0860832 - Thong The Nguyen (阮世聰)

Lab 02 Report

1. Introduction

This report describes the implementation of of EEGNet and DeepConvNet for classification task of BCI competition dataset.

This report also describes the experiment results of those models with various settings, specifically three activation functions including ReLU, Leaky ReLU and ELU.

2. Experimental Setup

A. The detail of model

```
EEGNet(
  (first_conv): 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)
  )
  (depthwise_conv): 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): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.75, inplace=False)
  )
  (separable_conv): 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): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.75, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
  (criterion): CrossEntropyLoss()
)
```

Screenshot of EEGNet architecture

```
DeepConvNet(
  (first_layer): 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): LeakyReLU(negative_slope=0.01)
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.8, inplace=False)
  )
  (second_layer): 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): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (third_layer): 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): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (fourth_layer): 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): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (dense): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
  (criterion): CrossEntropyLoss()
)
```

DeepConvNet structure

```python
def train(self, train_gen, test_gen):

    train_log, test_log = [], []
    for epoch in range(self.epochs):
        for inputs, labels in train_gen:
            self.optimizer.zero_grad()
            outputs = self.forward(inputs)
            loss = self.criterion(outputs, labels.long())
            loss.backward()
            self.optimizer.step()

        train_acc = self.test(train_gen)
        test_acc = self.test(test_gen)
        train_log.append((epoch, train_acc))
        test_log.append((epoch, test_acc))
        print('Epoch [%d] training accuracy: %.2f%%' % (epoch, train_acc))

    print('\nFinished training!')
    print('Test accuracy: %.2f%%' % test_log[-1][1])

    return train_log, test_log
```

Training code

- Training code including calculating the gradient and update weights by supporting of Pytorch (for both EEGNet and DeepConvNet).
- The implementation also including random shuffling dataset (which is added in the last experiment).

```
""" Testing accuracy with specific data generator (train or test) """
def test(self, gen):
    with torch.no_grad():
        corrects = 0
        n_test = 0
        for inputs, labels in gen:
            outputs = self.forward(inputs)
            corrects += torch.sum(torch.max(outputs, 1)[1] == labels).item()
            n_test += labels.size()[0]

    return corrects / n_test * 100
```

Testing code

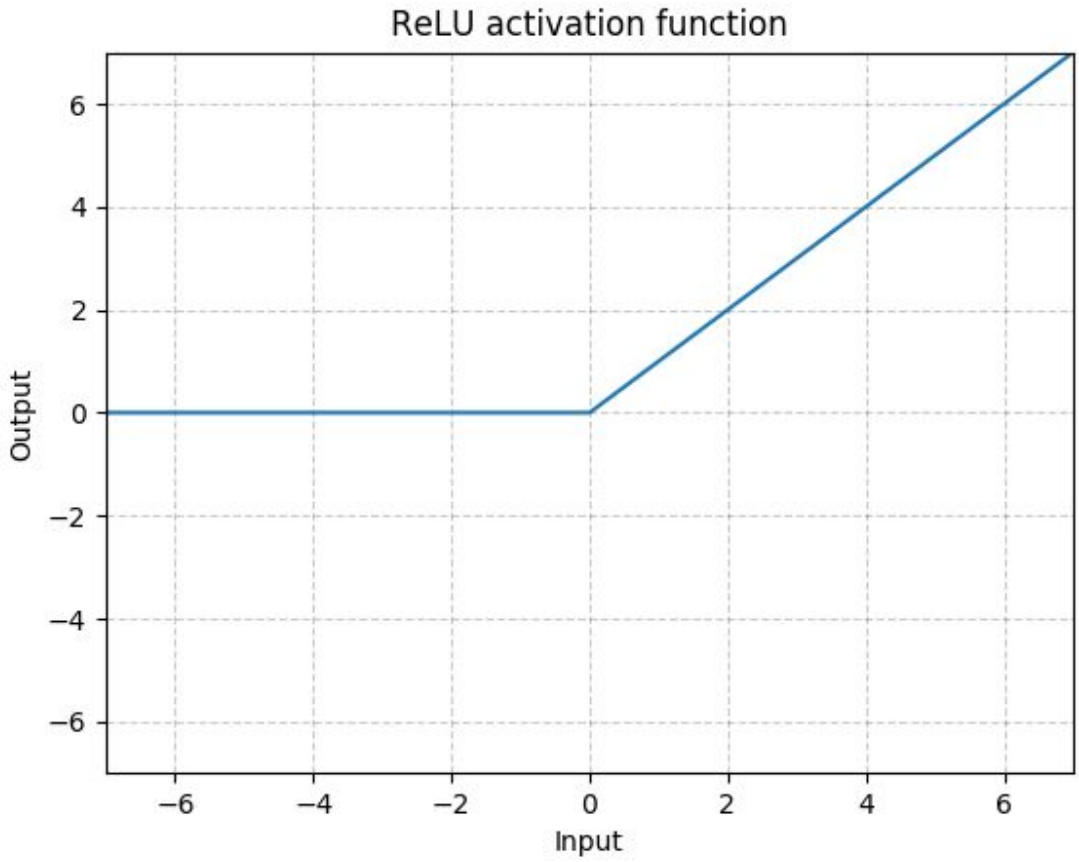
```
def forward(self, inputs):

    x = self.first_conv(inputs)
    x = self.depthwise_conv(x)
    x = self.separable_conv(x)
    x = self.classify(x.view(-1, 736))

    return x
```

```
def forward(self, inputs):

    x = inputs.view(-1, 1, 2, 750)
    x = self.first_layer(x)
    x = self.second_layer(x)
    x = self.third_layer(x)
    x = self.fourth_layer(x)
    x = self.dense(x.view(-1, self.flatten_size))

    return x
```

