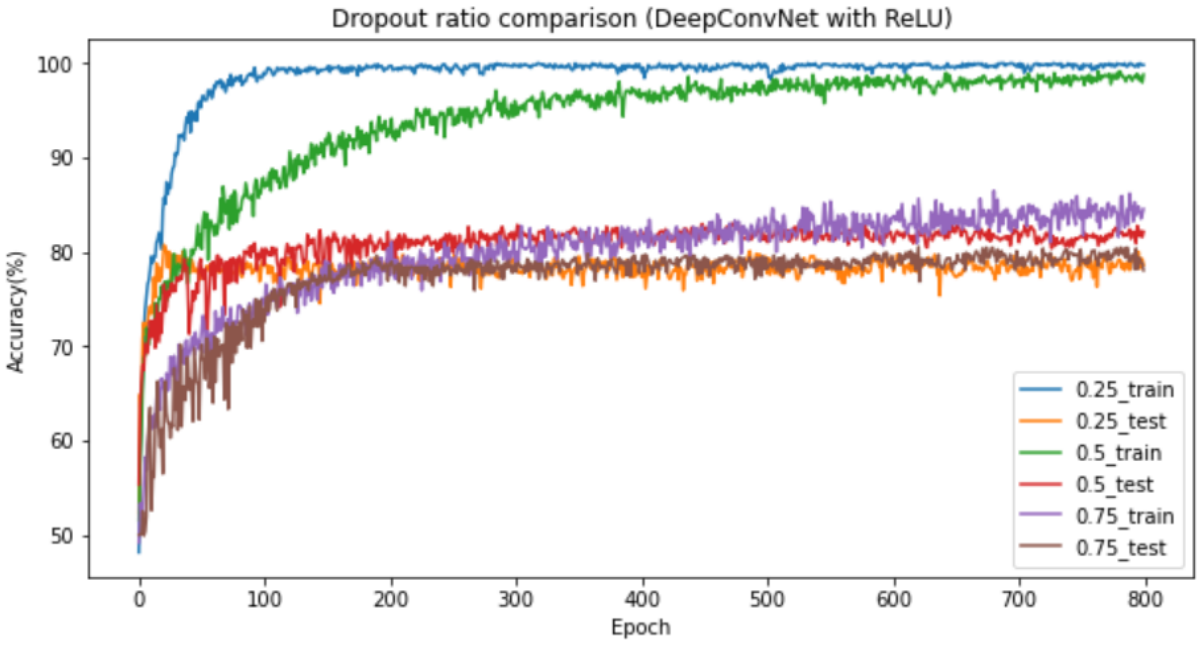
4.2 Different Dropout Ratios

In this section, I use both models with ReLU to do the experiment with different dropout ratios while the other settings remain the same as Section 3. The results are shown in Figure 9.

As shown in these figures, in both models, although the training converge faster with smaller dropout ratio in training set, I get a better testing result with dropout ratio = 0.5. I think the reason is that when dropout = 0.25, the model's capacity is too high, thus lead to the overfitting in the testing set. On the contrary, with dropout ratio = 0.75, the model's capacity is not large enough, so it leads to underfitting in the testing set.



(a) EEGNet



(b) DeepConvNet

Figure 9: Dropout ratio comparison