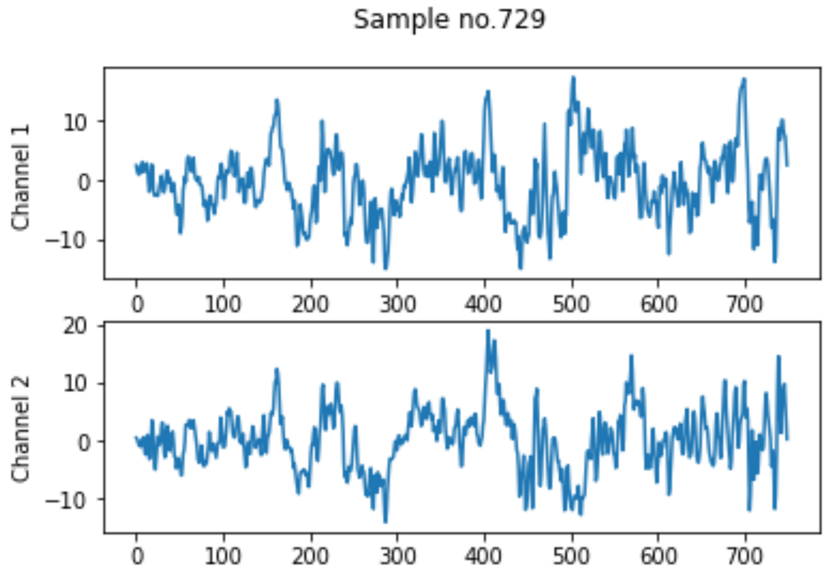
- Introduction (20%)

  EEG data 是大腦產生的電訊號, 會藉著在腦殼外電擊來進行測量, BCI dataset 利用 2 channels 收集了許多不同實驗的 EEG data, 共有為動左手、動右手兩種類別, 本實驗將會使用 BCI dataset 裡面的部分資料來實作簡單的 EEG 動作想像的分類，圖為其中一筆資料收集到的 EEG data, 並透過 EEGNet 與 DeepConvNet 兩種 model, 還有 ReLU, Leaky ReLU, ELU 三種 activation function 來實作分類算法。



- Experiment set up (30%)

  A. The detail of your model

  - EEGNet

    跟投影片不同的地方在於所有 Dropout layer 的部分, 都從 p=0.25 改成 p=0.5。

```
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)
  )
)
```

```python
class EEGNet(nn.Module):
    def __init__(self,activation):
        super(EEGNet, self).__init__()
        self.firstconv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(1,51), stride=(1, 1),padding=(0,25),bias=False),
            nn.BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16,32,kernel_size=(2,1), stride=(1,1),groups=16,bias=False),
            nn.BatchNorm2d(32,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            activation(),
            nn.AvgPool2d(kernel_size=(1,4), stride=(1,4), padding=0),
            nn.Dropout(p=0.5) #p=0.25
        )
        self.separableConv = nn.Sequential(
            nn.Conv2d(32,32,kernel_size=(1,15),stride=(1,1),padding=(0,7),bias=False),
            nn.BatchNorm2d(32,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True),
            activation(),
            nn.AvgPool2d(kernel_size=(1,8),stride=(1,8),padding=0),
            nn.Dropout(p=0.5) #p=0.25
        )
        self.classify = nn.Sequential(
            nn.Linear(in_features=736,out_features=2,bias=True)
        )

    def forward(self,x):
        h1 = self.firstconv(x)
        h2 = self.depthwiseConv(h1)
        h3 = self.separableConv(h2) #h3: (64, 32, 1, 23)
        h3 = h3.view(h3.shape[0],-1) #h3: (64, 736), flatten
        y = self.classify(h3)
        return y
```

- DeepConvNet

  表格中的 # filters 代表 nn.Conv2d 的第二個參數, 亦即 number of channels, 而 size 則代表 nn.Conv2d 的第三個參數, 亦即 kernel size。

  DeepConvNet 共有 1 個 input layer, 4 個 hidden layer, 1 個 output layer, 其中每層 hidden layer 都依序包含 Conv2D、BatchNorm、Activation、MaxPool2D、Dropout。

