## I. Introduction:

將 BCI 腦波圖的 dataset 丟進 EEGNet 和 DeepConvNet 做訓練，以成功做出 classification。

EEGNet 使用了 Depthwise Convolution layer 以及 Separable Convolution layer 的架構，而 DeepConvNet 則是用一般的 Convolution layer 去建構。

## II. Experiment setup:

A. The detail of model

EEGNet: (Lr= 0.001, Batch_size= 64, Epoch= 600)

```
EEGNet(
  (conv1): 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)
  )
  (conv2): 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, inplace=False)
  )
  (conv3): 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, inplace=False)
  )
  (out): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

首先，對資料做預處理，使用 pytorch 的 dataloader 工具包，將 train_data, train_label, test_data, test_label 轉換成 torch TensorDataset 的 format，

接著建立網路的架構，這邊就是依照 ppt 上的 architecture 去做，其中包含三個捲積層，一個全連接層。

在 train 的過程中，包含兩個 for loop，外層控制 epoch，內層控制 iteration，每個 iteration 當中，先將 loader 中的資料包成 Variable，接著將 model parameter 的 gradient 歸 0，避免有累加的情形，並將經過 Vaiable 包住的 data 丟進神經網路中訓練，接著採 cross entropy 計算 Loss，並使用 loss.backward()函式做 backpropogation，如此反覆跌代完成 training。

DeepConvNet: (Lr= 0.01, Batch_size= 256, Epoch= 600)

