# DLP Lab2 Report

310551055 鄭伯俞

The code for Lab2 is hosted on https://github.com/Nemo1999/NYCU_DLP

## 0. Table of Contents

# Introduction

In this Lab, I implemented two different NN architectures to classify EEG signals. The signal represents two different control states of the Machine-Brain Interface. The input of our model is two different channels of a 750 sample record.

I learned to load data, build NN models, and train NN models using simple pytorch functions. I also learned to work on remote machine with a GPU. The result of DeepConvNet is quite good (about 86%~88% final accuracy), and the result of EEGNet is also good (about 84%~86% final accuracy). Although the acc of EEGNet is slightly lower, the number of parameters in EEGNet is much lower, hence the training and evaluation speed is faster than DeepConvNet.

# 1. Experiment Setups

## A. Detail of my Model

### EEGNet

The architecture of EEGNet is shown below, I use exactly the same parameters as given in the spec. The depthwise convolution is implemented by setting the 'groups' parameters of the 'Conv2d' layer. If we set groups = #channels, then each channel will be convoluted with a separated kernel.

```
Using cuda device
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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, 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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
    (5): Flatten(start_dim=1, end_dim=-1)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=False)
  )
)
```

### DeepConvNet

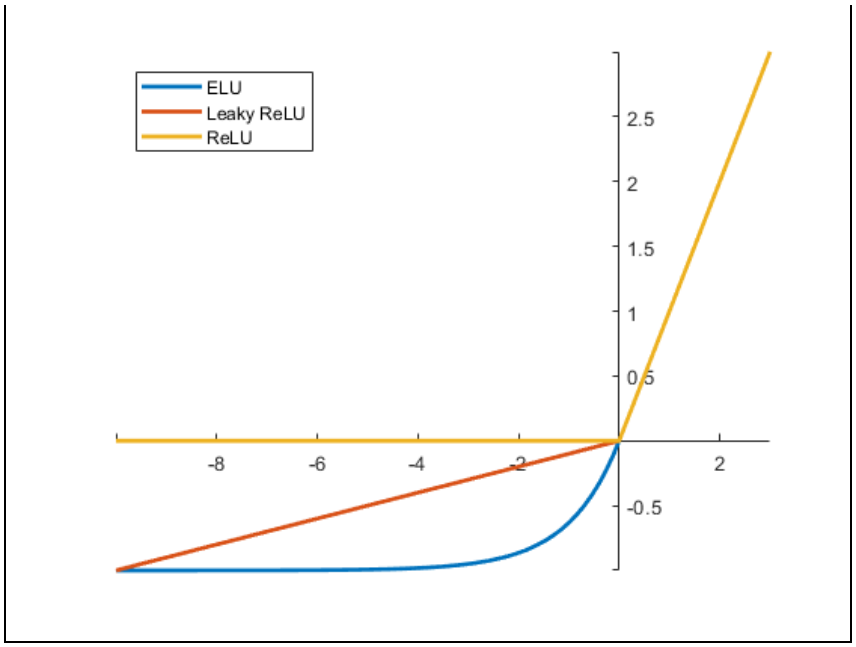
I slightly modify the model of DeepConvNet, first, I increased the size of Conv2d kernels from (1,5) to (1,8), second, I increased the dilation from (1,1) to (1,2). These two modifications gives as larger receptive field, hence enable the model to make prediction based on features involving larger time scales.

Also, I set the padding to ‘same’ for easier calculation of model dimensions.

```
DeepConvNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 8), stride=(1, 1), padding=same)
    (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=0.3, inplace=False)
  )
  (secondConv): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (thirdConv): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (fourthConv): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (classify): Linear(in_features=9200, out_features=2, bias=True)
)
```

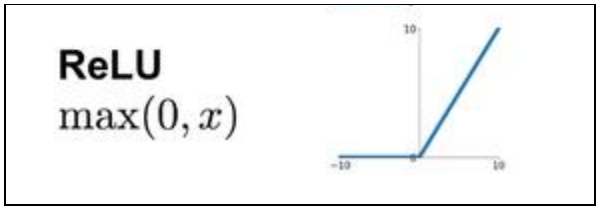
## B. Explain the activation function

We test three different types of parameters in this project. We compare their definition here. The plot of these three functions are shown here.



