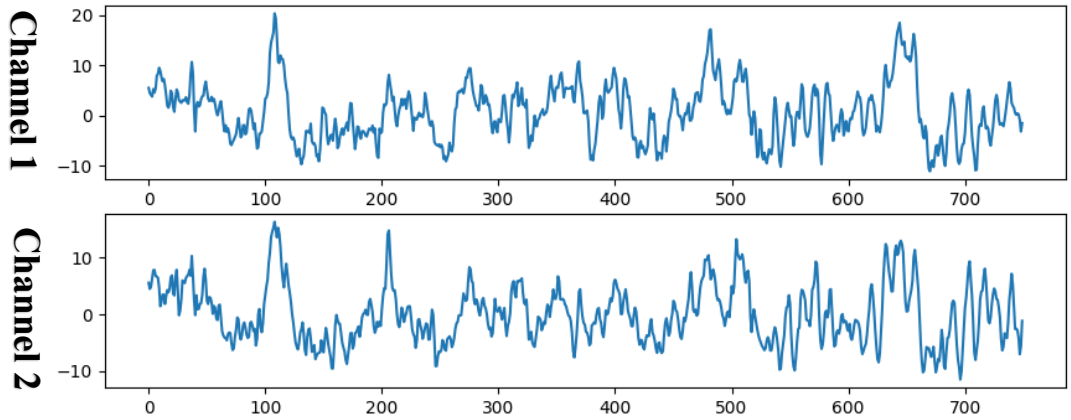
Deep Learning Lab 2

312551089 彭筱竹

1. Introduction

本次報告分別實作了 EEGNet 和 DeepConvNet 兩個神經網路架構，用以對 BCI competition dataset 進行分類，資料集由固定時間區段的雙通道 EEG 訊號組成，如下圖一所示。在實驗中，會針對 EEGNet 和 DeepConvNet 的結果進行比較，並使用 ReLU、Leaky ReLU 和 ELU 三種 activation functions 測試模型的準確率。



▲圖一、BCI competition dataset 第一筆資料

2. Experiment set up

A. The detail of your model

◆ **EEGNet**

EEGNet 架構如下圖二所示，由 Convolution、Depthwise Convolution、Seperable Convolution 三大區塊組成。第一個 Convolution 進行普通的卷積運算，接著 Depthwise Convolution 以 group 參數來實現對各 channel 逐個進行卷積，最後 Seperable Convolution 實現 Depthwise Convolution 與 Pointwise Convolution 的運算，單獨連接每個 feature map 並學習以最佳方式將 feature map 加權組合在一起，以此彌補 Depthwise Convolution 未很好利用不同 channel 在相同空間位置上之資訊的缺點。

```
class EEGNet(nn.Module):
    def __init__(self, activation):
        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,eps=1e-5, momentum=0.1, affine=True, track_running_stats=True)
        )
```

```
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False),
            nn.BatchNorm2d(32, eps=1e-5, 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.seperableConv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False),
            nn.BatchNorm2d(32, eps=1e-5, 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.Linear(736, 2, bias=True)

    def forward(self, input):
        output = self.firstconv(input)
        output = self.depthwiseConv(output)
        output = self.seperableConv(output)
        output = output.view(-1, 736)
        output = self.classify(output)
        return output
```

▲圖二、EEGNet

- **depthwise convolution**

  傳統的卷積運算是在輸入的每個通道上應用相同的 kernel filter，而 depthwise convolution 則是將 kernel filter 分成與輸入通道數量相同的個體，每個 kernel filter 單獨作用於對應的輸入通道，產生單獨的特徵圖。