Layer	# filters	kernel 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~~ 5 + 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~~ 5 + 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~~ 5 + 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 (include in nn.CrossEntropy)	max norm = 0.5

```
class DeepConvNet(nn.Module):
    def __init__(self,activation):
        super(DeepConvNet, self).__init__()
        channels = (25,25,50,100,200)
        kernel_sizes = ((1,5),(2,1),(1,5),(1,5),(1,5))
        self.conv0 = nn.Conv2d(1,channels[0],kernel_size=kernel_sizes[0]) #batchsize=1, number of
            channels=25, kernel size = (1,5)
        for i in range(1,len(channels)):
            #seems that locals()[f'self.conv{i}'] can't run
            setattr(self,f'conv{i}',nn.Sequential(
                nn.Conv2d(channels[i-1],channels[i],kernel_size=kernel_sizes[i]),
                nn.BatchNorm2d(channels[i],eps=1e-5,momentum=0.1),
                activation(),
                nn.MaxPool2d(kernel_size=(1,2)),
                nn.Dropout(p=0.5)
            ))
        self.classify = nn.Linear(in_features=8600,out_features=2) #8600 is because of the comment in
            DeepConvNet.forward.h5
    def forward(self,x):
        h1=self.conv0(x)
        h2=self.conv1(h1)
        h3=self.conv2(h2)
        h4=self.conv3(h3)
        h5=self.conv4(h4) #h5: (64, 200, 1, 43)
        h5 = h5.view(h5.shape[0],-1) #h5: (64, 8600), flatten
        y = self.classify(h5)
        return y
```

B. Explain the activation function (ReLU, Leaky ReLU, ELU)

- ReLU

$$\mathrm{ReLU}(x) = \max(0, x)$$
$$\frac{d\mathrm{ReLU}(x)}{dx} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$$

```
x = torch.arange(-10,10,dtype=torch.float,requires_grad=True)
y = nn.ReLU()(x)
y.backward(torch.ones(y.shape))
print(x.grad)
```

```
In [13]: x
Out[13]:
tensor([-10.,  -9.,  -8.,  -7.,  -6.,  -5.,  -4.,  -3.,  -2.,  -1.,   0.,   1.,
          2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.], requires_grad=True)

In [14]: y
Out[14]:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 2., 3., 4., 5., 6., 7.,
        8., 9.], grad_fn=<ReluBackward0>)

In [15]: x.grad
Out[15]:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1.,
        1., 1.])
```

- Leaky ReLU

$$\text{default negative\_slope } = 0.01$$
$$\mathrm{LeakyReLU}(x) = \begin{cases} \text{negative\_slope } * x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$
$$\frac{d\mathrm{LeakyReLU}(x)}{dx} = \begin{cases} \text{negative\_slope} & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

```
y = nn.LeakyReLU()(x)
x.grad.data.zero_()
y.backward(torch.ones(y.shape))
print(x.grad)
```

```
In [17]: x
Out[17]:
tensor([-10.,  -9.,  -8.,  -7.,  -6.,  -5.,  -4.,  -3.,  -2.,  -1.,   0.,   1.,
          2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.], requires_grad=True)

In [18]: y
Out[18]:
tensor([-0.1000, -0.0900, -0.0800, -0.0700, -0.0600, -0.0500, -0.0400, -0.0300,
        -0.0200, -0.0100,  0.0000,  1.0000,  2.0000,  3.0000,  4.0000,  5.0000,
         6.0000,  7.0000,  8.0000,  9.0000], grad_fn=<LeakyReluBackward0>)

In [19]: x.grad
Out[19]:
tensor([0.0100, 0.0100, 0.0100, 0.0100, 0.0100, 0.0100, 0.0100, 0.0100, 0.0100,
        0.0100, 0.0100, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000,
        1.0000, 1.0000])
```

- ELU

$$
\begin{aligned}
\text{default } \alpha &= 1.0 \\
\mathrm{ELU}(x) &= \max(0, x) + \min(0, \alpha * (\exp(x) - 1)) \\
&= \begin{cases} \alpha(e^x - 1) & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases} \\
\frac{\mathrm{dELU}(x)}{\mathrm{d}x} &= \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}
\end{aligned}
$$