```
DeepConvNet(
  (conv1): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=(0, 25), bias=False)
  )
  (conv2): Sequential(
    (0): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), bias=False)
    (1): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 4), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), padding=(0, 7), bias=False)
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 8), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv4): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), padding=(0, 7), bias=False)
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 8), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (conv5): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), padding=(0, 7), bias=False)
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
```

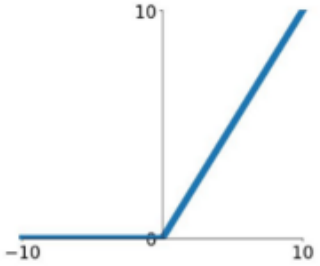
```
  (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 8), padding=0, dilation=1, ceil_mode=False)
  (4): Dropout(p=0.5, inplace=False)
)
(out): Sequential(
  (0): Linear(in_features=400, out_features=2, bias=True)
)
```

將網路的架構稍微做了更動，只使用單純的 Convolution layer 去做訓練，kernal size 縮小，並多加了兩層的 convolution layer。

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

**ReLU:**

$$\mathrm{ReLU}(x) = \max(0, x)$$



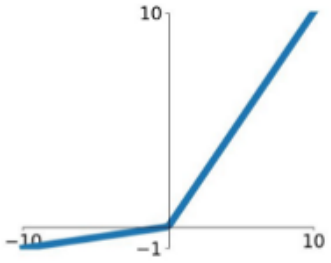

當事件的值大於 0 的時候，將完整事件輸出，否則就以 0 作為這個函式的輸出，代表這件事不被參考。

相較於 sigmoid, tanh 等 activation function，ReLU 的分段線性性質能有效的克服梯度消失的問題，且 Relu 會使部分神經元的輸出為 0，可以讓神經網路變得稀疏，緩解過度擬合的問題，且計算量小，不用指數運算，但衍生出另一個問題是，如果把一個神經元停止後，就難以再次開啟。

**Leaky ReLU:**

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

By default, the negative slope = 0.01

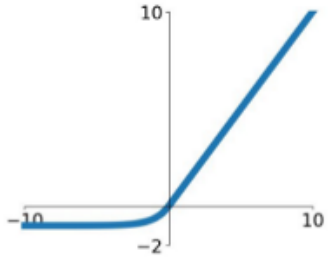



ReLU 是將所有的負值都設為零，而 Leaky ReLU 是給所有負值賦予一個非零斜率，解決了 ReLU 如上所述之問題。

**ELU:**

$$\mathrm{ELU}(x) = \max(0, x) + \min(0, \alpha * (\exp(x) - 1))$$

The α value for the ELU formulation.Default: 1.0




α 是一個可調整的參數，它控制著 ELU 負值部分在何時飽和。

右側線性部分使得 ELU 能夠緩解梯度消失，ELU 的輸出均值接近於零，收斂速度較 ReLU 和其變形(eg. LeakyReLU..)來的快。

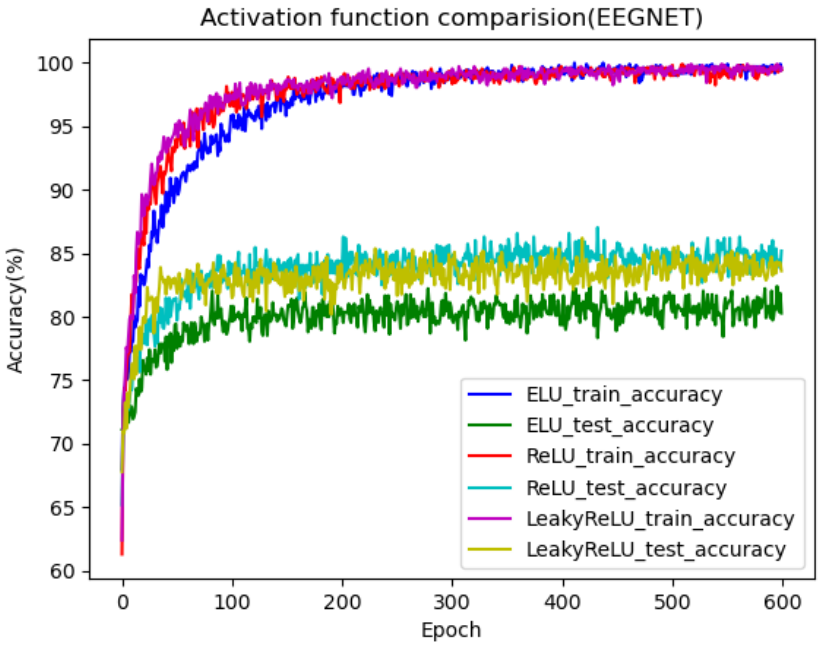
## III. Experimental results

A. The highest testing accuracy: 87.87%

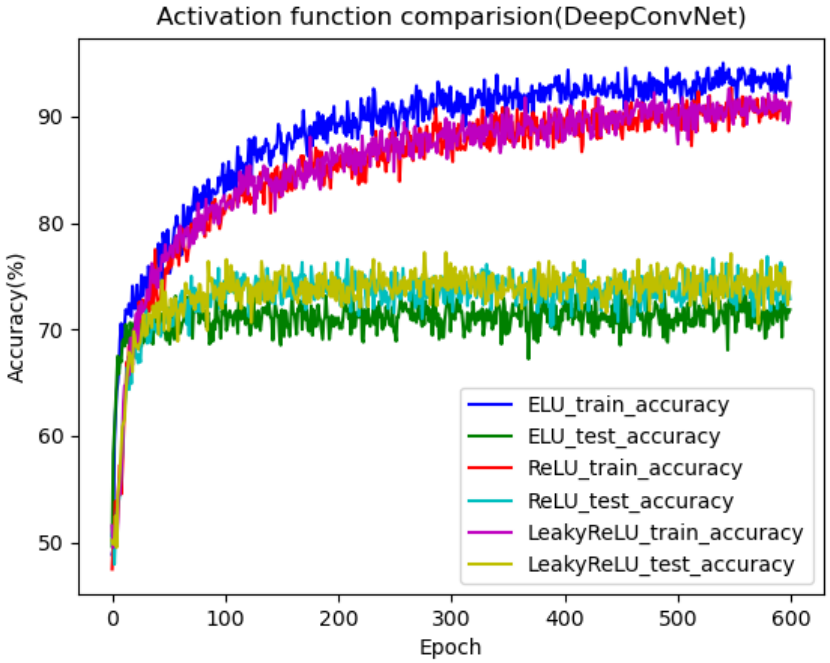
| | ELU | ReLU | LeakyReLU |
|---|---|---|---|
| EEGNet:<br>Lr= 0.001,<br>Batch_size= 64,<br>Epoch= 600 | 82.41 % | 87.04 % | 86.20 % |
| DeepConvNet:<br>Lr= 0.01,<br>Batch_size= 256,<br>Epoch= 600 | 74.63% | 76.85 % | 77.22 % |

B. Comparison figures

EEGNet:




DeepConvNet:




## IV. Discussion

一開始在 train 時，忘記使用 batch 分批做訓練，得到的最高的 acc 只有 80 點幾趴，在思考如何提高 acc 的時候才想到可以用分批去做，在 batch_size=64 時，acc 已經可以來到 87%，且跑的速度快上許多。另外，在 applied 不同的

activation function 時，一開始是同樣的 net 寫三次，代換裡面的 activation function，後來才找到 nn.ModuleDict 這個函式，能用變數的方式直接在 net 中做代換的動作。