1. Introduction

EEGNet – 腦神經介面(BCI)，他使用腦神經活動作為控制訊號，以此實現與電腦的直接連結。該訊號通常來自腦電波圖（EEG）的訊號。對於 BCI 實例，特徵提取和分類，會對預期的 EEG 控制信號的不同特徵而制定。常用於電腦視覺和語音辨識的卷積類神經網絡（CNN）已成功應用在基於 EEG 的 BCI 資料，而 EGGNet 設計單個 CNN 架構來準確分類來自不同 BCI 的 EEG 訊號，是一種緊湊卷積類神經網絡。EGGNet 引入了深度和可分離卷積的使用概念，來構建 EEG 特定模型，模型封裝了 BCI 的 EEG 特徵提取。

DeepConvNet – 為一卷機網絡，該卷積類神經網絡主要用於對影像進行分類，透過相似性（ex: 影像比對）對它們進行聚類，並在不同情境中執行識別。可以是臉部辨識、個體辨識等等。該網絡讓字符識別成為可能，像是手寫辨識、自然語言處理。現今該網絡已直接應用於文本分析以及影像數據分析。ConvNets 在影像辨識中的能力深度學習崛起的主要原因之一。它推動了電腦視覺（CV）的重大進程，在自動駕駛車，機器人，無人機，國防安全，醫療

診斷以及視障治療方面有越來越明顯有效的應用。

2. Experiment setups

A. The detail of your model

i. EEGNet

```
class EEGNet(nn.Module):
    def __init__(self):
        super(EEGNet, self).__init__()
        # Layer 1
        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))

        # Layer 2
        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),
            nn.ELU(alpha=1.0),
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4),padding=0),
            nn.Dropout(p=0.25))

        # Layer 3
        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),
            nn.ELU(alpha=1.0),
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1,8), padding=0),
            nn.Dropout(p=0.25))

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

```
def forward(self, x):
    # Layer 1
    x = self.firstconv(x)
    # Layer 2
    x = self.depthwiseConv(x)
    # Layer 3
    x = self.separableConv(x)

    # FC Layer
    x = x.view(x.size(0),-1)
    x = self.classify(x)

    return x
```

ii. DeepConvNet

```
class DeepConvNet(nn.Module):
    def __init__(self):
        super(DeepConvNet, self).__init__()

        #Layer initial
        self.initiallayer = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5), stride=(1,1), bias=False)
         )

        # Layer 1
        self.firstlayer = nn.Sequential(
            nn.Conv2d(25, 25, kernel_size=(2, 1), stride=(1,1), bias=False),
            nn.BatchNorm2d(25, eps=1e-05, momentum=0.1 , affine=True, track_running_stats=True),
            nn.LeakyReLU(negative_slope=0.01),
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False),
            nn.Dropout(0.5)
        )

        # Layer 2
        self.secondlayer = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size=(1, 5), stride=(1,1),bias=False),
            nn.BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            nn.LeakyReLU(negative_slope=0.01),
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False),
            nn.Dropout(p=0.5))
```

```
# Layer 3
self.thirdlayer = nn.Sequential(
    nn.Conv2d(50, 100, kernel_size=(1, 5),stride=(1,1),padding=(0, 25),bias=False),
    nn.BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
    nn.LeakyReLU(negative_slope=0.01),
    nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False),
    nn.Dropout(p=0.5))

# Layer 4
self.fourthlayer = nn.Sequential(
    nn.Conv2d(100, 200, kernel_size=(1, 5),stride=(1,1),padding=(0, 25),bias=False),
    nn.BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
    nn.LeakyReLU(negative_slope=0.01),
    nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False),
    nn.Dropout(p=0.5))

# FC Layer
self.classify = nn.Sequential(
    nn.Linear(in_features=16000, out_features=1, bias=True),
    )
```

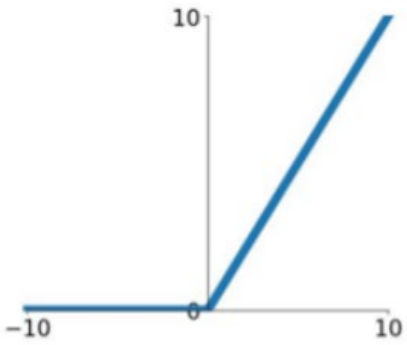
```
def forward(self, x):
    # Layer initial
    x = self.initiallayer(x)
    # Layer 1
    x = self.firstlayer(x)
    # Layer 2
    x = self.secondlayer(x)
    # Layer 3
    x = self.thirdlayer(x)
    # Layer 4
    x = self.fourthlayer(x)