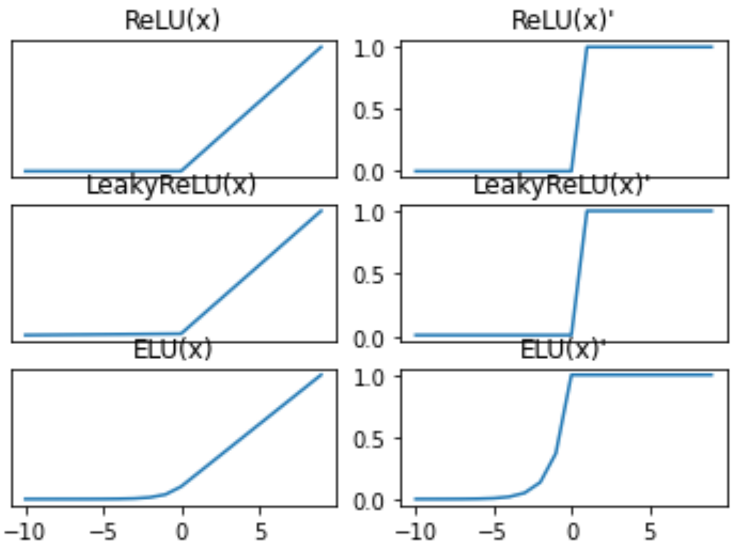
```python
y = nn.ELU()(x)
x.grad.data.zero_()
y.backward(torch.ones(y.shape))
print(x.grad)
```

```
In [21]: x
Out[21]:
tensor([-10.,  -9.,  -8.,  -7.,  -6.,  -5.,  -4.,  -3.,  -2.,  -1.,   0.,   1.,
          2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.], requires_grad=True)

In [22]: y
Out[22]:
tensor([-1.0000, -0.9999, -0.9997, -0.9991, -0.9975, -0.9933, -0.9817, -0.9502,
        -0.8647, -0.6321,  0.0000,  1.0000,  2.0000,  3.0000,  4.0000,  5.0000,
         6.0000,  7.0000,  8.0000,  9.0000], grad_fn=<EluBackward>)

In [23]: x.grad
Out[23]:
tensor([4.5419e-05, 1.2338e-04, 3.3545e-04, 9.1189e-04, 2.4788e-03, 6.7379e-03,
        1.8316e-02, 4.9787e-02, 1.3534e-01, 3.6788e-01, 1.0000e+00, 1.0000e+00,
        1.0000e+00, 1.0000e+00, 1.0000e+00, 1.0000e+00, 1.0000e+00, 1.0000e+00,
        1.0000e+00, 1.0000e+00])
```

所有 Activation function 的圖示如下：



而在實驗中使用的 activation function 的參數都是預測參數，即 Leaky ReLU's negative_slope = 0.01, ELU's α = 1.0。

3. Experimental results (30%)

A. The highest testing accuracy

- Screenshot with two models

	ReLU	ELU	LeakyReLU
EEGNet	0.8703703703703703	0.8324074074074074	0.850925925925926
DeepConvNet	0.8018518518518518	0.8175925925925925	0.8064814814814815

