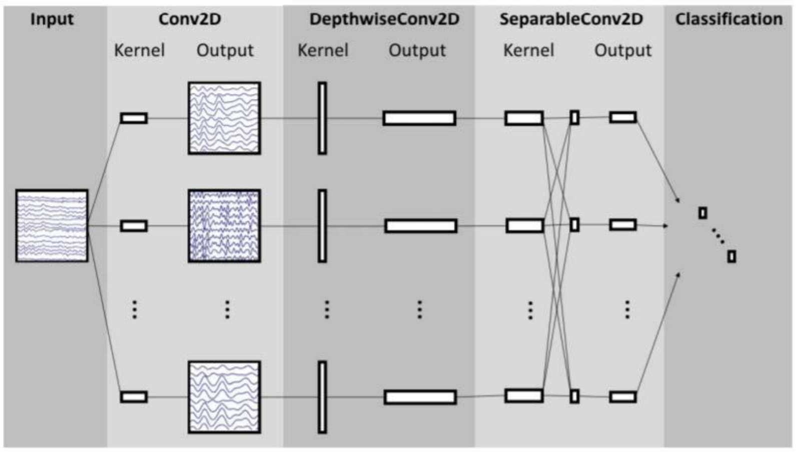
Deep Learning and Practice Lab 2

309554005 黃睿宇

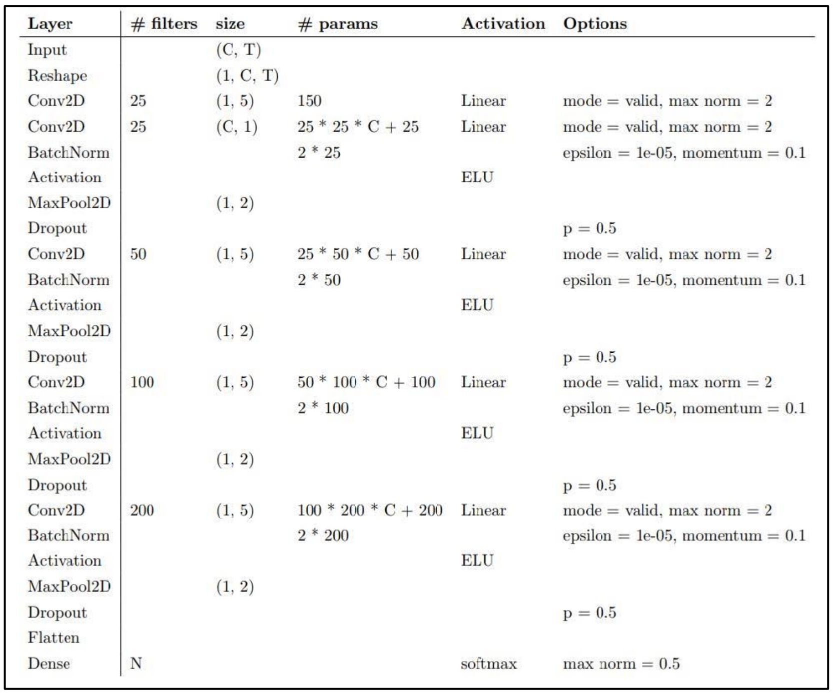
1 Introduction

In this lab, we implement simple EEG classification models which are EEGNet and DeepConvNet with three activation functions including ReLU, Leaky ReLU and ELU to classify signals of brain into two classes.

The overall visualization of the EEGNet architecture is shown below. The first Conv2D learns how to extract features from signals, then DepthwiseConv2D learns how to combine multiple channel signal into one for each input. The DepthwiseConv2D is different from normal convolution layers since it doesn't fully connect between input and output. Finally, SeparableConv2D learns how to extract features from the output from DepthwiseConv2D. The reference is from [EEGNet: A Compact Convolutional Neural Network for EEG-based Brain-Computer Interfaces]().

