1.Introduction

這次的作業就是經由給定data，透過EEEGNET和DeepConvNet來做binary classification。
而這次作業主要的目標就是希望學生們能讀懂網路的規格表，實作EEEGNET和DeepConvNet，並且學習如何調hyper parameter。

2.Experiment setup

A. The detail of your model

EEGNet:

其實就照著reference刻，總共可分為firstconv，depthwiseConv，separableConv，flatten和classify共五層。

```
def forward(self,input):
    input1 = self.firstconv(input)
    input2 = self.depthwiseConv(input1)
    input3 = self.separableConv(input2)
    input4 = torch.flatten(input3,start_dim=1)
    output = self.classify(input4)
    return output
```

firstconv這層就是做一次conv和batchNorm。

```
self.firstconv = torch.nn.Sequential(
    torch.nn.Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0,25), bias=False),
    torch.nn.BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

depthwiseConv這層就是做一次conv，batchNorm2d，ELU，AvgPool和Dropout。

```
self.depthwiseConv = torch.nn.Sequential(
    torch.nn.Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False),
    torch.nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
    torch.nn.ELU(alpha=1.0),
    torch.nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
    torch.nn.Dropout(p=0.25)
)
```

separableConv這層基本上和depthwiseConv差不多，只是Conv和AvgPool的大小不太一樣。

```
self.separableConv = torch.nn.Sequential(
    torch.nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False),
    torch.nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
    torch.nn.ELU(alpha=1.0),
    torch.nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
    torch.nn.Dropout(p=0.25)
)
```

然後因為此時輸出還是4dim矩陣，而為了要binary classification，所以我們要透過flatten將4dim矩陣轉成2dim。

最後再做classification，也就單純將channel數從736變成2。

```
self.classify = torch.nn.Sequential(
    torch.nn.Linear(in_features=736, out_features=2, bias=True)
)
```

DeepConvNet:

主要也是照抄，分成network，flatten和classify。

```
def forward(self,input):
    input1 = self.network(input)
    input2 = torch.flatten(input1,start_dim=1)
    output = self.classify(input2)
    return output
```

network的部分，一開始先疊一個conv，後面再疊四次conv->batchnorm->activation_func->maxpool->dropout，然後conv的out_channel也是用後面越大。