- Anything you want to present

  本次實驗 Model 以外參數的細節如下

  batchsize = 1080

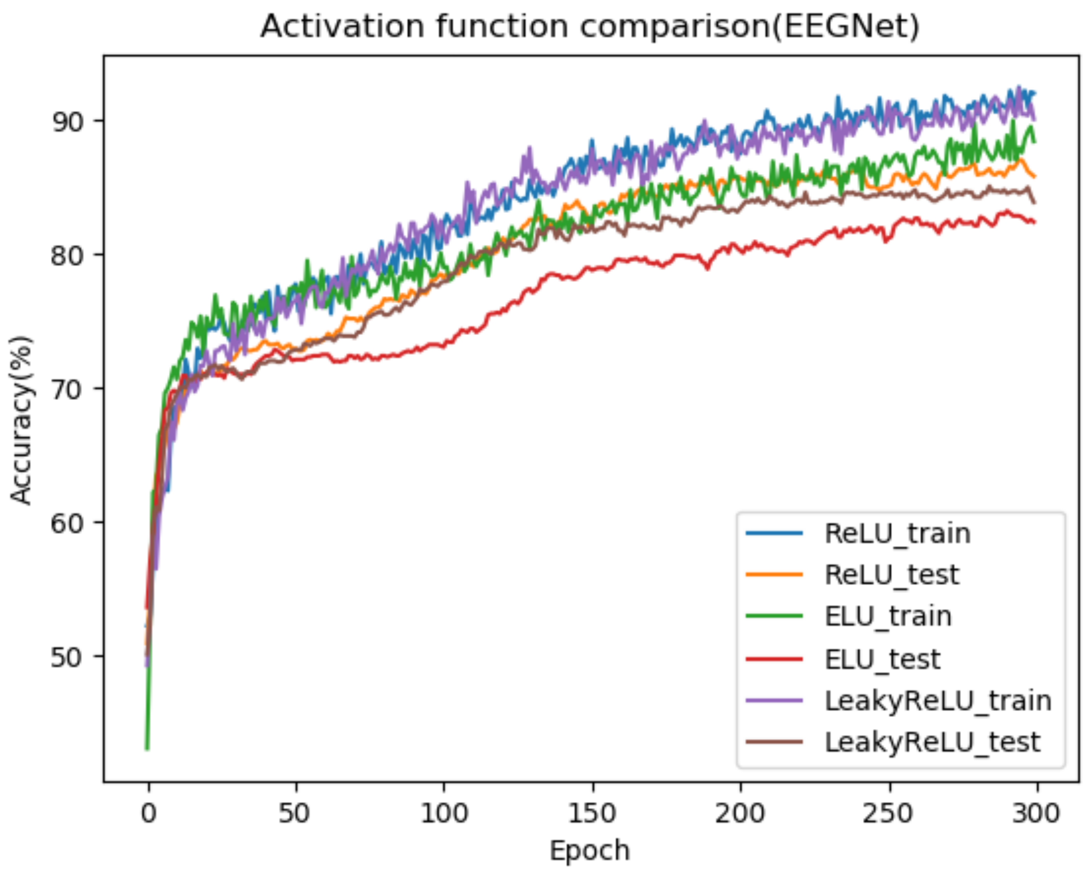
  epochsize = 300