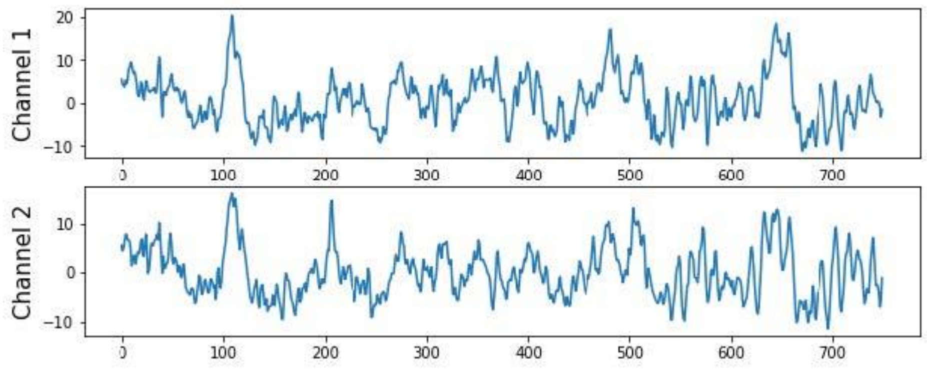


DeepConvNet architecture has multiple convolution layers, which is the same as normal CNN. The network is connected as C → CBAPD → CBAPD → CBAPD → CBAPD → fully connected, where C denotes convolution, B denotes batchnormalized, A denotes activation function, P denotes pooling, D denotes dropout. The parameters that we use are shown in the following table, where C = 2, T = 750, and N = 2.



Layer	# filters	size	# params	Activation	Options
Input		(C, T)			
Reshape		(1, C, T)			
Conv2D	25	(1, 5)	150	Linear	mode = valid, max norm = 2
Conv2D	25	(C, 1)	25 * 25 * C + 25	Linear	mode = valid, max norm = 2
BatchNorm			2 * 25		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	50	(1, 5)	25 * 50 * C + 50	Linear	mode = valid, max norm = 2
BatchNorm			2 * 50		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	100	(1, 5)	50 * 100 * C + 100	Linear	mode = valid, max norm = 2
BatchNorm			2 * 100		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	200	(1, 5)	100 * 200 * C + 200	Linear	mode = valid, max norm = 2
BatchNorm			2 * 200		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Flatten					
Dense	N			softmax	max norm = 0.5

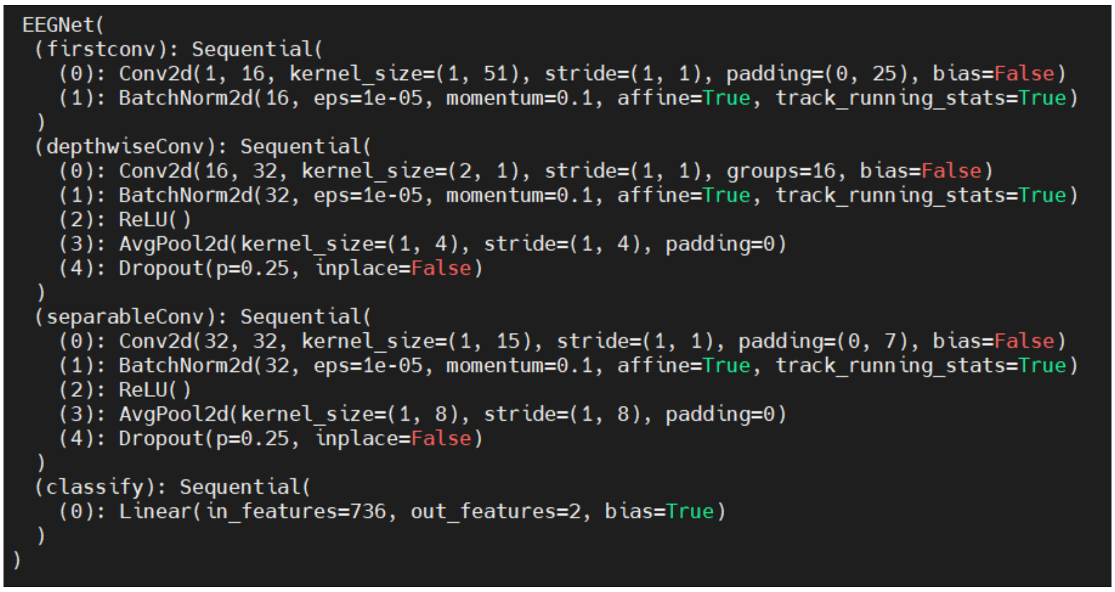
The dataset is from BCI competition dataset, which has two channels with two classes (left hand and right hand).