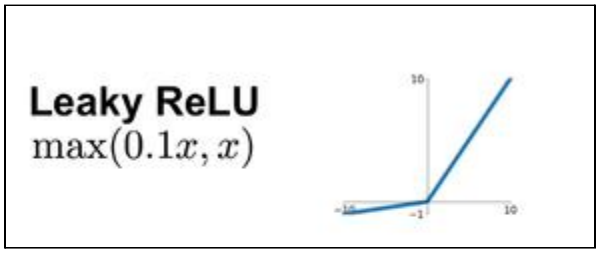

[Image source]

## ReLU



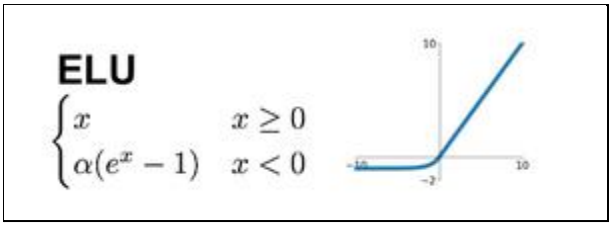

Pytorch implementation : ‘nn.ReLU()’
Relu is fast because of its simple definition

## Leaky ReLU




Pytorch implementation: ‘nn.LeakyReLU()’
Leaky ReLU tries to let us backprop the negative part of input, so we may trains faster than ReLU.

ELU




Torch implementation : ‘nn.ELU()’
ELU is a differentiable version of ReLU, also, output would have negative component, which makes the distribution more centered at zero.

# 2. Experimental results

## A. The highest testing accuracy

### EEGNet with ELU

```
Using cuda device
Model:EEGNet with ELU
  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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, 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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
    (5): Flatten(start_dim=1, end_dim=-1)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=False)
  )
)
Test ACC: 0.8546296296296296
Best ACC during training: 0.8555555555555555
```

### EEGNet with ReLU

```
Using cuda device
Model:EEGNet with ReLU
  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=0.25, 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=0.25, inplace=False)
    (5): Flatten(start_dim=1, end_dim=-1)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=False)
  )
)
Test ACC: 0.8712962962962963
Best ACC during training: 0.8796296296296297
```

## EEGNet with Leaky ReLU

```
Using cuda device
Model:EEGNet with LeakyReLU
  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): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, 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): LeakyReLU(negative_slope=0.01)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
    (5): Flatten(start_dim=1, end_dim=-1)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=False)
  )
)
Test ACC: 0.8666666666666667
Best ACC during training: 0.8824074074074074
```

## Deep ConvNet with ELU

```
Model:DeepConvNet with ELU
  DeepConvNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 8), stride=(1, 1), padding=same)
    (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): ELU(alpha=1.0)
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.3, inplace=False)
  )
  (secondConv): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.3, inplace=False)
  )
  (thirdConv): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.3, inplace=False)
  )
  (fourthConv): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.3, inplace=False)
  )
  (classify): Linear(in_features=9200, out_features=2, bias=True)
)
Test ACC: 0.8648148148148148
Best ACC during training: 0.8833333333333333
```

## Deep ConvNet with ReLU

```
Using cuda device
Model:DeepConvNet with ReLU
  DeepConvNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 8), stride=(1, 1), padding=same)
    (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=0.3, inplace=False)
  )
  (secondConv): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (thirdConv): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (fourthConv): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (classify): Linear(in_features=9200, out_features=2, bias=True)
)
Test ACC: 0.8583333333333333
Best ACC during training: 0.8907407407407407
```

## Deep ConvNet with LeakyReLU

```
Using cuda device
Model:DeepConvNet with LeakyReLU
  DeepConvNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 8), stride=(1, 1), padding=same)
    (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=0.3, inplace=False)
  )
  (secondConv): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (thirdConv): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (fourthConv): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 8), stride=(1, 1), padding=same, dilation=(1, 2))
    (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=0.3, inplace=False)
  )
  (classify): Linear(in_features=9200, out_features=2, bias=True)
)
Test ACC: 0.8851851851851852
Best ACC during training: 0.8888888888888888
```

## Best Test ACC During Training

Here we see that the best acc during training isn't very relevant to particular activation or model. Leaky ReLU seems to perform the best, while ReLU is also good.