  optimizer's weight_decay = 1e-3

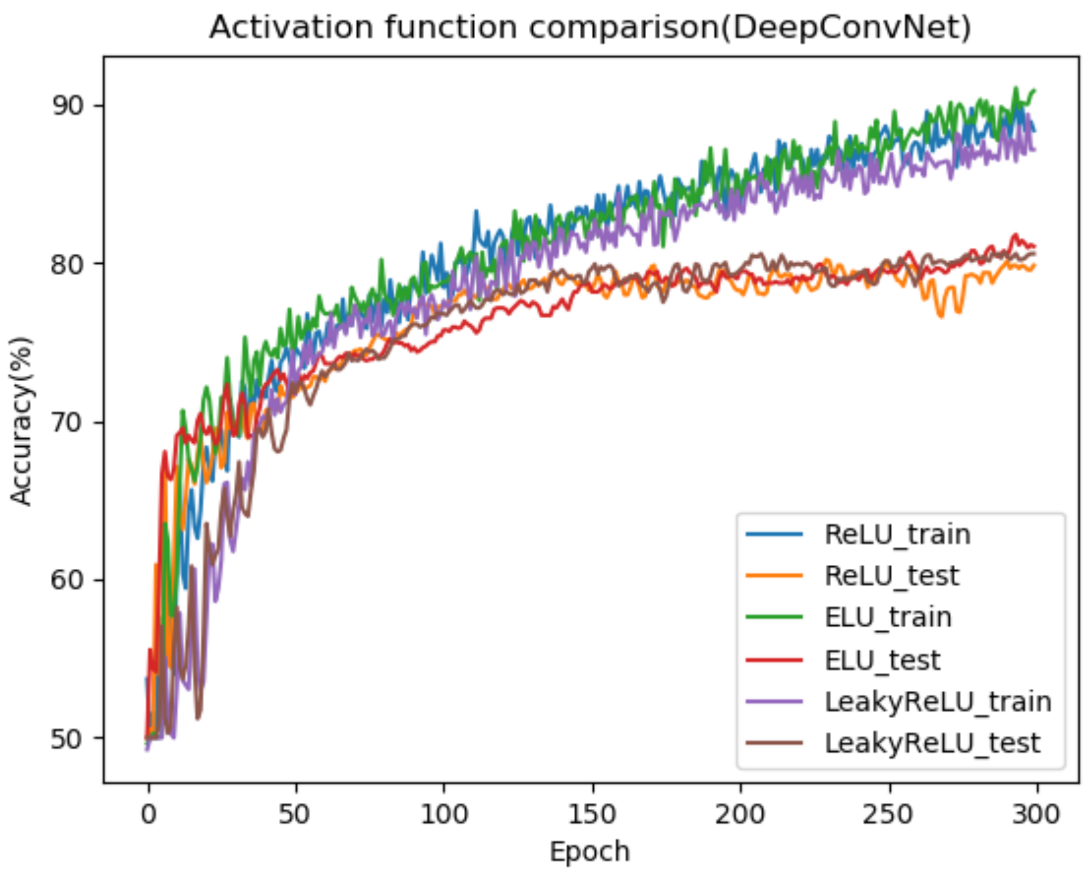
  learning rate = 1e-3

B. Comparison figures