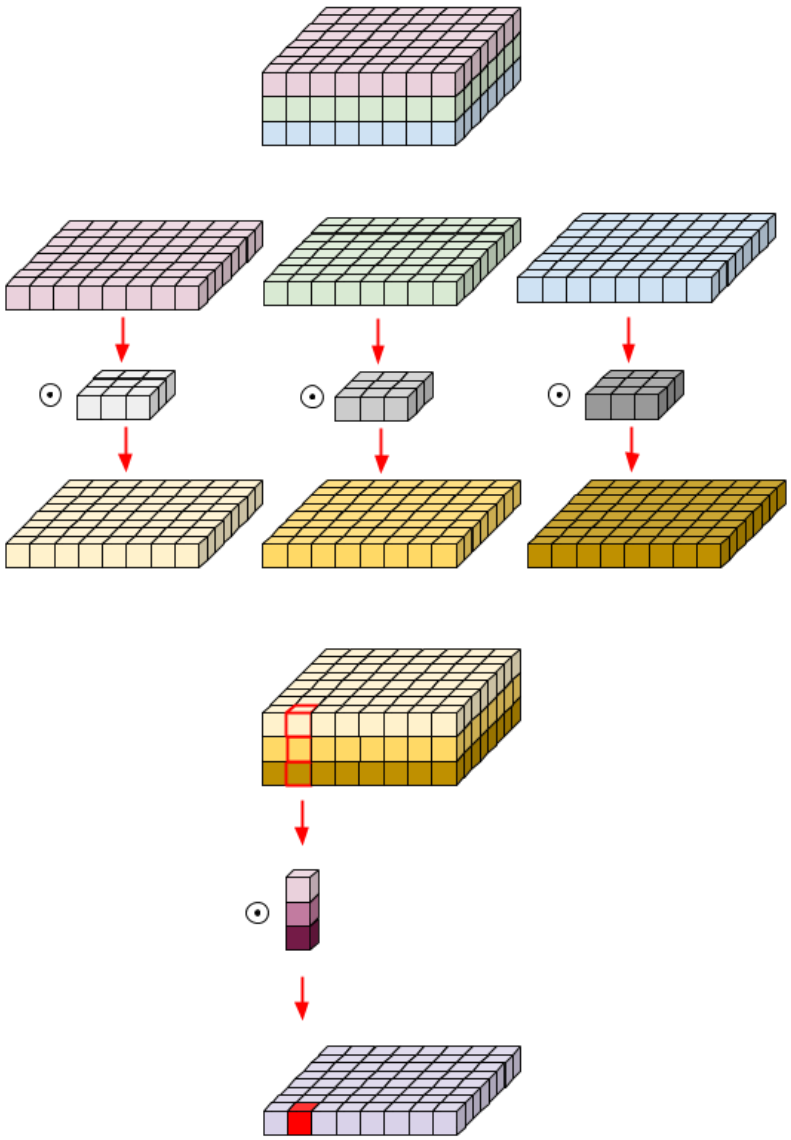
  具體來說，如果輸入資料具有 C 個通道，那麼會有 C 個 kernel filter，每個 kernel filter 只與輸入的一個通道進行卷積操作，意味著每個通道的特徵圖會被單獨地學習相應的特徵，深度卷積的輸出是 C 個單通道的特徵圖。這樣可以大幅減少參數量和計算量，但缺點是信息在通道之間無法交互。

- **pointwise convolution**

  Pointwise convolution 也被稱為 1x1 卷積，它的 kernel size 為 1x1，所以實際上每個輸出的特徵圖像素值只與對應輸入特徵圖在相同位置的單個像素值有關，即將所有通道上相同空間位置的卷積結果加權求和，作為輸出特徵圖在該位置的值。pointwise convolution 實際上是在特徵圖的通道維度上做線性組合，起到跨通道整合信息的作用。

- **seperable convolution**

  Separable convolution 是 depthwise convolution 加上 pointwise convolution 的組合，Depthwise convolution 擷取空間資訊，Pointwise convolution 將 depthwise convolution 產生的特徵圖在通道維度上做組合，聚合通道資訊。由於 1x1 大小的 kernel 大幅降低了參數量和計算量，又可以很好地近似傳統卷積的效果。



◆ **DeepConvNet**

DeepConvNet 為傳統的 CNN 架構，由 Convolution、Batchnormalized、activation function、Pooling、Dropout 組成，最後透過 Fully Connected Layer 輸出最終的預測結果。

```
class DeepConvNet(nn.Module):
    def __init__(self, activation):
        super(DeepConvNet, self).__init__()
        self.conv0 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(25, eps=1e-5, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.conv1 = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(50, eps=1e-5, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(100, eps=1e-5, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.BatchNorm2d(200, eps=1e-5, momentum=0.1),
            activation,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
```

```
    self.classify = nn.Linear(8600, 2, bias=True)

def forward(self, input):
    output = self.conv0(input)
    output = self.conv1(output)
    output = self.conv2(output)
    output = self.conv3(output)
    output = output.view(-1, 8600)
    output = self.classify(output)
    return output
```