```
self.network = torch.nn.Sequential(
    torch.nn.Conv2d(1, 25, kernel_size=(1, 5)),
    torch.nn.Conv2d(25, 25, kernel_size=(2, 1)),
    torch.nn.BatchNorm2d(25, eps=1e-05, momentum=0.1),
    activation_funcion(activation_func),
    torch.nn.MaxPool2d(kernel_size=(1, 2)),
    torch.nn.Dropout(p=0.5),
    torch.nn.Conv2d(25, 50, kernel_size=(1, 5)),
    torch.nn.BatchNorm2d(50, eps=1e-05, momentum=0.1),
    activation_funcion(activation_func),
    torch.nn.MaxPool2d(kernel_size=(1, 2)),
    torch.nn.Dropout(p=0.5),
    torch.nn.Conv2d(50, 100, kernel_size=(1, 5)),
    torch.nn.BatchNorm2d(100, eps=1e-05, momentum=0.1),
    activation_funcion(activation_func),
    torch.nn.MaxPool2d(kernel_size=(1, 2)),
    torch.nn.Dropout(p=0.5),
    torch.nn.Conv2d(100, 200, kernel_size=(1, 5)),
    torch.nn.BatchNorm2d(200, eps=1e-05, momentum=0.1),
    activation_funcion(activation_func),
    torch.nn.MaxPool2d(kernel_size=(1, 2)),
    torch.nn.Dropout(p=0.5)
)
```

然後因為此時輸出還是4dim矩陣，而為了要binary classification，所以我們要透過flatten將4dim矩陣轉成2dim。

最後再做classification，也就單純將channel數從8600變成2。

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

ELU:

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha * (\exp(x) - 1), & \text{if } x \leq 0 \end{cases}$$

他的偏微分是

if x>=0:

1

if x<0:

$$dy/dx = y + \alpha$$

理論上，ELU具有RELU的所有優點，而且不會有dead_relu的問題。
缺點是計算成本有一點高。

Leaky ReLU:

$$\text{LeakyRELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$

偏微分則是:
if x>=0:
1
else:
negative_slope
理論上，Leaky ReLU具有RELU的所有優點，而且不會有dead_relu的問題。

ReLU:

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$

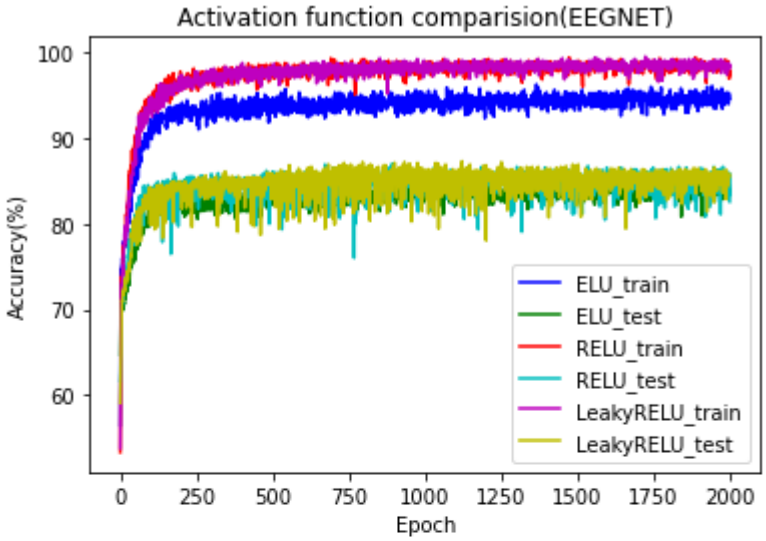
偏微分則是:
if x>=0:
1
else:
0
優點是計算成本小，不存在梯度飽合問題。
缺點是可能發生dead_relu，也就是當x<0，就完全不更新。

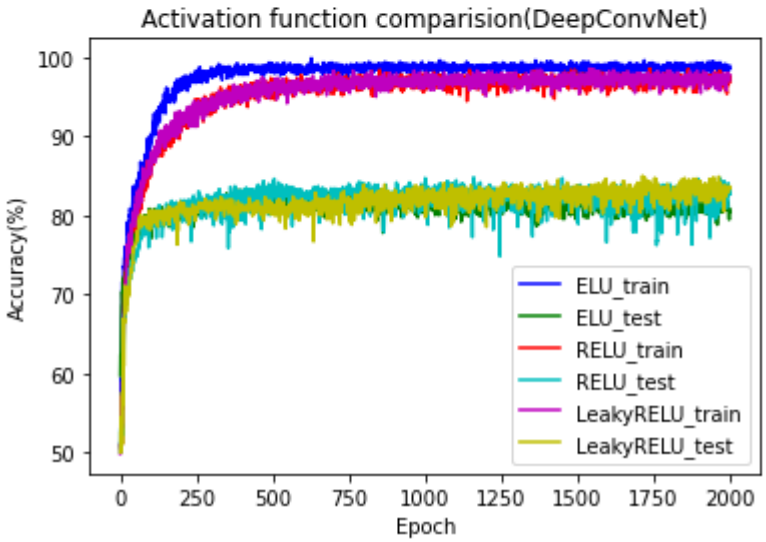
3.Experimental results

A. The highest testing accuracy