- EEGNet



- DeepConvNet

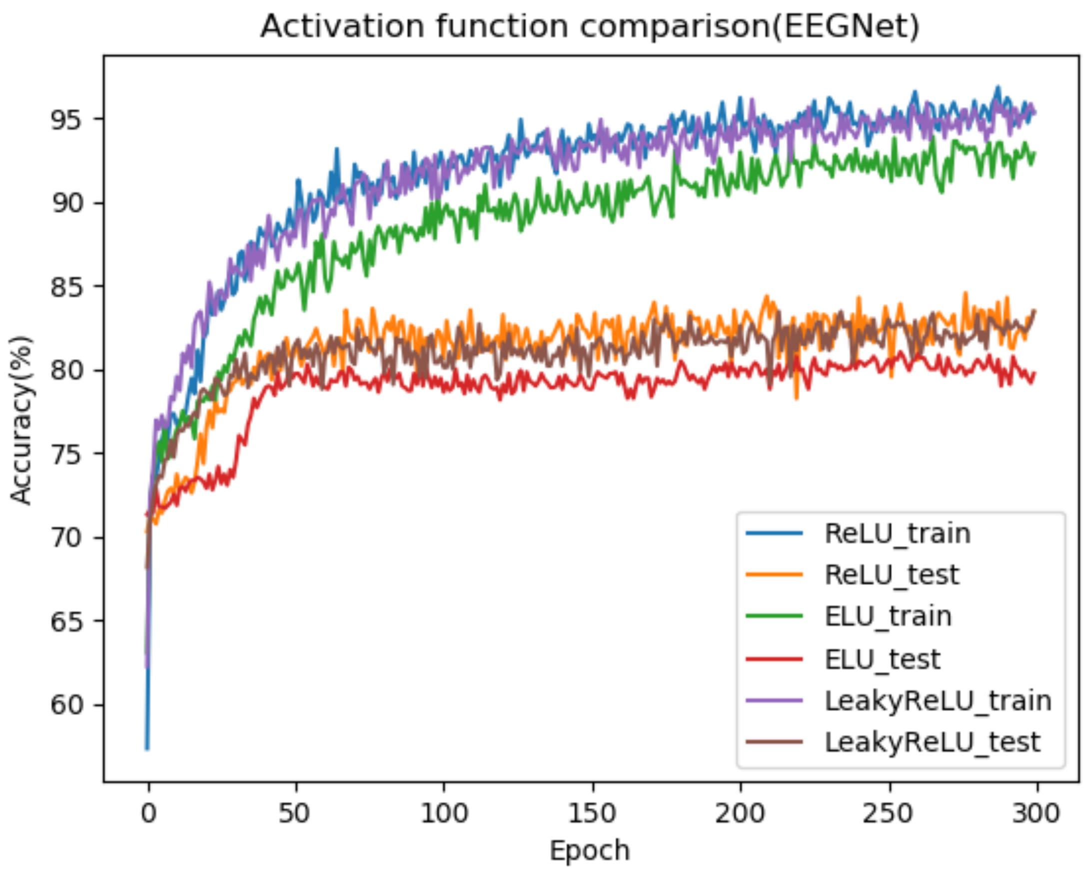


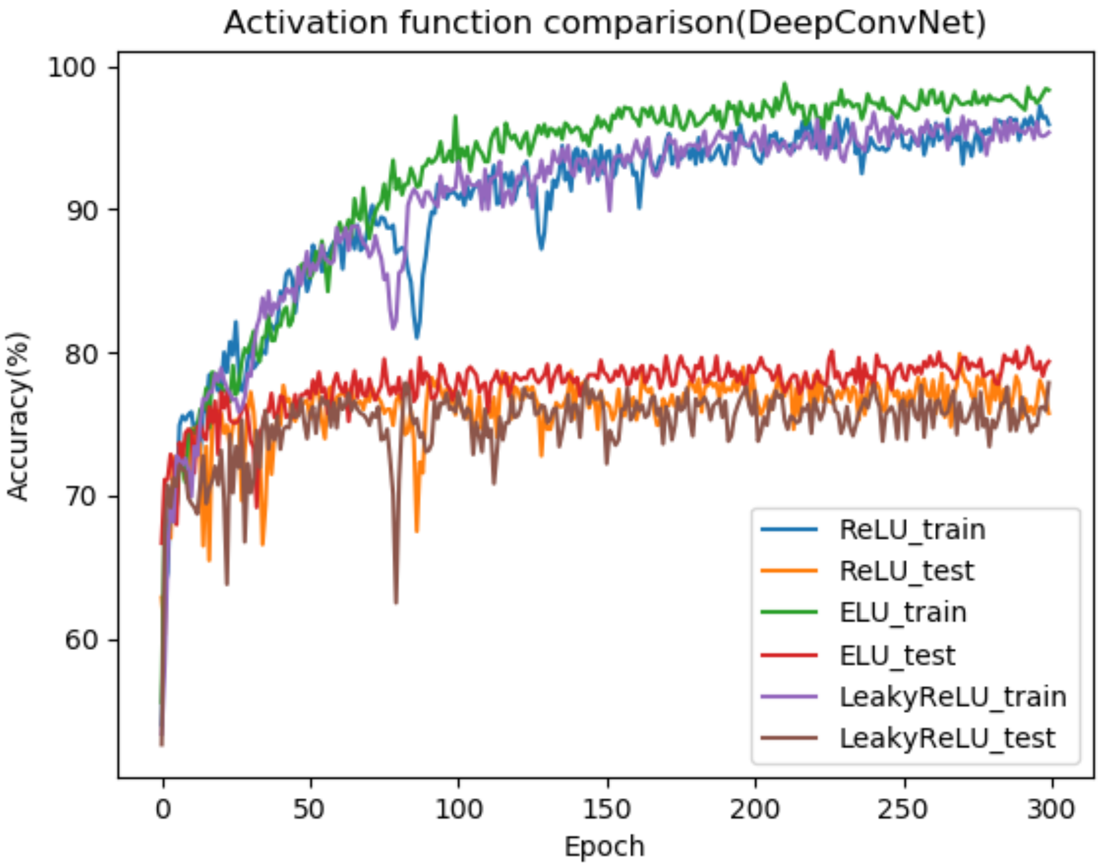
4. Discussion (20%)