    # FC Layer
    x = x.view(x.size(0),-1)
    x = self.classify(x)

    return x
```

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

i. ReLU

ReLU

$$\max(0, x)$$



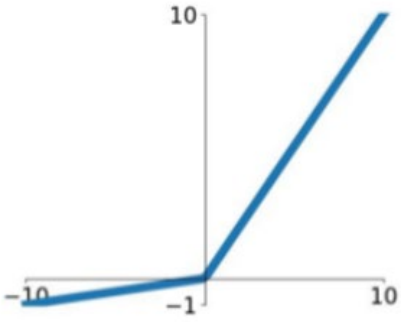
ReLU 是目前世界上使用最多的激活函數，幾乎用於所有捲積神經網絡或深度學習。如上圖所

述，假設一 R(x)，當 x 小於零時，f（x）為零，當 x 大於或等於零時，f（x）等於 x。他的範圍介於[0~無窮大]，但問題是所有負值立即變為零，降低了模型適當擬合或訓練數據的能力。

ii. Leaky ReLU

Leaky ReLU

$$\max(0.1x, x)$$

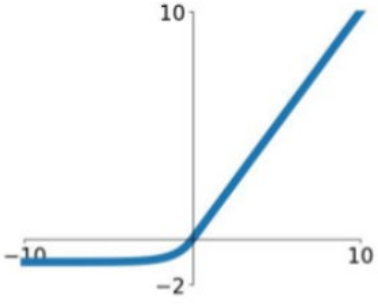


為了解決 Dead ReLU Problem，後來的研究者提出了將 ReLU 的前半段設為 0.1x 而非 0。理論上來講，Leaky ReLU 有 ReLU 的所有優點，外加不會有 Dead ReLU 問題，但是在實際操作當中，並沒有完全證明 Leaky ReLU 總是好於 ReLU。

iii. ELU

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ELU 也是為解決 ReLU 存在的問題而提出，顯然，ELU 有 ReLU 的基本所有優點，同 Leaky ReLU 一樣，不會有 Dead ReLU 問題，至於輸出的均值接近 0，zero-centered。

它的一個小問題在於計算量稍大。類似於 Leaky ReLU，理論上雖然好於 ReLU，但在實際使用中目前亦沒有好的證據 ELU 總是優於 ReLU。

3. Experimental results

A. The highest testing accuracy

i. Screenshot with two models

```
print ("ACC of EEGNet: ", ACC)
```

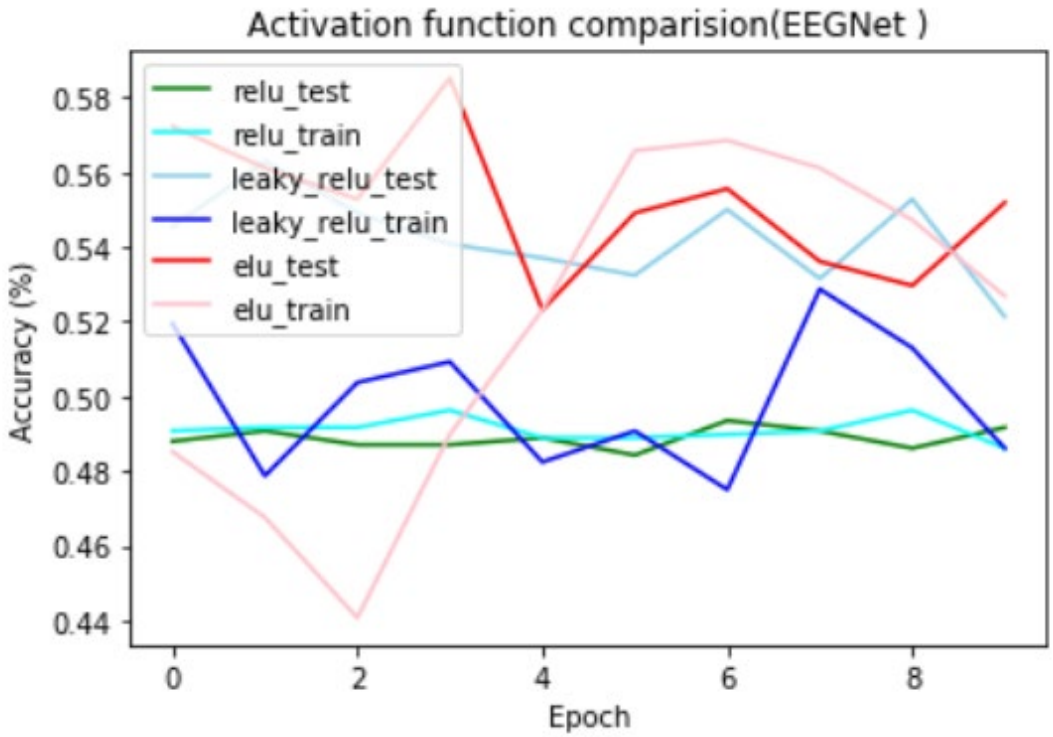
```
ACC of EEGNet:  0.5518518518518518
```