▲圖三、DeepConvNet

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

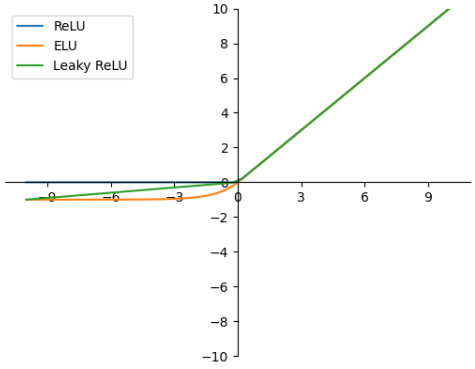
- $\mathrm{ReLU}(x) = max(0, x)$

  ReLU 是最常用的激活函數之一，其輸出為輸入值與 0 的最大值，也就是說當輸入 x<0 時輸出為 0;當 x>=0 時輸出為原值 x。ReLU 的優點在於它簡單且計算高效，同時也解決了梯度消失的問題，然而，ReLU 在負數部分的輸出為 0，可能導致一些神經元死亡。

- $\text{Leaky ReLU}(x)\begin{cases} x, if\ x \geq 0 \\ negative\_slope \times x, otherwise \end{cases}$

Leaky ReLU 是 ReLU 的改進版本，它在負數部分引入一個小的斜率，使得當輸入小於 0 時不再完全為 0，而是乘以一個小的斜率 α，其中 α 是一個接近於 0 的小常數，通常設置為 0.01，從而緩解了 ReLU 的死亡神經元問題。

- $\mathrm{ELU}(x) = max(0, x) + min(0, \alpha * (e^x - 1))$
  ELU 在負數部分引入了一個指數函數，可以產生非零均值的輸出，其中 α 是一個正數常數，控制 ELU 函數在負數部分的輸出接近於 0 的速度。ELU 的優點在於它可以在負數部分保持一個非零的梯度，有助於緩解梯度消失的問題。



▲圖四、activation function

ReLU、Leaky ReLU、ELU 之函數圖如圖四所示。可以看到 Leaky ReLU、ELU 和 ReLU 不同，皆有負的輸出，代表兩者的輸入小於零時，有一個 gradient 存在，而當 ReLU 輸入小於零時，gradient 為零，神經網路的權重將無法更新，Leaky ReLU 和 ELU 的設計避免了 dying ReLU 的情況。

ELU 與 Leaky ReLU 類似，在輸入為負值時曲線相對 Leaky ReLU 較為平滑，zero-centered 的特性能夠加快訓練速度，而曲線左側的飽和性(微分趨近零)則帶來更好的收斂。