Forward implementation of EEGNet (left) and DeepConvNet (right)

B. Explain the activation function

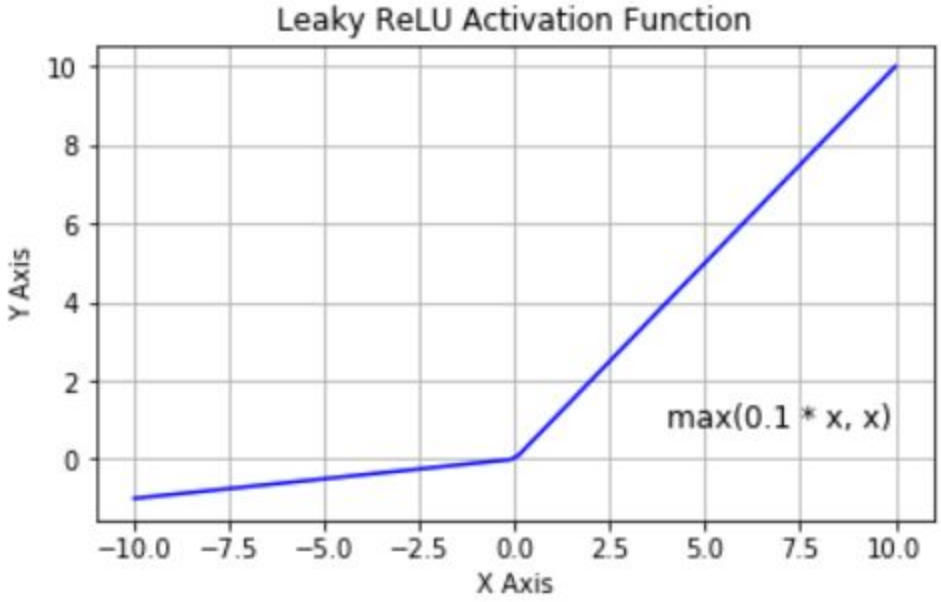
ReLU (Rectified Linear Unit)



Graph of ReLU function

- ReLU function: f(x) = max(0, x)
- ReLU is the most commonly used activation function in neural networks, especially in CNNs, because of its simple properties.
- The gradient of ReLU is simple and easy to compute, 0 when negatives and 1 when positive. Therefore, the model takes less time to run.
- ReLU is sparsely activated