```
highest_acc:
EEGNET_ELU_test :  85.83333333333333 %
EEGNET_RELU_test :  87.03703703703704 %
EEGNET_LeakyRELU_test :  87.31481481481481 %
DeepConvNet_ELU_test :  83.61111111111111 %
DeepConvNet_RELU_test :  84.81481481481481 %
DeepConvNet_LeakyRELU_test :  85.0 %
```

B. Comparison figures





4.Discussion

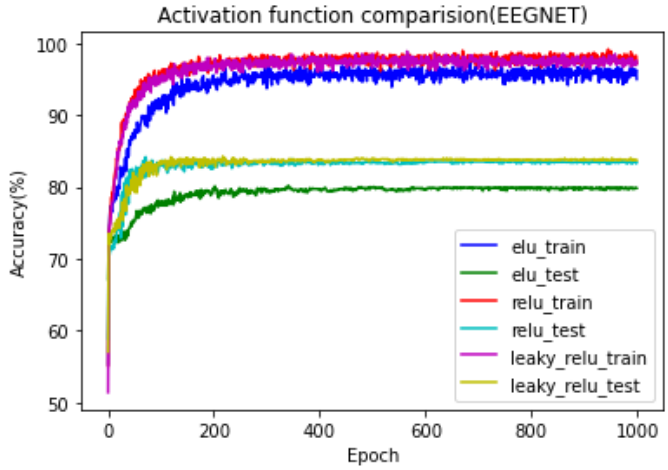
A. Anything you want to share

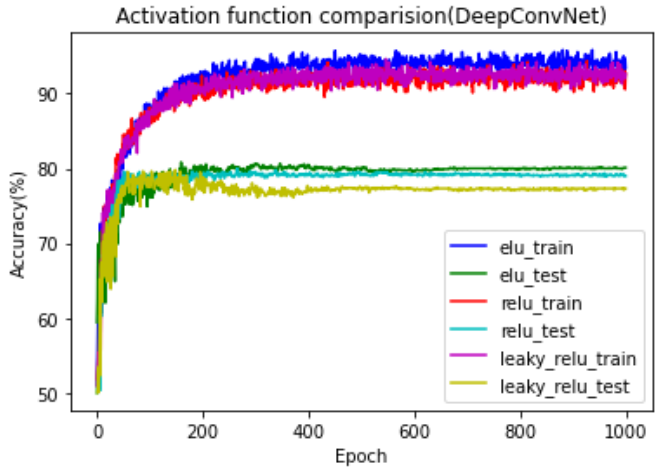
1.

從圖表(3)可以觀察到，DeepConvNet的test_acc一直比EEGNET的test_acc還來的差，這是因為DeepConvNet比較適合用在dataset較大的情況。

值得注意的是，train_acc和test_acc其實也不一定有正相關，像ELU的train_acc整體來說應該是六種裡面最差的，但是他的test_acc在六種裡面是第三好的。

2.我曾經試過加入scheduler，發現其實sheduler對於結果來說影響並不大，如下圖，我對助教給的參數們加入scheduler=stepLR(0.99)，結果和沒加scheduler差不多，所以最後我就拿掉了。






```
highest_acc:
EEGNET_elu_test :  80.18518518518519 %
EEGNET_relu_test :  84.16666666666667 %
EEGNET_leaky_relu_test :  84.25925925925925 %
DeepConvNet_elu_test :  80.83333333333333 %
DeepConvNet_relu_test :  80.0 %
DeepConvNet_leaky_relu_test :  79.81481481481481 %
```

3.寫這份作業時，我有和其他人討論hyperparameter的設置，後來發現就算程式碼和hyperparameter都相同，使用不同gpu好像也會對test_acc影響到2~3%，所以hyperparameter的設置我覺得好像沒這麼絕對。像我為了改良test_rate的acc，我的batch_size用到了400，雖然一般人都用batch_size=32或64，然後跑出了87%，但用我RTX2060跑，最高好像只能到84%。

至於其他hyperparameter，我設置epoch=2000，learning_rate=1e^-3，optimizer=adam()，weight_decay=0.01，這個過程純粹通靈。