3. Experimental results

A. The highest testing accuracy

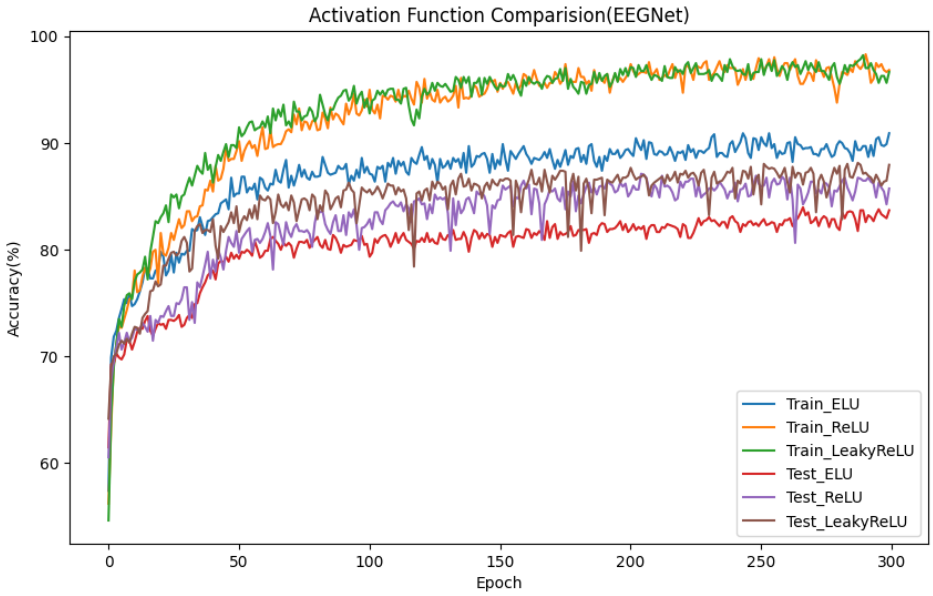
EEGNet	DeepConvNet
Train_ELU Max Accuracy:90.92592592592 Train_ReLU Max Accuracy:98.33333333333333 Train_LeakyReLU Max Accuracy:98.24074074074075 Test_ELU Max Accuracy:83.98148148148148 Test_ReLU Max Accuracy:87.12962962962963 Test_LeakyReLU Max Accuracy:88.14814814814815	Train_ELU Max Accuracy:98.33333333333333 Train_ReLU Max Accuracy:94.81481481481481 Train_LeakyReLU Max Accuracy:94.53703703703704 Test_ELU Max Accuracy:82.5925925925926 Test_ReLU Max Accuracy:85.0925925925926 Test_LeakyReLU Max Accuracy:83.98148148148148

	ReLU	Leaky ReLU	ELU

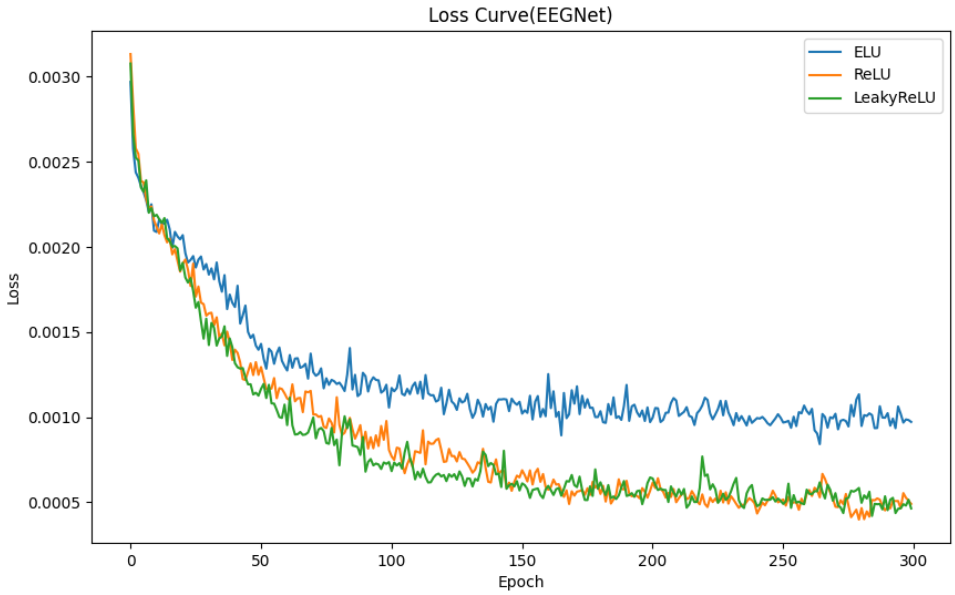
EEGNet	**87.13%**	**88.15**	**83.98**
DeepConvNet	85.09	83.92	82.59