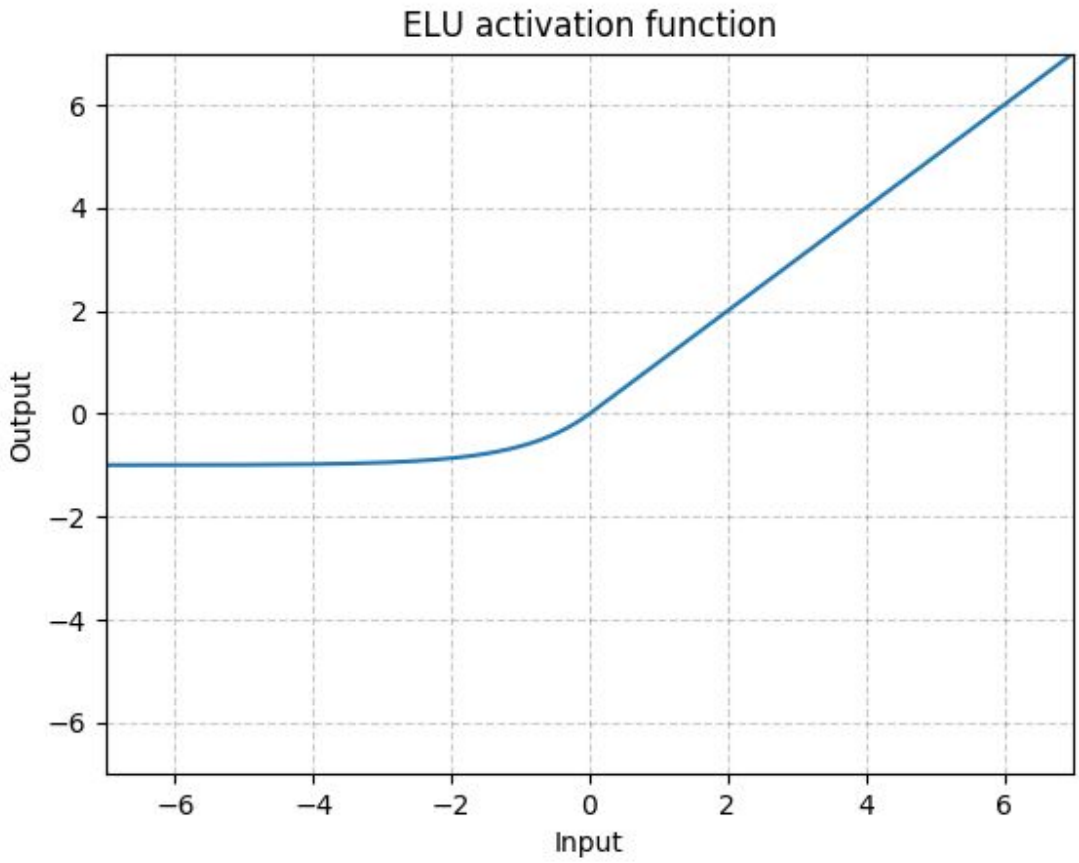
Leaky ReLU



Graph of Leaky ReLU function

- $LeakyReLU(x) = max(0, x) + negative_slope * min(0, x)$
- Leaky ReLU has a small slpe for vegative values, instead of zero like ReLU
- Leaky ReLU fix the problem of “dying ReLU” when ReLU gradient is always 0.

ELU (Exponential Linear Unit)



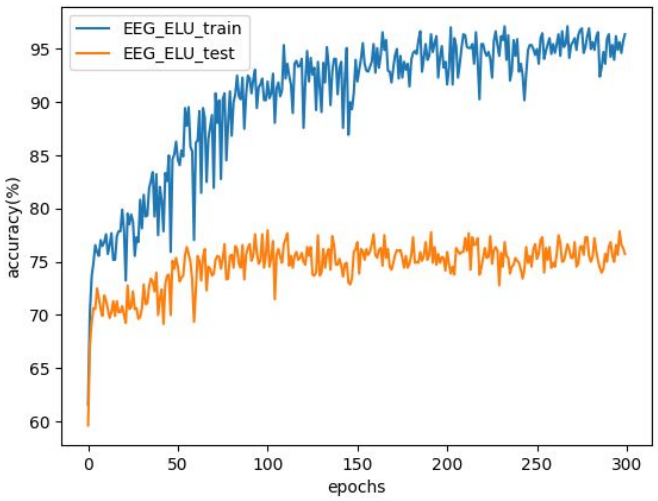
Graph of ELU function

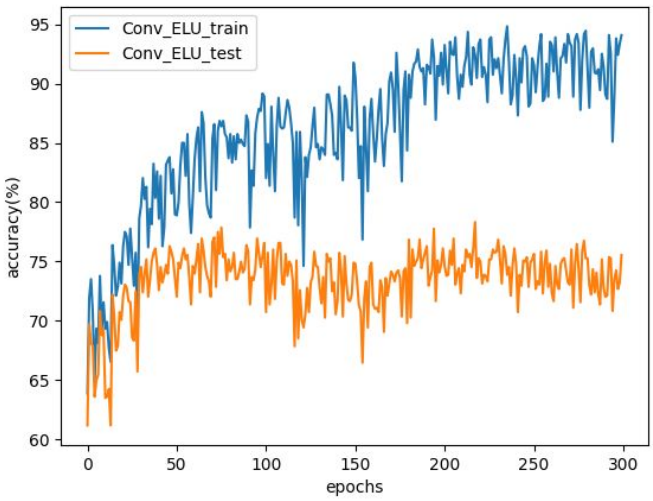
- $ELU(x) = max(0, x) + min(0, \alpha * (exp(x) - 1))$
- Instead of a straight line at negative value, the ELU use exponential function to make a curve.
- ELU combines the good part of ReLU and Leaky ReLU, while it avoids “dying ReLU”, it also saturates when the negative values are too large.