| | ELU | ReLU | Leaky ReLU |
|---|---|---|---|
| EEGNet | 85.5% | 87.9% | **88.2%** |
| DeepConvNet | 88.3% | **89.0%** | 88.8% |

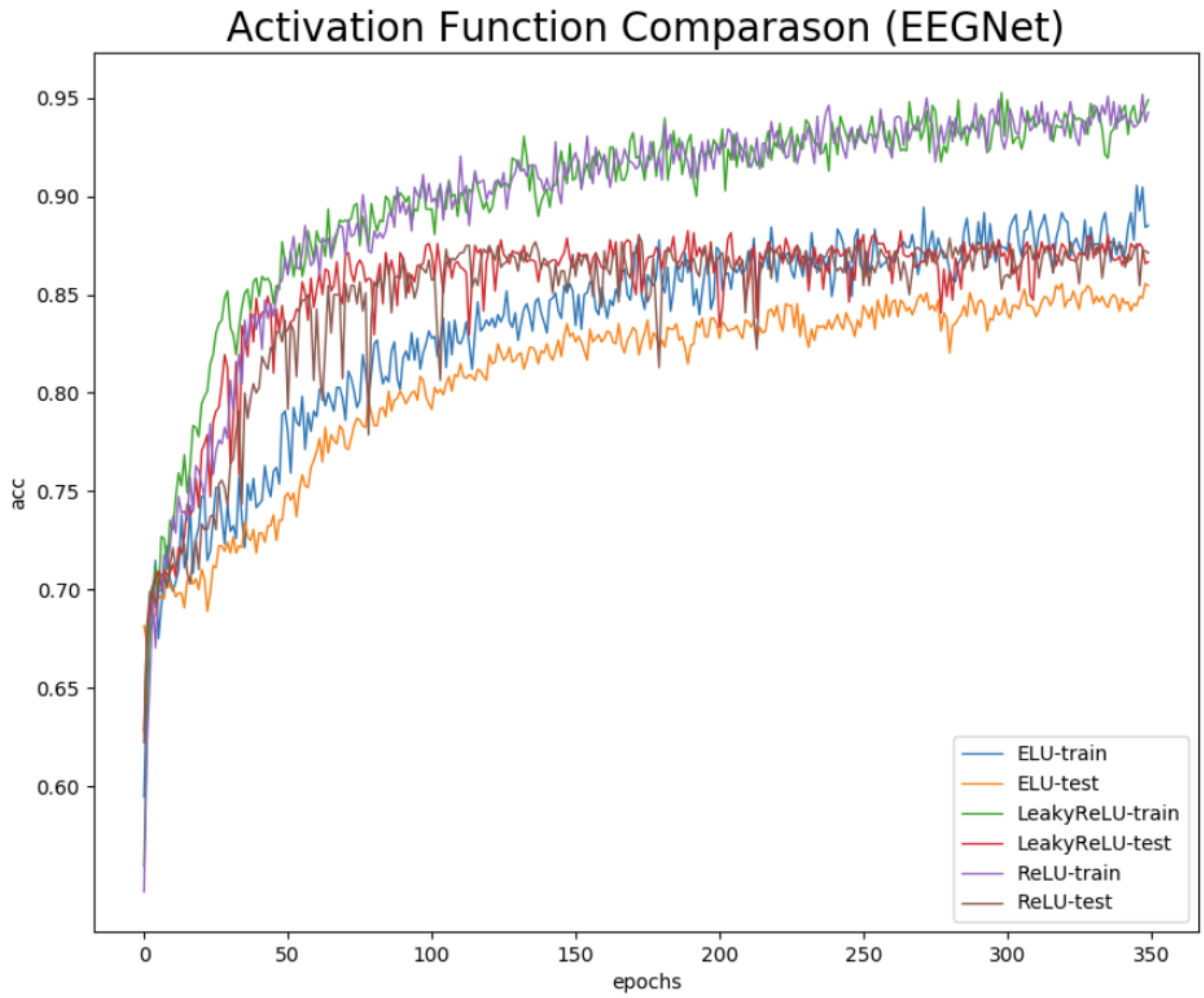
## Test Acc at epoch 300

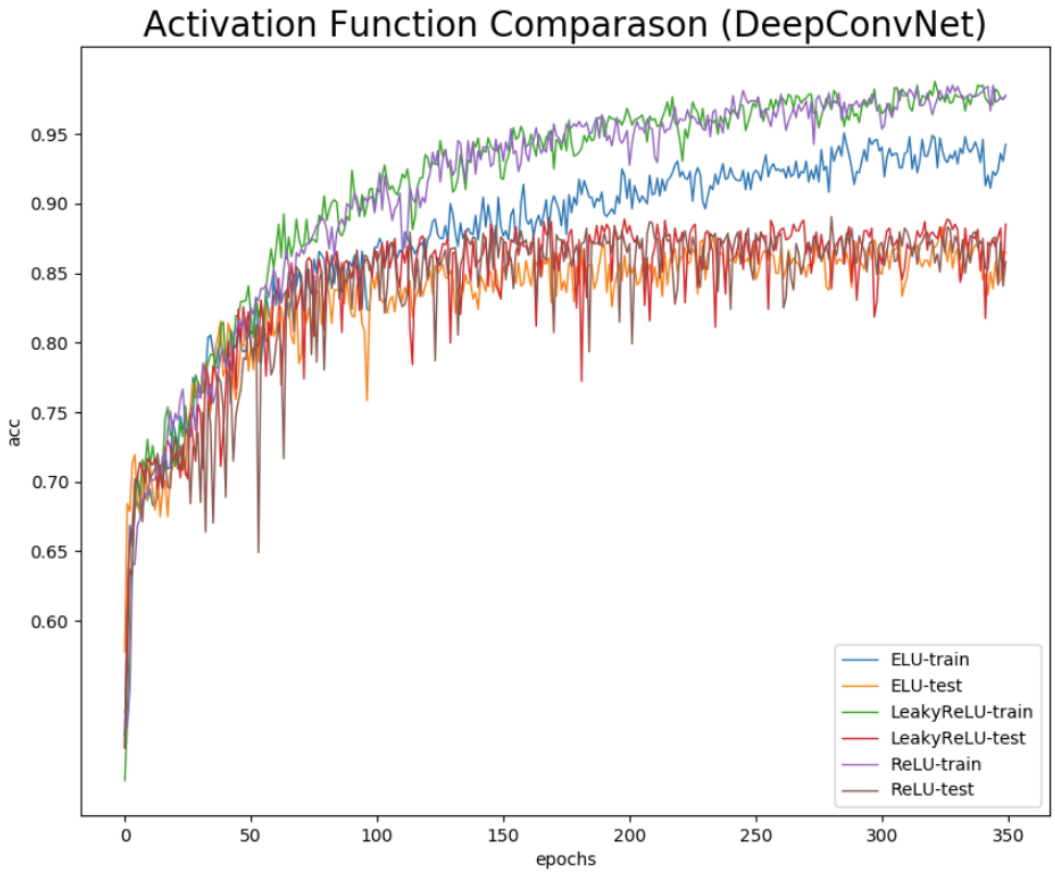
It is hard to decide when to stop training, for plotting purpose, we save the model parameter and accuracy at 300'th epoch. We record their testing accuracy as follow.

| | ELU | ReLU | Leaky ReLU |
|---|---|---|---|

| | | | |
|---|---|---|---|
| EEGNet | 85.4% | 87.1% | 86.6% |
| DeepConvNet | 86.4% | 85.8% | 88.5% |

## B. Comparison figures






# 3. Discussion

Here I want to discuss how I do data augmentation for EEG signals. First I notice that the signal is time invariant, which means that its shape looks similar at different starting time. We can shift the training data randomly before feeding them into our model, this gives us more different training examples, and let the phenomenon of overfitting less severe . Here I use ‘np.roll’ to shift the signal.

```python
class random_shift_dataset(Dataset):
    def __init__(self,x,y):
        assert x.shape[0] == y.shape[0], 'invalid dataset'
        self.x = x
        self.y = y
    def __len__(self):
        return self.x.shape[0]
    def __getitem__(self,idx):

        shift = np.random.randint(low = -self.x.shape[-1]/8, high = self.x.shape[-1]/8)
        x = np.roll(self.x[idx],shift,axis=-1)
        return tensor(x,dtype=torch.float), tensor(self.y[idx],dtype=torch.long)
```

After doing data augmentation, the ACC of my model boosts for about 3~5%.
In conclusion, I know that data augmentation is very important when the dataset is small.