2 Experiment Setups

2.1 EEGNet

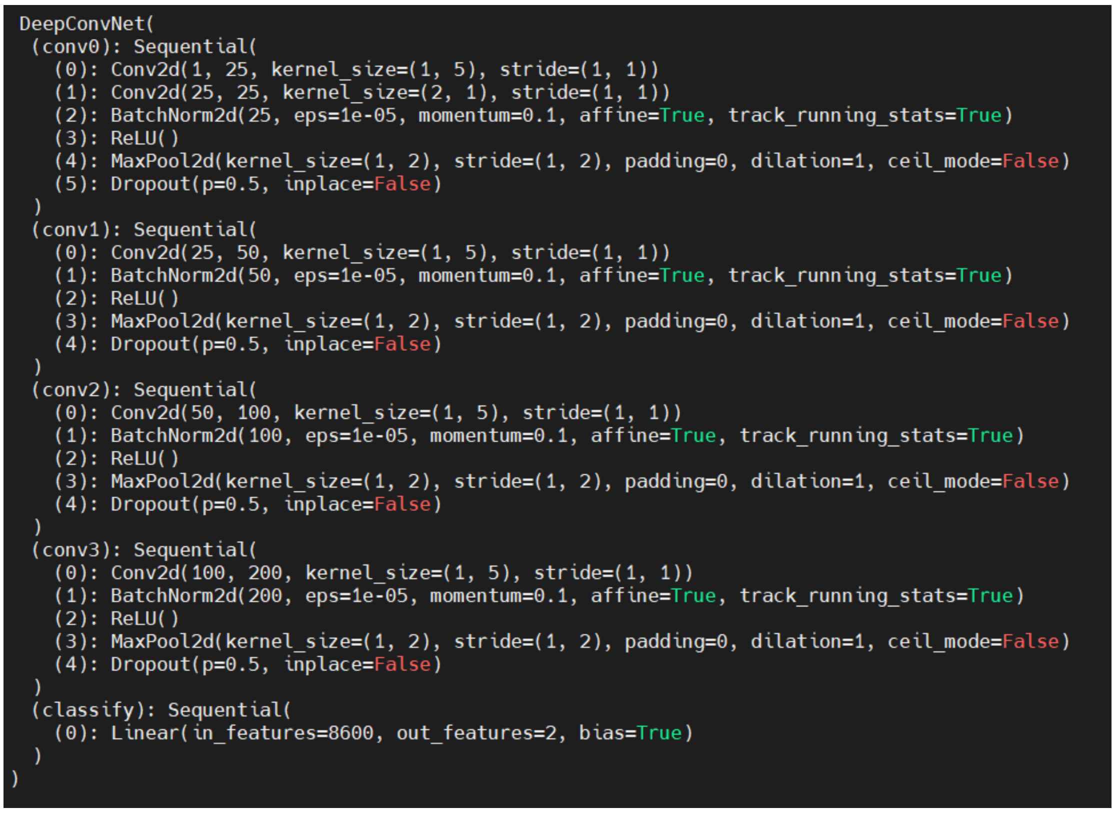
In our EEGNet model, there are five modules in both depthwise convolution and separable convolution, including 2D convolution, batch normalization, activation function, 2D average pooling, and dropout. While in first convolution, there are only two modules, including 2D convolution and batch normalization. The parameters of each module, including input size, output size, kernel size, stride, are shown below.