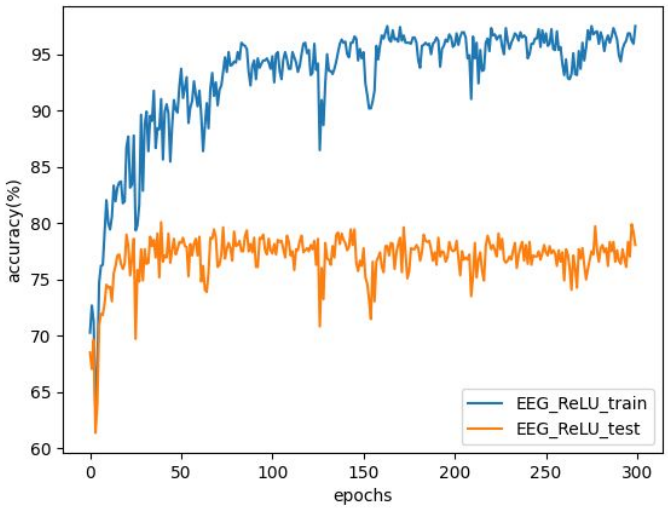
3. Experimental Result

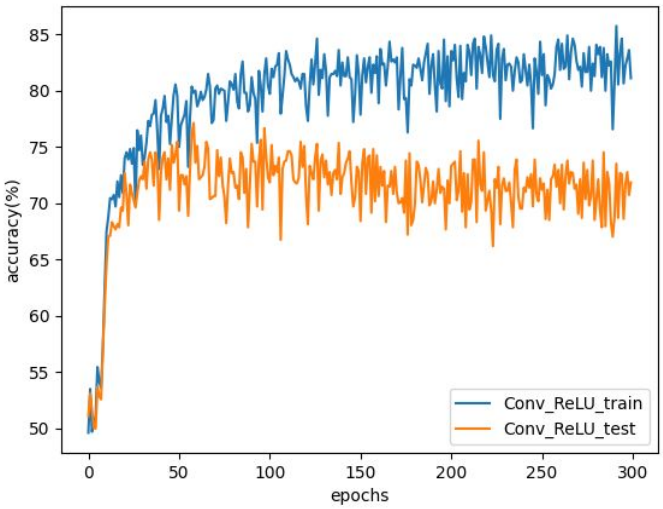
Experiments

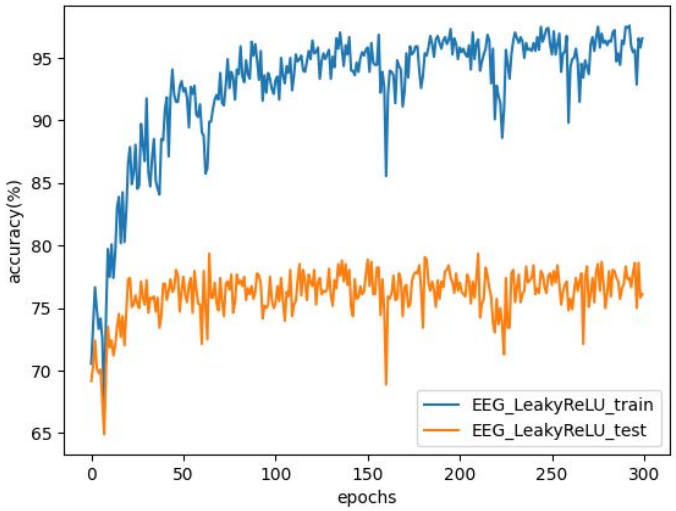
First configuration is the one recommended is the project description: *learning rate*: 0.01, *batch size*: 64, *epochs*: 150, *optimizer*: Adam, *loss*: CrossEntropy.

