Deep Learning
Lab3:EEG classification

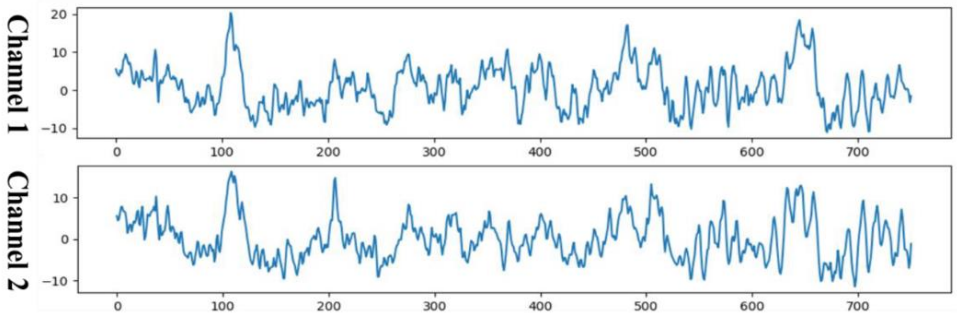
311512064 鄧書桓

1. Introduction:

i. Lab Objective

首先，利用 Pytorch 來建構 EEGNET 和 DeepConvNet 並以此實現 EEG 訊號分類模型。其中，需要使用不同的 Activation function(Relu, Leaky Relu 和 ELU)與觀察各自的結果。最後將結果可視化並秀出最高的準確率之結果。

ii. Dataset



bipolar EEG signal

此 dataset 有兩個通道，每個通道各自有 750 個 data point，兩個分別代表球掉落到左手或右手邊。另外，此 dataset 已被整理成[S4b_train.npz, X11b_train.npz] 和 [S4b_test.npz, X11b_test.npz]。

2. Experiment setups:

A. The detail of your model

i. Load Data

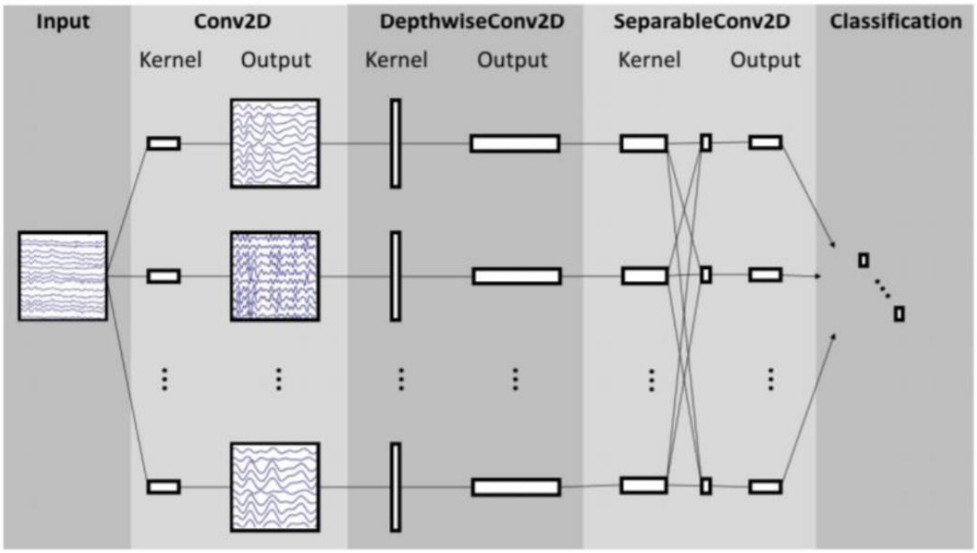
```
# load data (輸入的資料型態須為tensor)
train_data, train_label, test_data, test_label=read_bci_data()

trian_dataset=TensorDataset(torch.Tensor(train_data), torch.Tensor(train_label))
trian_loader = DataLoader(trian_dataset, batch_size = Batch_size, shuffle=True)
total_train_size = len(trian_dataset)
train_evaluate = DataLoader(trian_dataset, batch_size = total_train_size)

test_dataset = TensorDataset (torch.Tensor(test_data), torch.Tensor(test_label))
total_test_size = len(test_dataset)
test_loader=DataLoader(test_dataset, batch_size=total_test_size)
```

使用 tensordataset 與 dataloader 將資料以 tensor 的型態載入，並對每次取樣進行 shuffle，這樣能獲得更好的準確率。

ii. EEGNet



第一層 Conv2D 是用來提取訊號的特徵，第二層 DepthwiseCon2D 將 multi channel 整理成一個 channel，最後一層的 SeperableCon2D 從 DepthwiseConv2D 中提取特徵。