   - nn.Module 有分 train mode 跟 test mode, 造成 BatchNorm layer 跟 Dropout layer 有不同的實作方式。 BatchNorm 的 train mode 會去估算資料中的平均數與變異數, test mode 則使 BatchNorm 利用之前估算的平均數與變異數來當作未知資料的平均數與變異數； Dropout 在 train mode 的時候會被激活運作, 在 test mode 則不會運作。
   - 提高對 testing data 準確率的方法包含 regularization, dropout, early stopping, ensembling, data augmentation, feature normalization, cross validation, adjust learning rate, adjust batch size 等等, 而在本次實驗中透過 Adam optimizer 的 weight_decay 使用了 L2 regularization (參考https://stackoverflow.com/questions/42704283/adding-l1-l2-regularization-in-pytorch), 並利用 Dropout來減低 model capacity, 但這個參數不能調太大, 否則準確率反而會下降 (參考https://stats.stackexchange.com/questions/291779/why-accuracy-gradually-increase-then-suddenly-drop-with-dropout)。
   - 調整 batchsize 會影響到整個 process 的計算速度與 converge 的速度(參考https://stats.stackexchange.com/questions/164876/what-is-the-trade-off-between-batch-size-and-number-of-iterations-to-train-a-neu、https://mydeeplearningnb.wordpress.com/2019/02/23/convnet-for-classification-of-cifar-10/)

     Case1. batchsize = 32, 可以看到 EEGNet 跟 DeepConvNet 的 testing epoch 都大約在 50~100 的時候就收斂了, 但整個 process 算完需要花二十幾分鐘。

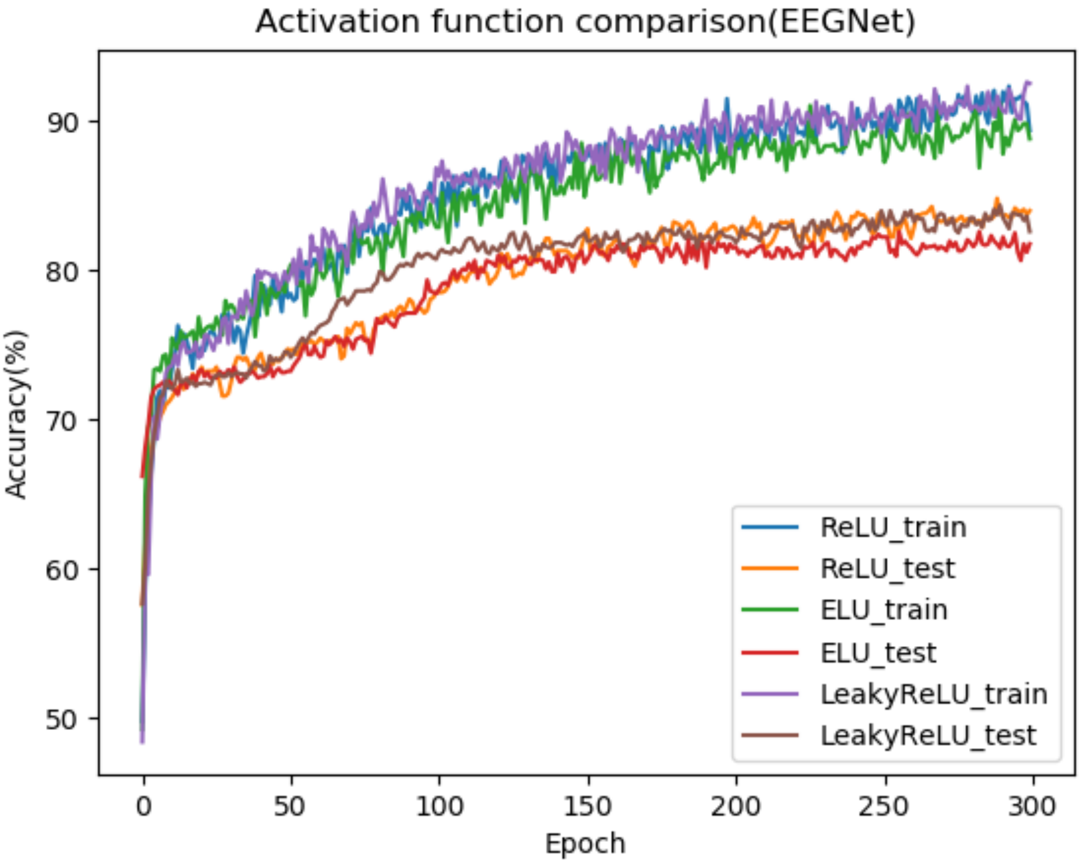
	ReLU	ELU	LeakyReLU
EEGNet	0.8453703703703703	0.8101851851851852	0.8342592592592593
DeepConvNet	0.799074074074074	0.8037037037037037	0.7805555555555556