B. Comparison figures

◆ **EEGNet**

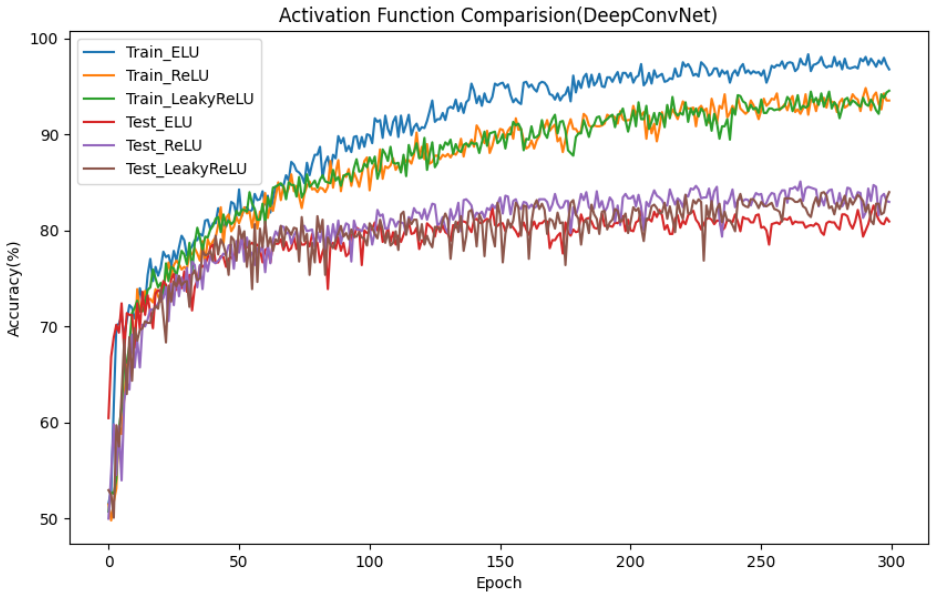


▲圖五、EEGNet accuracy curve with different activation functions

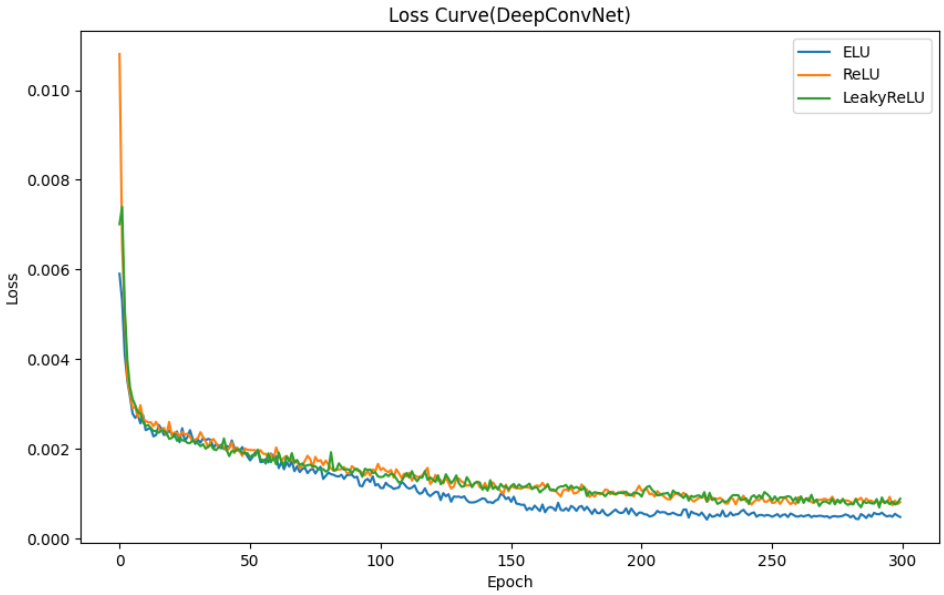


▲圖六、EEGNet loss curve with different activation functions

◆ **DeepConvNet**



▲圖七、DeepConvNet accuracy curve with different activation functions

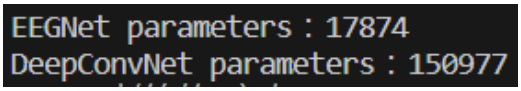


▲圖八、DeepConvNet loss curve with different activation functions

4. Discussion

依據實驗結果，可以觀察到 EEGNet 預測結果的準確率普遍較 DeepConvNet 高。

由於 Depthwise Convolution 一個 kernel 只對一個 channel 進行運算，並非像傳統 CNN 連接至所有的 feature map，因此 EEGNet 的參數量遠小於 DeepConvNet，擁有更高的計算效率，如下圖七所示。