```
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)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

2.2 DeepConvNet

In our DeepConvNet model, there are five modules in convolution, including 2D convolution, batch normalization, activation function, 2D average pooling, and dropout. The parameters of each module, including input size, output size, kernel size, stride, are shown below.

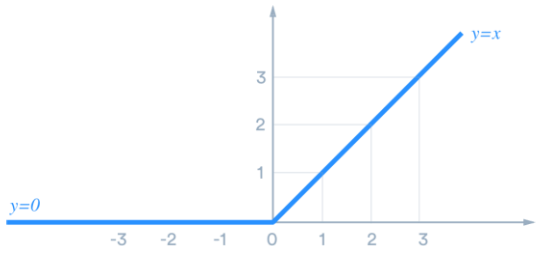


```
DeepConvNet(
  (conv0): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU()
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (conv1): 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)
  )
  (conv2): 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)
  )
  (conv3): 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)
  )
)
```

2.3 Activation functions

- ReLU

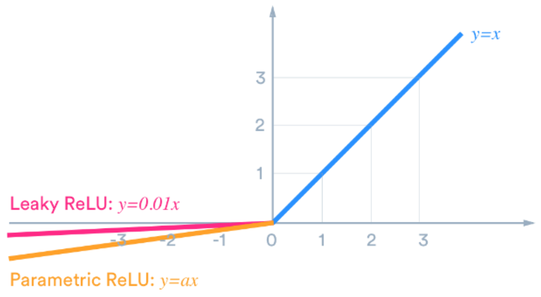
  ReLU stands for rectified linear unit. It is defined as $\text{ReLU}(x) = \max(0, x)$. It is linear for all positive values and zero for all negative values, which means model can therefore take less time to train or run since it's cheap to compute as there is no complicated math.



  However, the downside for being zero for all negative values is a problem called "dying ReLU". Since the slope of ReLU in the negative range is also 0, once a neuron gets negative, it's unlikely for it to recover.

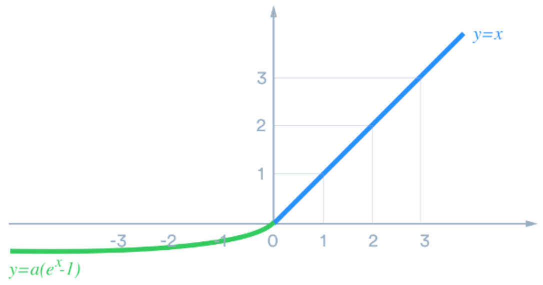
- Leaky ReLU

  Leaky ReLU has a small slope for negative values instead of altogether zero. It is defined as Leaky $\text{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax \text{ where } a < 0, & \text{otherwise} \end{cases}$. Leaky ReLU fixes the "dying ReLU" problem as it doesn't have zero-slope parts.



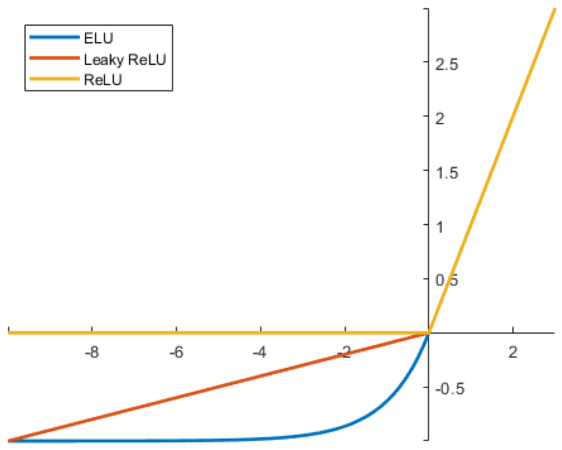
- ELU

  Similar to leaky ReLU, ELU has a small slope for negative values. Instead of a straight line, it uses a log curve like the following.