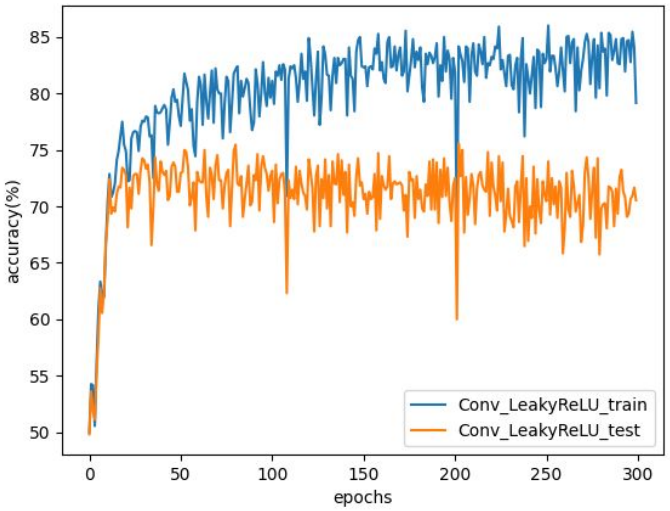








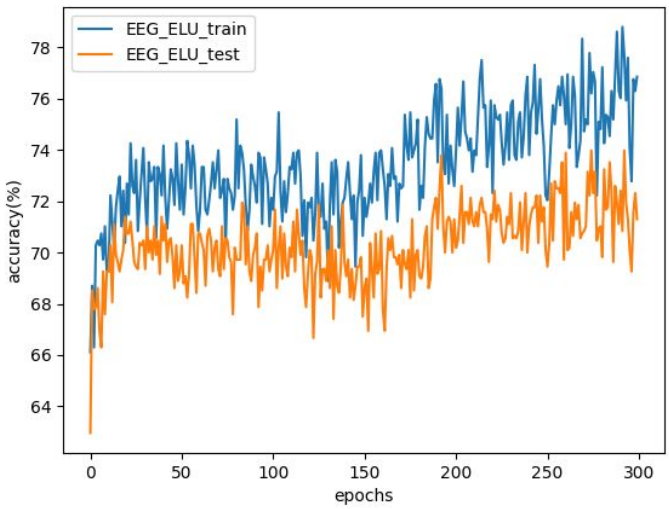


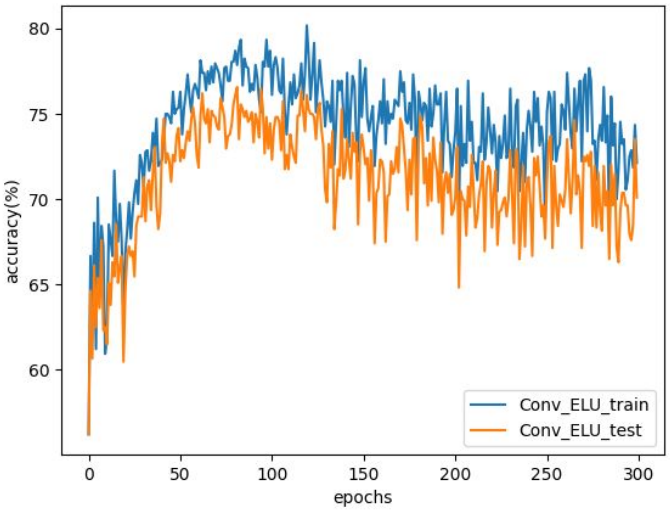


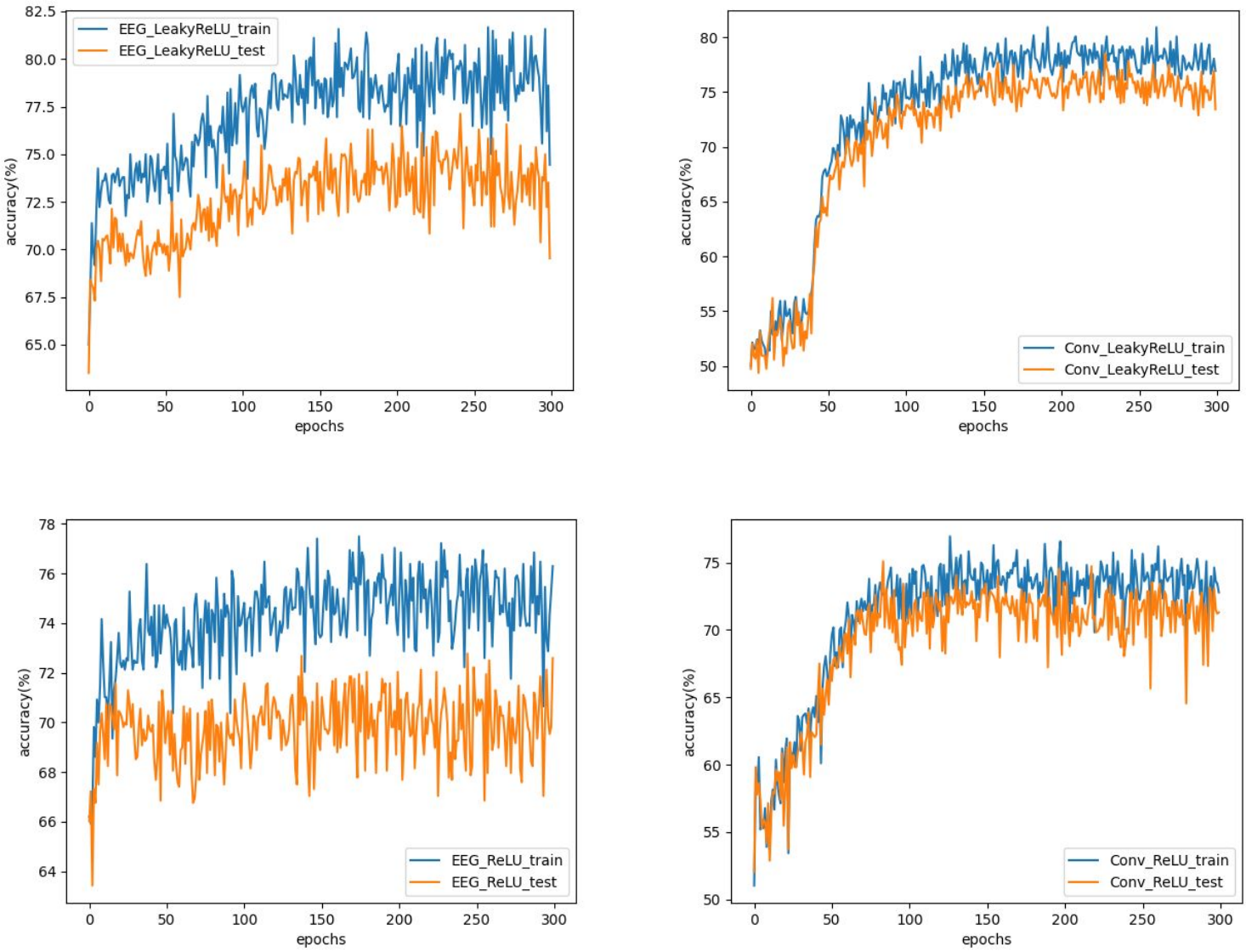
Training and testing result of EEGNet (left) and DeepConvNet (right)