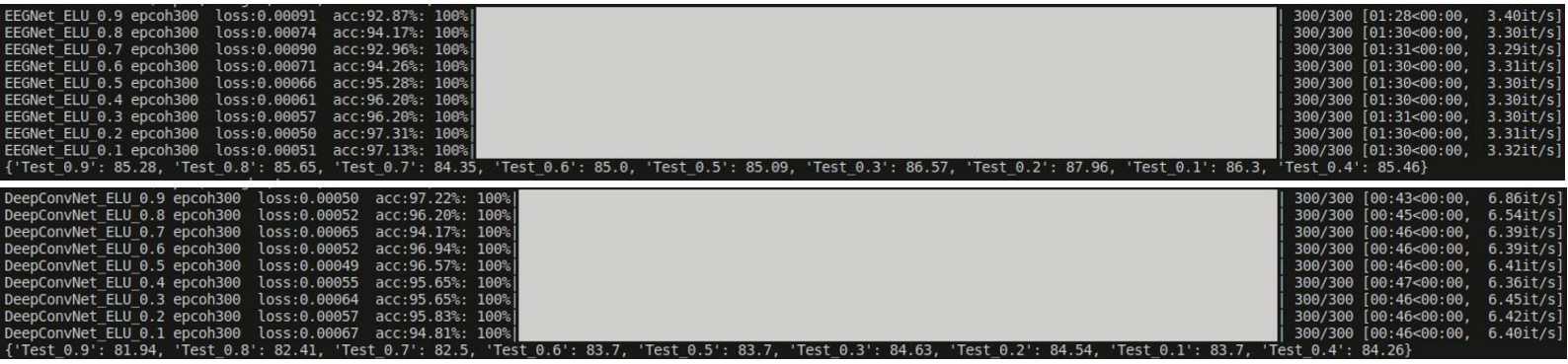
▲圖九、number of parameters

另一方面，本次所使用的 BCI competition dataset 是多維空間的 EEG 訊號，EEGNet 架構的 Depthwise、Seperable Convolution 設計相較單純的 Convolution 能更有效提取時間序列內特定的頻率特徵。

總體而言，在 EEG 上的應用，不論是從準確率還是參數量，EEGNet 皆優於 DeepConvNet。

此外，根據實驗結果，EEGNet 使用 Leaky ReLU 取得了最高的 88.15%準確率，高於 ReLU 和 ELU，這可能是因為 Leaky ReLU 在負值區域保留了一些梯度信息，避免了神經元完全失活從而更有效地捕捉 EEG 信號中的微小變化。ELU 的 83.98%準確率相對較低，可能是因為 ELU 的計算開銷較大，在這個任務上沒有顯著優勢。

而在 DeepConvNet 中，Leaky ReLU 和 ELU 在此模型上表現反而不如 ReLU，ReLU 以 85.09%準確率名列首位，可能受益於其簡單性及避免梯度消失的特性。



從上圖可以看到在訓練階段 EEGNet 和 DeepConvNet 趨勢相反，alpha 越小，EEGNet 效果越好，DeepConvNet 效果變差。

ELU alpha	EEGNet	DeepConvNet
0.9	85.28	81.94
0.8	85.65	82.41
0.7	84.35	82.5
0.6	85	83.7
0.5	85.09	83.7
0.4	85.46	84.26
0.3	86.57	**84.63**

0.2	**87.96**	84.54
0.1	86.3	83.7

不同的 ELU 參數值（alpha 值）對於模型有著不同影響，對於 EEGNet，當 alpha 值為 0.2 時性能達到了最高的 87.96%，而 DeepConvNet 在相同的 alpha 值下性能略有下降，為 84.54%。這表明不同的神經網路對於 ELU 參數的選擇敏感程度不同，需要針對具體的網路結構和任務進行調整。

ELU 參數的微小變化會導致性能的顯著變化，例如當 alpha 值從 0.1 增加到 0.2 時，EEGNet 的性能從 86.3%提高到了 87.96%，

在大多數情況下，EEGNet 的性能略高於 DeepConvNet。

5. Github

https://github.com/pengemma/AIMI/tree/main/LAB2