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

上圖為實作的細節，下圖為根據此來設計的 network。

```python
class EEGNet(nn.Module):
    def __init__(self, activation=nn.ELU):
        # super() 是一個內建函數，用於調用父類的方法
        # 在子類中重寫一個父類的方法，需要使用 super() 函數調用父類的同名方法，以保留父類的行為
        # 這邊的self為代表nn.Module
        super(EEGNet, self).__init__()

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

        self.second_layer = nn.Sequential(
            nn.Conv2d(16,32,kernel_size=(2,1),stride=(1,1),groups=16,bias=True),
            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=0.25)
        )

        self.third_layer = nn.Sequential(
            nn.Conv2d(32,32,kernel_size=(1,15),stride=(1,1),padding=(0,7), bias=True),
            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=0.25)
        )

        self.classify=nn.Sequential(
            nn.Flatten(),
            nn.Linear(736, 2, bias=True),
        )
    def forward(self, x):
        output = self.first_layer(x)
        output = self.second_layer(output)
        output = self.third_layer(output)
        output = self.classify(output)
        return output
```

iii. DeepConvnet

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

上圖為實作細節與參數，其中的 C = 2, T = 250 和 N = 2 ，另外 max norm 可以忽略。

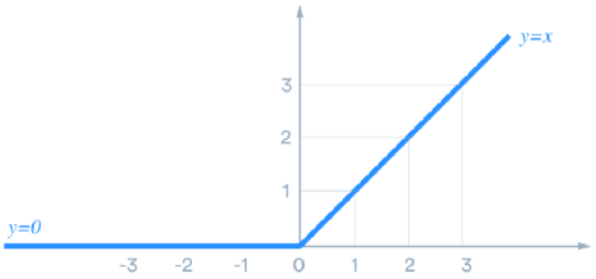
```
class DeepConvNet(nn.Module):
    def __init__(self, activation=nn.ELU):
        super(DeepConvNet, self).__init__()
        self.all_layer = nn.Sequential(
            nn.Conv2d(1, 25 , kernel_size = (1,5), stride = (1,1)),
            nn.Conv2d(25, 25, kernel_size = (2,1), stride = (1,1), bias=True),
            nn.BatchNorm2d(25, eps=1e-05, momentum=0.1, affine = True, track_running_stats = True),
            activation(),
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Conv2d(25, 50, kernel_size = (1,5), stride = (1,1), bias=True),
            nn.BatchNorm2d(50, eps=1e-05, momentum=0.1, affine = True, track_running_stats = True),
            activation(),
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Conv2d(50, 100, kernel_size = (1,5), stride = (1,1), bias=True),
            nn.BatchNorm2d(100, eps=1e-05, momentum=0.1, affine = True, track_running_stats = True),
            activation(),
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Conv2d(100, 200, kernel_size = (1,5), stride = (1,1), bias=True),
            nn.BatchNorm2d(200, eps=1e-05, momentum=0.1, affine = True, track_running_stats = True),
            activation(),
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Flatten(),
            nn.Linear(8600,2),
        )

    def forward(self, input):
        out = self.all_layer(input)
        return out
```

上圖為實作細節。

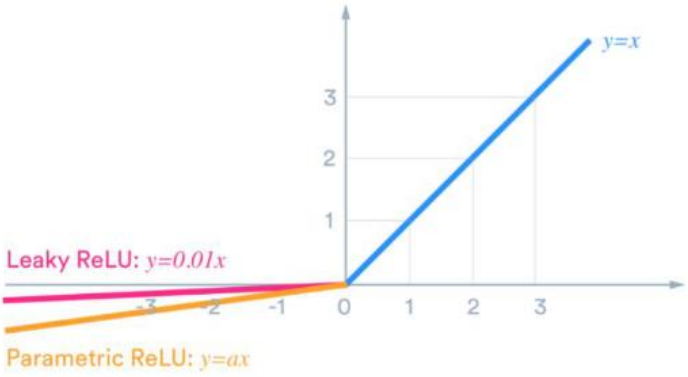
B. Explain the activation function (ReLU, Leaky ReLU, ELU)

i.Relu:



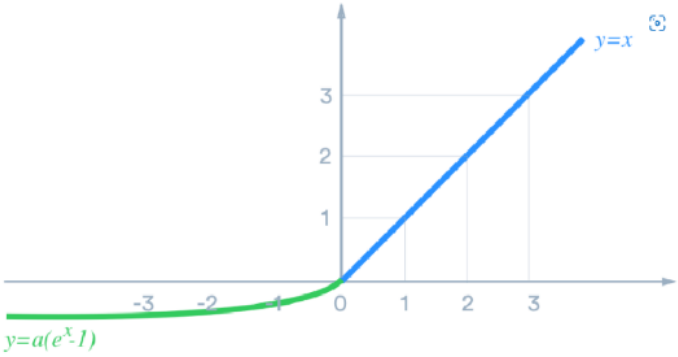
Relu 為 Rectified Linear Units 的縮寫，特點為會將所有負數變成 0，正數維持不變。優點為計算快速與 sparsely activated。缺點為正值可能會無限大，而負數更改為 0 會導致斜率變成 0，導致無法更新該部分的 weight，導致 dead neuron 的產生。