We can see that both 2 models with 3 different actionvations can achieve around 95% accuracy on training set (although DeepConvNet is a little bit lower), but accuracy on testing set stopped around 75%. The plot shows that models may be overfitting and using 3 actionvation functions does not induce any different in the current setting. I will change some hyper-parameters to check whether result is improving or not.

To improve the (possibly) overfitting, we will try to increase the probability of dropout from 0.5 to 0.75 for hidden layers.

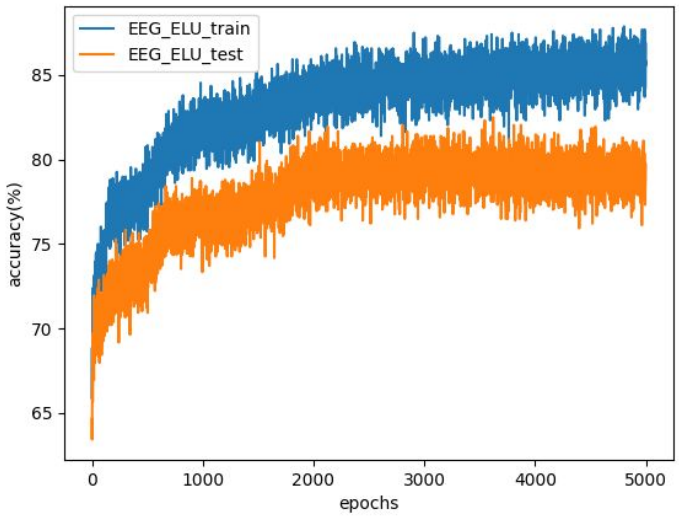


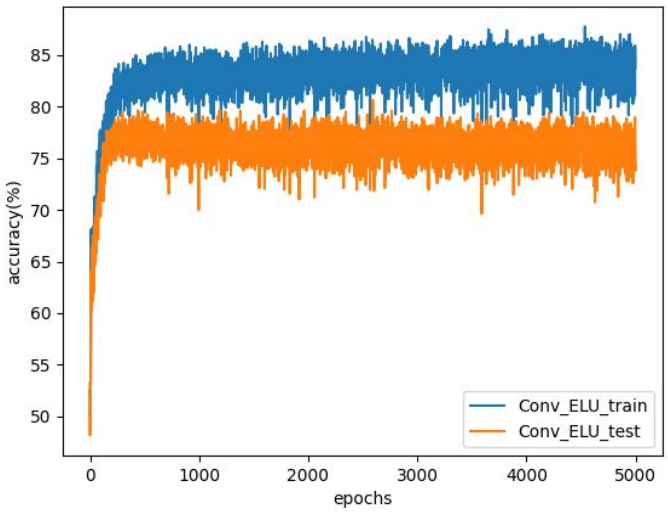


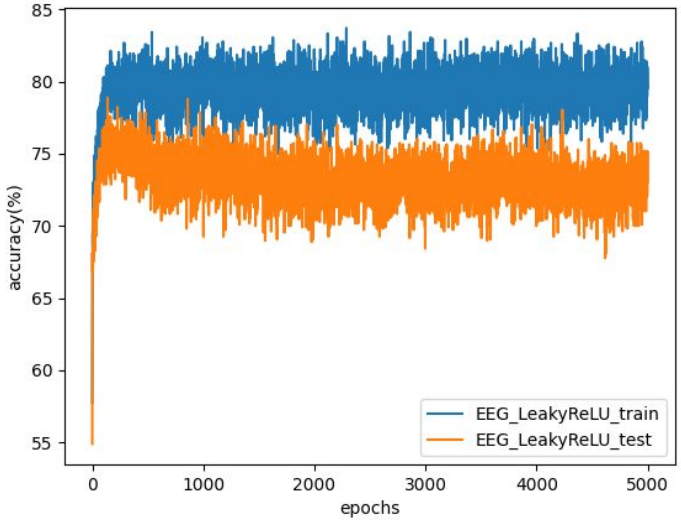


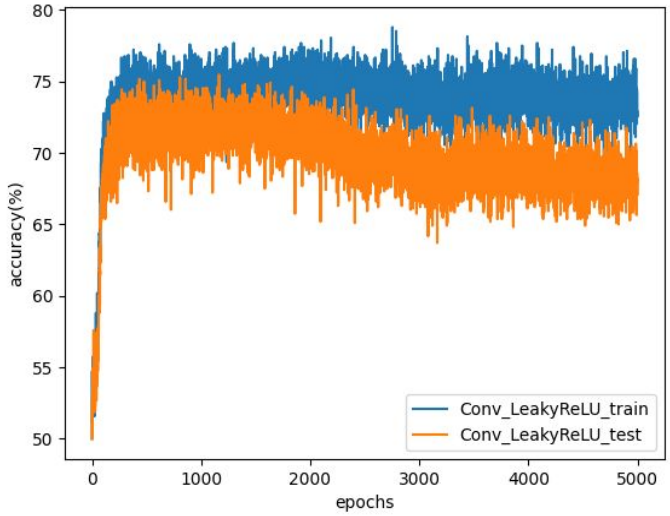
Training and testing result of EEGNet (left) and DeepConvNet (right)

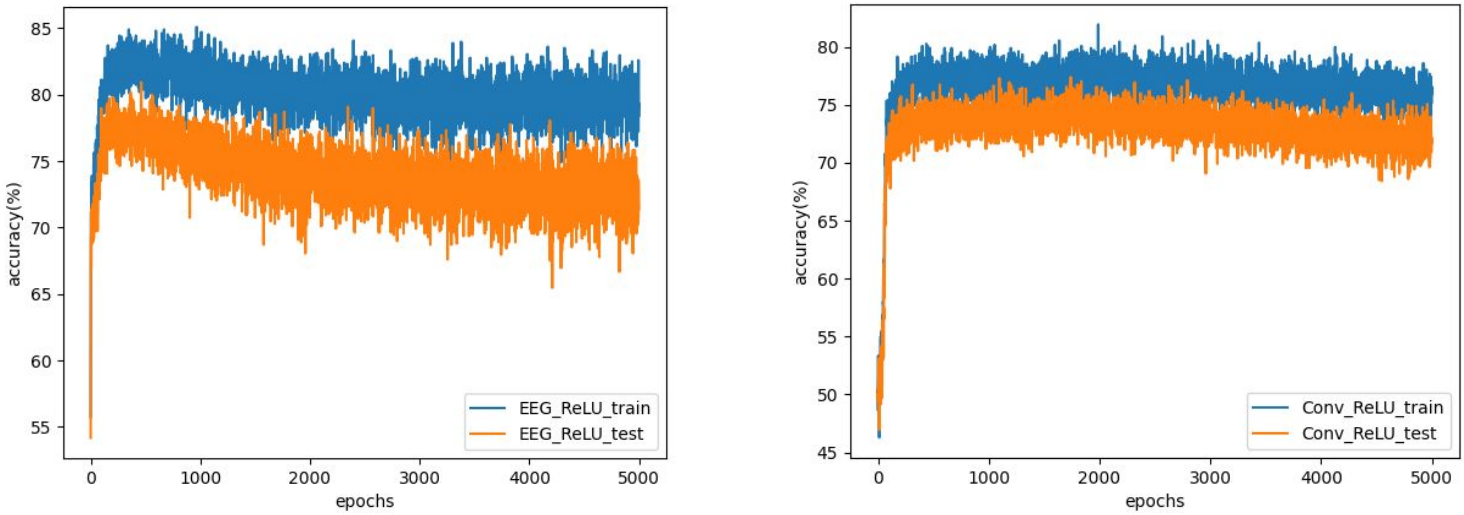
After changing the probability of dropout, we can see that both test accuracy of 2 models still follows the increasing trend of training accuracy, so we need to increase the number of epochs to see that we can get better accuracy.


