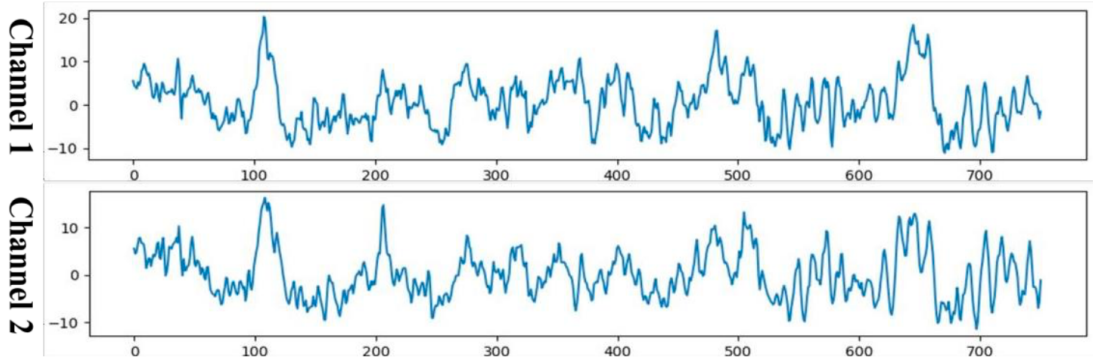
Lab3 EEG classification

鄧奕辰 311605014

1. Introduction

這次的實驗包含了 EEGNet 與 DeepConvNet 兩種模型的比較，其中也包含不同 activation function 的比較。此實驗的訓練資料為下圖所示，



訓練資料的大小為 (C=1, H=2, W=750)，訓練資料與測試資料各有 1080 筆，訓練目標為使用不同 3 種不同的 activation functions 來獲得最高的準確度。

2. Experimental setup

EEGNet 的模型為下圖所示，

```
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)
  )
)
```

EGGNet 作為分析電圖信號的腦電波型態，與一般的 CNN 不同的是，使用了 Depthwise Separable Convolution 的捲機操作，以提高模型的效率的準確性。

```python
class EEGNet(nn.Module):
    def __init__(self, activation=nn.ReLU()):
        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)
        )
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1),
                      stride=(1, 1), groups=16, bias=False),
            nn.BatchNorm2d(32),
            activation,
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(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),
            activation,
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p=0.25)
        )
        self.classify = nn.Sequential(
            nn.Linear(736, 2, bias=True)
        )

    def forward(self, x):
        x = self.firstConv(x)
        x = self.depthwiseConv(x)
        x = self.separableConv(x)
        x = x.view(x.shape[0], -1)
        x = self.classify(x)
        return x
```

第一層（Conv2D）：卷積層，用於提取輸入訊號的空間特徵。EEGNet 中的卷積層採用 1D 卷積核，可以捕捉輸入訊號中的時域特徵。

第二層（DepthwiseConv2D）：深度卷積層，可以在不增加參數量的情況下增加模型的深度。DepthwiseConv2D 會對每個輸入通道應用獨立的卷積核，以提取更多的特徵信息。

第三層（SeparableConv2D）：輕量級的卷積層，可以在保持模型參數量相對較小的情況下提高模型的複雜度。SeparableConv2D 可以將每個卷積核分解成深度卷積和逐點卷積兩個步驟進行計算，以減少參數量和計算量。

第四層（Dense）：全連接層，用於學習高層次的特徵表示和分類器。

DeepConvNet 的模型為下圖所示，

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

DeepConvNet 作為分析電圖信號的腦電波型態，與傳統淺神經層不同，使用了多個卷積層與池化層，並採用了稱為“通道優先”的輸入數據格式，用於加速並行運算。

```
class DeepConvNet(nn.Module):
    def __init__(self, activation=nn.ReLU):
        super(DeepConvNet, self).__init__()
        self.conv0 = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=25, kernel_size=(1,5)),
            nn.Conv2d(in_channels=25, out_channels=25, kernel_size=(2,1)),
            nn.BatchNorm2d(25, eps=1e-5, momentum=0.1),
            activation
        )
        self.conv1 = nn.Sequential(
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Conv2d(in_channels=25, out_channels=50, kernel_size=(1,5)),
            nn.BatchNorm2d(50, eps=1e-5, momentum=0.1),
            activation
        )
        self.conv2 = nn.Sequential(
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Conv2d(in_channels=50, out_channels=100, kernel_size=(1,5)),
            nn.BatchNorm2d(100, eps=1e-5, momentum=0.1),
            activation
        )
        self.conv3 = nn.Sequential(
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Conv2d(in_channels=100, out_channels=200, kernel_size=(1,5)),
            nn.BatchNorm2d(200, eps=1e-5, momentum=0.1),
            activation
        )
        self.conv4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=(1,2)),
            nn.Dropout(p=0.5),
            nn.Flatten(),
        )
        self.classify = nn.Linear(8600, 2)

    def forward(self, x):
        x = self.conv0(x)
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.classify(x)
        return x
```