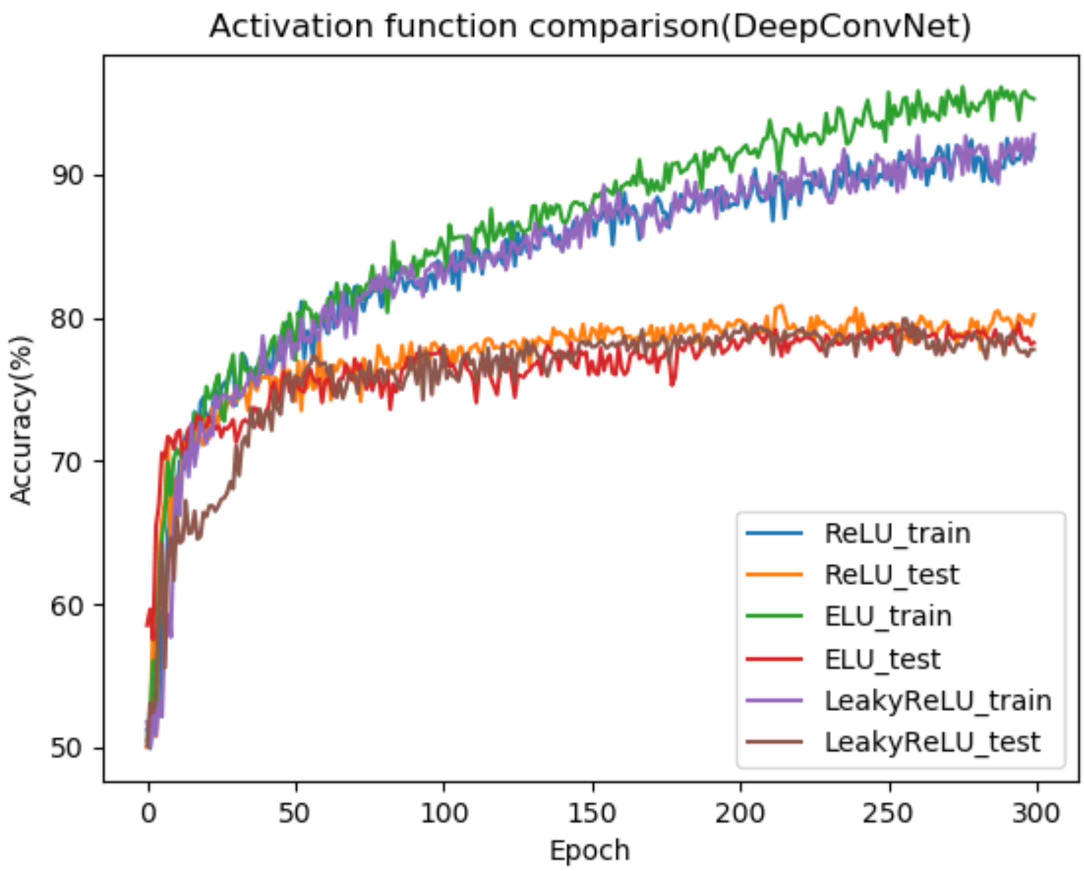




Case 2. batchsize = 512, 可以看到 EEGNet 跟 DeepConvNet 的 testing epoch 都大約要在 150~200 的時候才會收斂(變得比較平緩), 而整個 process 算完只需要十分鐘左右。

	ReLU	ELU	LeakyReLU
EEGNet	0.8481481481481481	0.825925925925926	0.8435185185185186
DeepConvNet	0.8083333333333333	0.7962962962962963	0.799074074074074




Activation function comparison(DeepConvNet)
Accuracy(%)
Epoch
90
80
70
60
50
0
50
100
150
200
250
300
ReLU_train
ReLU_test
ELU_train
ELU_test
LeakyReLU_train
LeakyReLU_test