  ELU is defined as $\text{ELU}(x) = \max(0, x) + \min(0, a \times (\exp(x) - 1))$. It is designed to combine the good parts of ReLU and leaky ReLU.

The following is the illustration of output of ReLU, Leaky ReLU, and ELU function with varying input values.



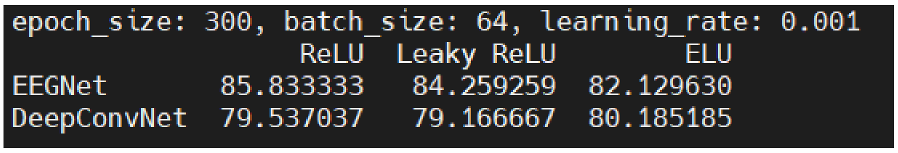
3 Experimental Results

3.1 The highest testing accuracy

The highest testing accuracy are trained on following parameters:

- optimizer: Adam
- loss function: cross entropy
- epochs = 300
- batch size = 64
- learning rate = 1e-3
- dropout for EEGNet = 0.25
- dropout for DeepConvNet = 0.5

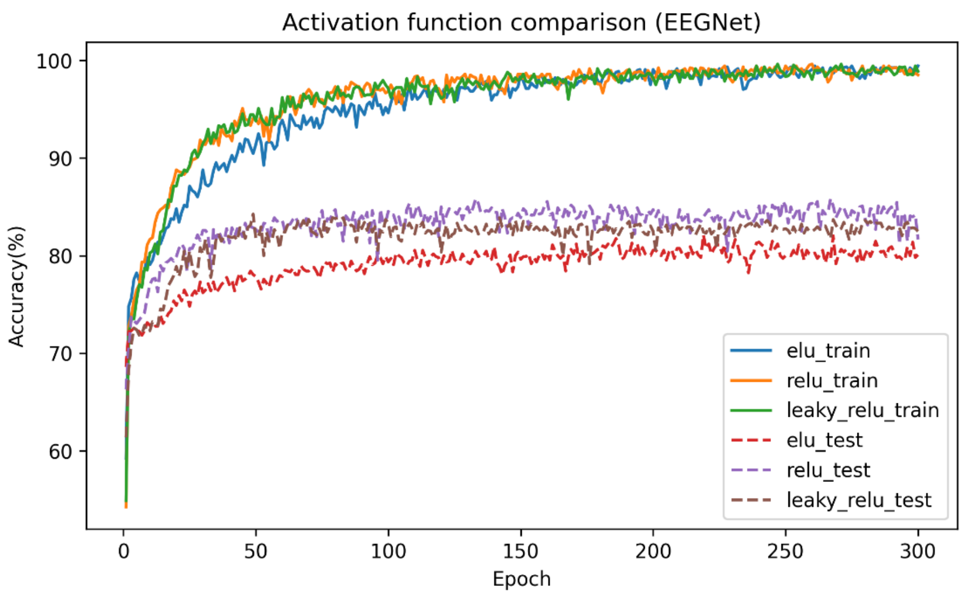