ii.Leaky Relu



Leaky Relu 修正了 dying relu 的問題，防止輸出為負號時永遠無法被激活的問題，另外也繼承 Relu 的優點。但是，在實際應用上並無法證實哪者較優，因此還須根據不同狀況而定。

iii. ELU

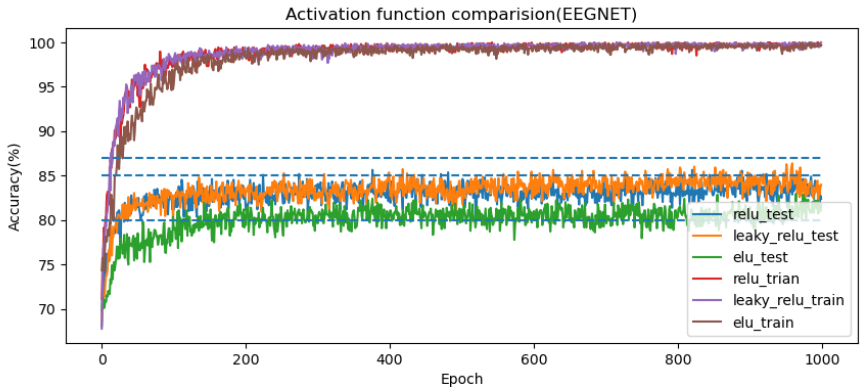


ELU 也是為了解決 Relu 的問題而提出的，同時結合上述兩者的優點，但是會在負值很大時飽和。

3. Experimental results

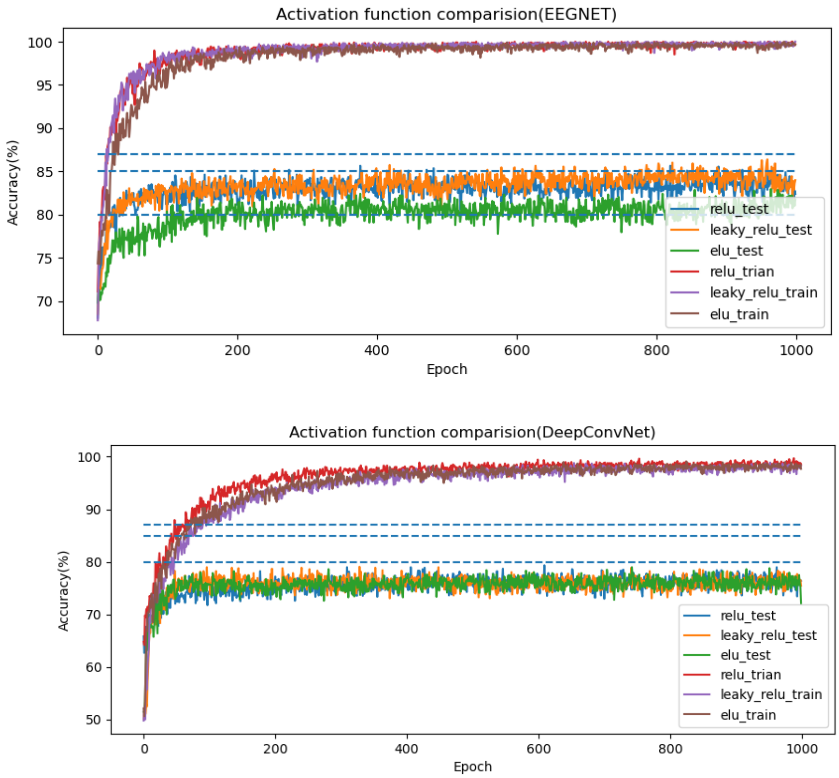
A. The highest testing accuracy

分數最高的為 EEGNet，當 activation function 為 Leaky Relu，準確率為 86.39%，其突入下所示:




```
(base) widden@widden-home:~/NCTU/class/Deep_Learning/lab3$ python lab3.py
Epoch 1000: 100%|                                        | 1000/1000 [01:03<00:00, 15.68it/s]
Epoch 1000: 100%|                                        | 1000/1000 [01:00<00:00, 16.51it/s]
Epoch 1000: 100%|                                        | 1000/1000 [01:01<00:00, 16.33it/s]
最高準確率: 86.38888888888889 %  ,active function為: EEGLeakyrelu
```

B. Comparison figures:



EEGNet 與 deepconvnet 相比擁有較高的準確度，且在 train 時的收斂也較快。

4. Discussion

這次是我第一次接觸到 pytorch，與前幾次自己刻 model 相比，真的快很多也很方便，尤其是其中的反向傳播更新函式，我一開始還把 zero_grad 放在 backward 後面，導致預測出來的準確率只有 50%，幸好後來有成功找到問題。另外，一開始的 dataloader 也讓我弄了很久，需要將資料先變成 torch 的形式才能做之後的動作，讓我學到了這寶貴的一課。最後，本次 train 的 network 應該滿小的，我用自己的電腦跑不到 3 分

鐘就結束了，希望之後可以試試看更大一點的。