```
print ("ACC of DeepConvNet: ", ACC)
```

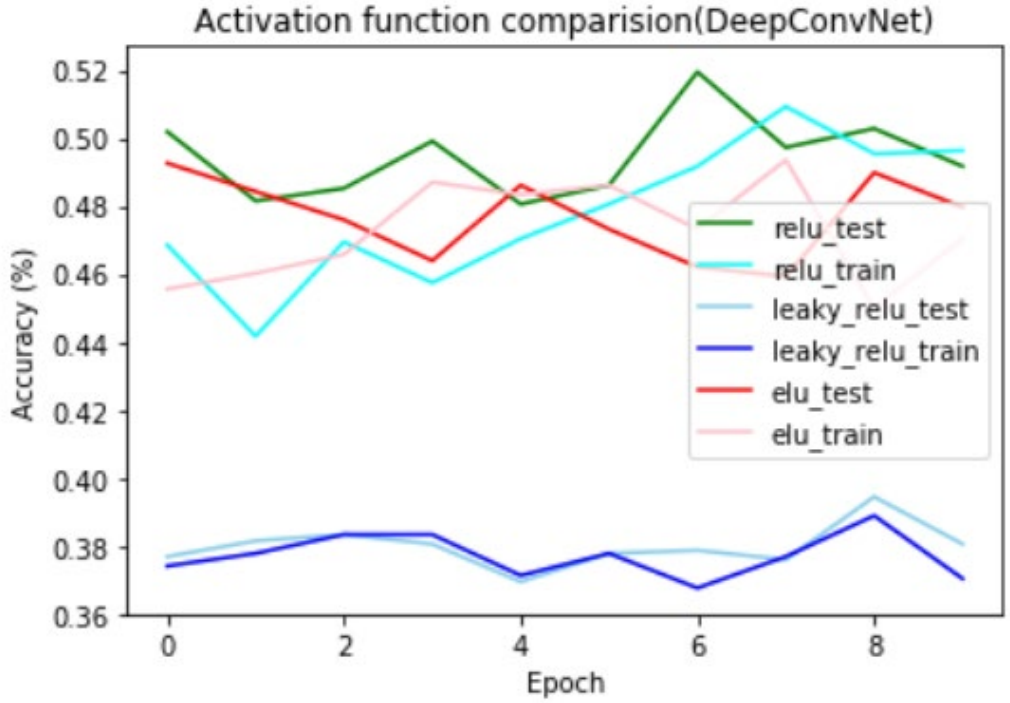
```
ACC of DeepConvNet:  0.4962962962962963
```

B. Comparison figures

i. EEGNet



ii. DeepConvNet



4. Discussion

不同於上一份作業，這份作業要使用 Pytorch，在選定架構的過程遇到很多問題，像是嘗試很久才發現如何使用 Sequential 建立各層 Layer 的設定，還有最後輸出層的承接也調整了很久，backward 的部分，對於 loss 設定上的調整，也花了不少時間，不同 activation function 著實會影響結果的好壞，還有這次 epoch 的數量可能要設定到很高才會有明顯好的結果，起因應該是樣本數較多的問題，整體來說還需要時間來訓練模型，才能達到如預期般良好的效果。