EEG_ELU_train
EEG_ELU_test
accuracy(%)
epochs


Conv_ELU_train
Conv_ELU_test
accuracy(%)
epochs

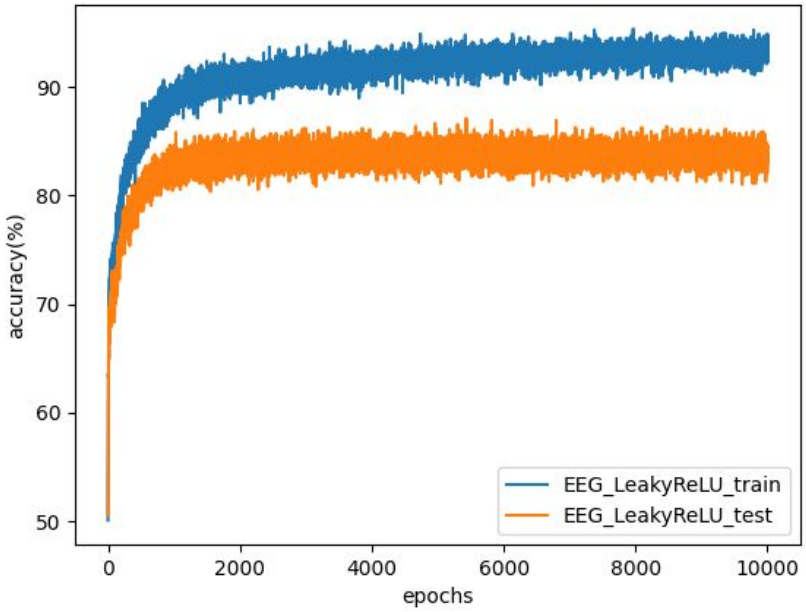

EEG_LeakyReLU_train
EEG_LeakyReLU_test
accuracy(%)
epochs


Conv_LeakyReLU_train
Conv_LeakyReLU_test
accuracy(%)
epochs

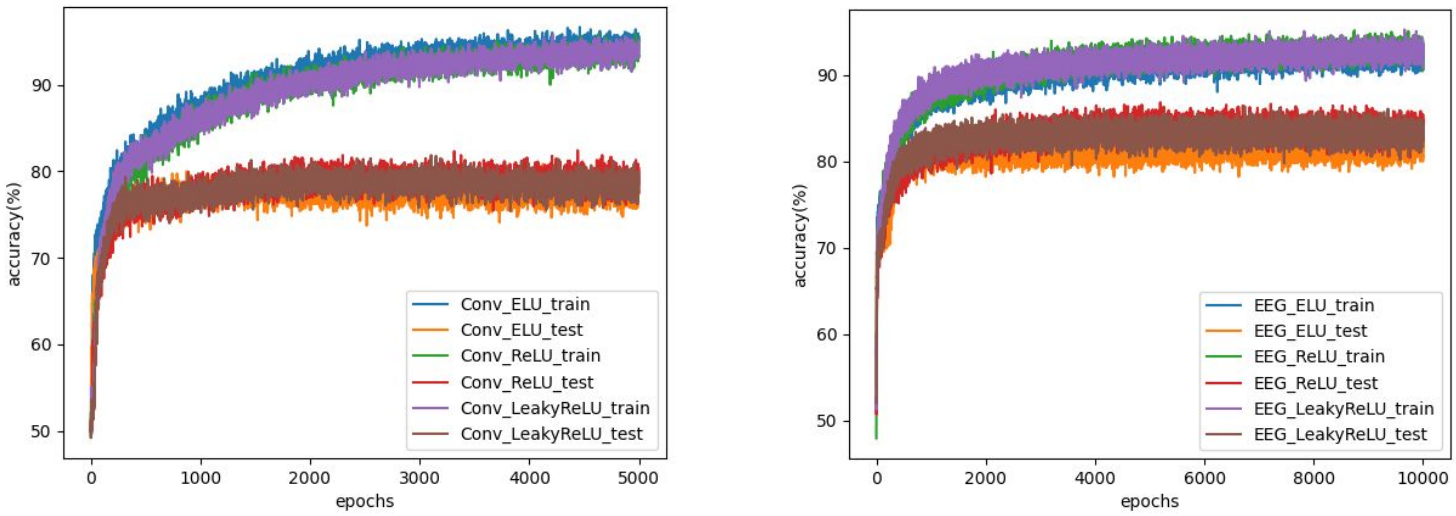


Training and testing result of EEGNet (left) and DeepConvNet (right)

The testing accuracy still not improve much, which is around 75%. We will try to improve the result by implement data shuffling after every epochs, reduce the learning rate to 0.001 to avoid overfitting.



After data shuffling, the model that has highest accuracy (85%) is EGGNet with LeakyReLU activation.



Comparision between training and testing for DeepConvNet & EEGNet

Comparision figure

The best result so far of 2 models after trying various configurations. EEGNet achieved the better result than DeepConvNet

	ELU	ReLU	Leaky ReLU

EEGNet;
batch_size=32;
learning_rate=1e-4;
dropout=0.5;
epochs=10000
	78.89%	84.54%	**85.65%**

DeepConvNet;
batch_size=1080,
learning_rate=1e-3;
dropout=0.75;
epochs=20000
	79.35%	74.81%	75.09%

4. Discussion

After some experiments, we can see that a small dataset (only 1080 data points), can induce overfitting for the model. Dropout is a regularization method that prevent overfitting by randomly set output of neuron to zero.

With small batch size, we should use learning rate small to reduce overfitting because of gradient is updated based on a small number of data points.

Randomly shuffle dataset will improve accuracy and reduce overfitting.