	ReLU	Leaky ReLU	ELU
EEGNet	85.83%	84.26%	82.13%
DeepConvNet	79.54%	79.17%	80.19%



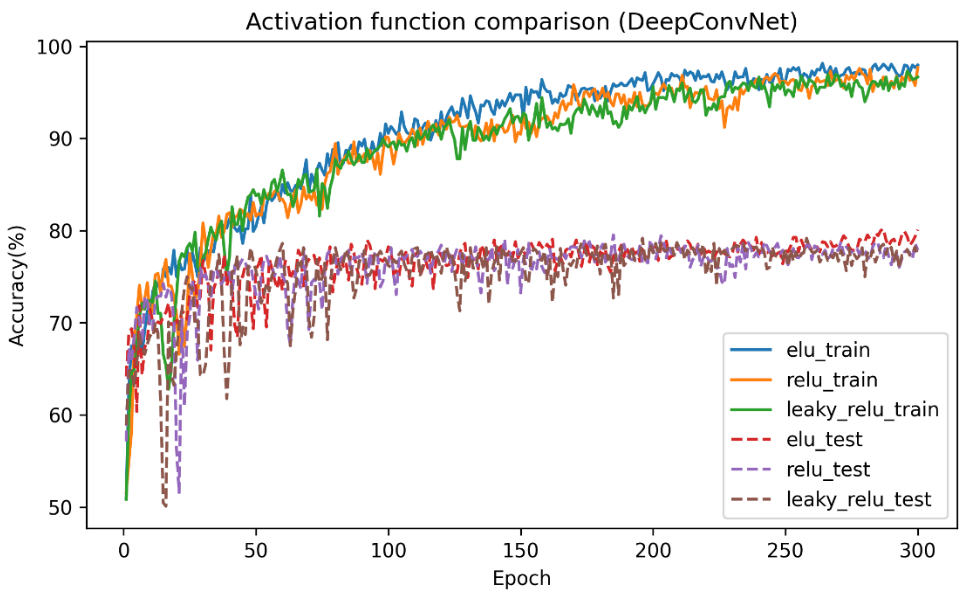
```
epoch_size: 300, batch_size: 64, learning_rate: 0.001
                  ReLU  Leaky ReLU        ELU
EEGNet       85.833333   84.259259  82.129630
DeepConvNet  79.537037   79.166667  80.185185
```

3.2 Comparison figures

- EEGNet



- DeepConvNet

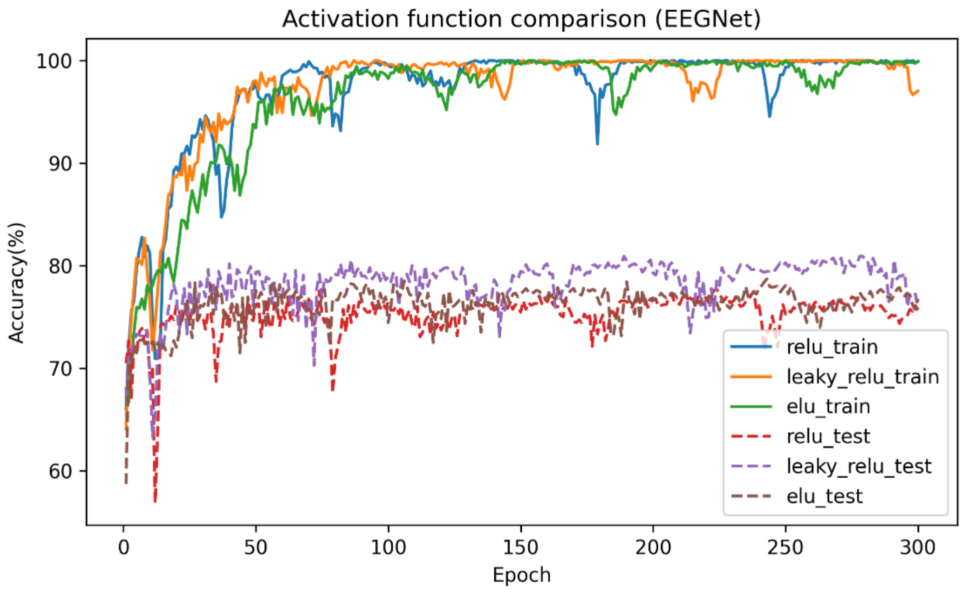


4 Discussion

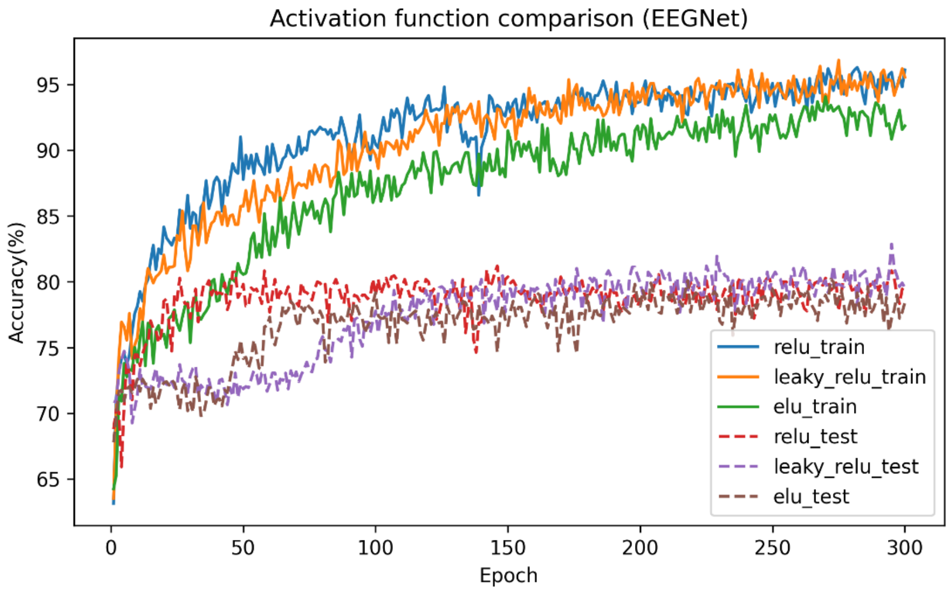
The dropout layer is used in both models. Hyper parameter of dropout layer is dropout rate that decide how many chances that input value becomes zero. If dropout rate is