資料讀取：

```
train_data, train_label, test_data, test_label = read_bci_data()
    dataset = TensorDataset(torch.from_numpy(train_data),
                            torch.from_numpy(train_label))
    train_dataset = DataLoader(dataset, batch_size=64, shuffle=True)
```

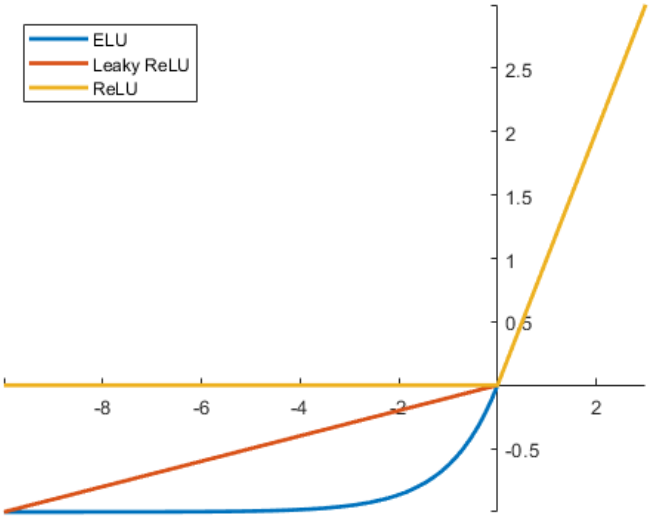
使用助教所提供的 dataloader 中 read_bci_data()的函數來取得訓練與測試的資料

3. Explain the activation function

ReLU: $f(x) = \max(0, x)$

Leaky ReLU: $f(x) = \max(0.1x, x)$

ELU: $f(x) = \max(0, x) + \min(0, \alpha(e^x - 1))$



ReLU（Rectified Linear Unit）：將負輸入值截斷為零，將正輸入值通過線性函數輸出。簡單高效，避免梯度消失問題，但也存在死亡 ReLU 神經元的問題。

LeakyReLU：在負輸入值上引入一個小的斜率，即在輸入為負數時乘以一個很小的常數，從而避免死亡神經元的問題。LeakyReLU 相較於 ReLU 表現更穩定，通常能提高模型的性能。

ELU（Exponential Linear Unit）：在負輸入值上引入指數形式的非線性，從而避免死亡神經元的問題，並且對負輸入值有一個平滑的負斜率。ELU 的表現比 ReLU 和 LeakyReLU 更好，但計算複雜度也相對較高。

4. Experimental result

For EGGNet:

```
ReLU  best accuracy EEG:  88.05555555555556 %
LeakyReLU  best accuracy EEG:  88.42592592592592 %
ELU  best accuracy EEG:  85.0 %
```

For DeepConvNet:

```
ReLU  best accuracy Deep:  85.27777777777777 %
LeakyReLU  best accuracy Deep:  84.81481481481481 %
ELU  best accuracy Deep:  82.87037037037037 %
```

	ReLU	Leaky ReLU	ELU
EEGNet	**88.1%**	**88.4%**	74.1%
DeepConvNet	**85.3%**	84.8%	82.9%

Hyperparameter:

Epoch: 350

Optimizer: Adam

Criterion: CrossEntropy

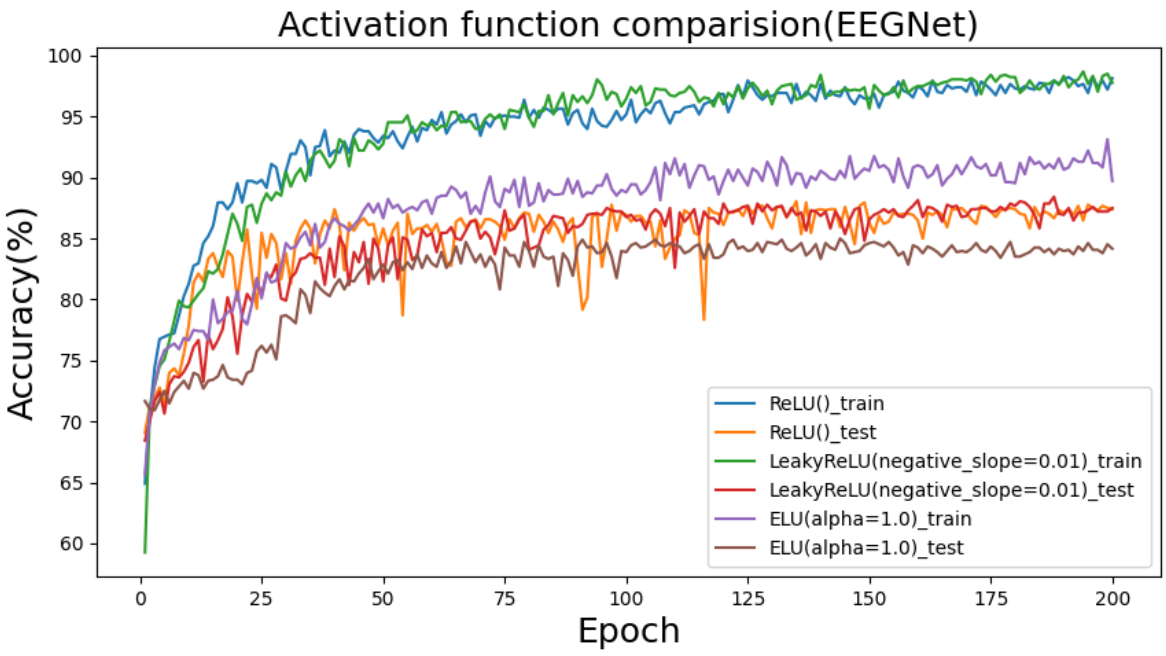
Batch size: 64

Learning rate for EGGNet: 0.001

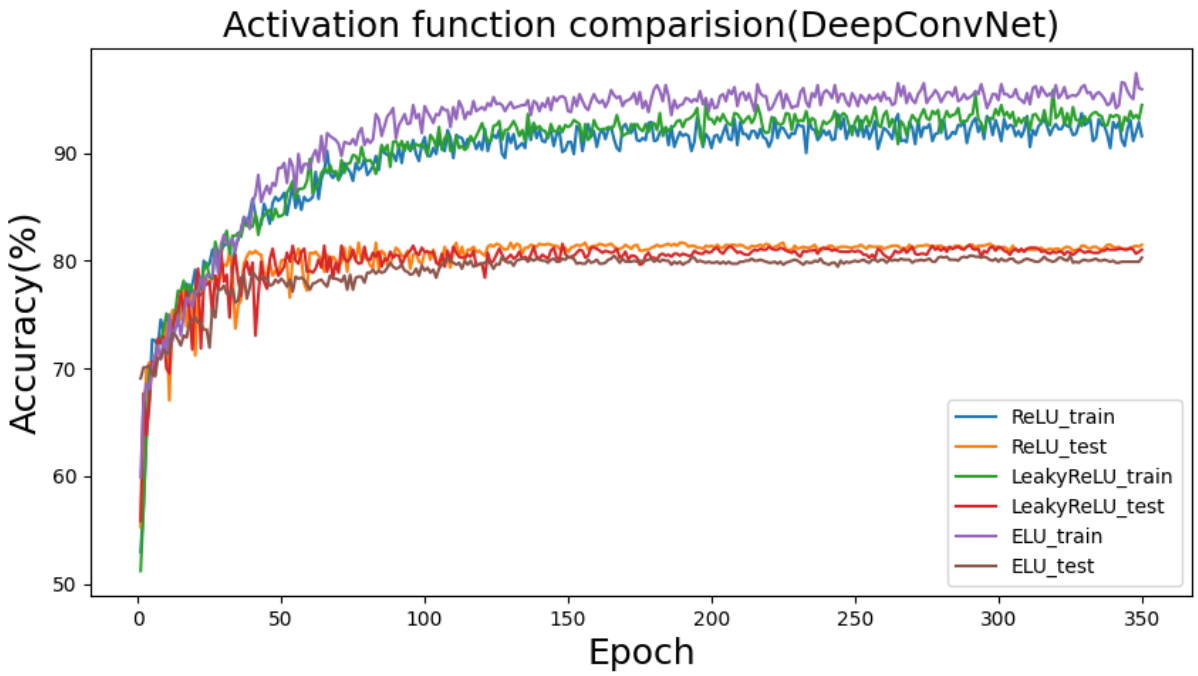
Learning rate for DeepConvNet: 0.001

5. Comparison figures

EEGNet:



DeepConvNet:



6. Discussion

- 在訓練的過程中，有發現已經訓練到 200~300 epoch 仍有震盪的情形，因此我對 learning rate 做逐步降低的規劃，如下圖所示，每 30 個 epoch 會將 learning rate * 0.8，藉此來完成 model 在後期時的細微優化。

```
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',
                                                       factor=0.8, patience=30,
                                                       verbose=False, min_lr=0.00001,
                                                       cooldown=10)
```

- 讀取資料時的資料型態為 numpy，因此須先用 Tensordataset()轉化成 tensor 格式
- Model 需使用 model.to(device)來調用 GPU，加速運算過程。
- 在兩個 model 的 evaluation 過程中，ReLU 都有更好的表現，雖然 Leaky ReLU, ELU 都有針對 ReLU 的缺點做些改善，但在實際訓練過程中，還是有其他參數的影響，不一定換一個 activation function 就一定會提升準確率，且 ELU 在所有實驗中，都有較差的表現。