bigger, more input values will become zero, and vice versa. The dropout layer causes unstable in output accuracy of model but it can solve overfitting problem since model can't learn all features from training data when training (some input become zero).

To observe the influence of dropout rate, we tried to modify dropout rate in EEGNet model. The results are shown below.

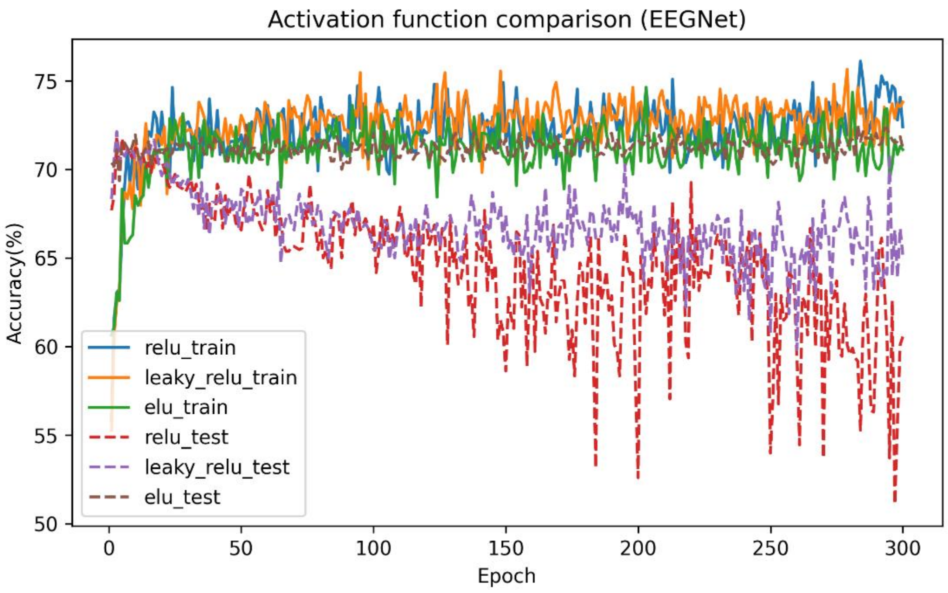
- dropout rate = 0.1



- dropout rate = 0.5



- dropout rate = 0.9



We could find that the accuracy in high dropout rate is worse than that in low dropout rate. Thus, choosing a proper dropout rate is important as well.