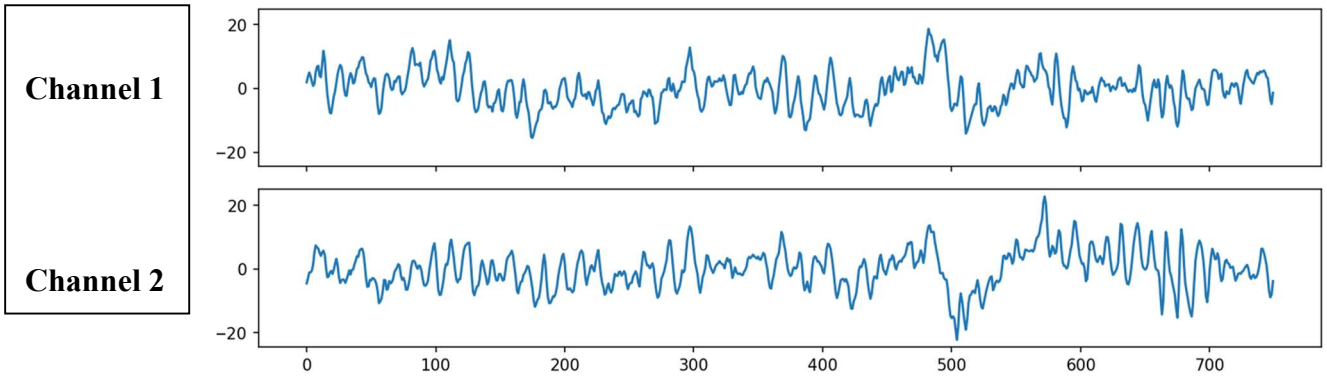
Artificial Intelligence on Medical Imaging Lab 2

314553020 許良亦

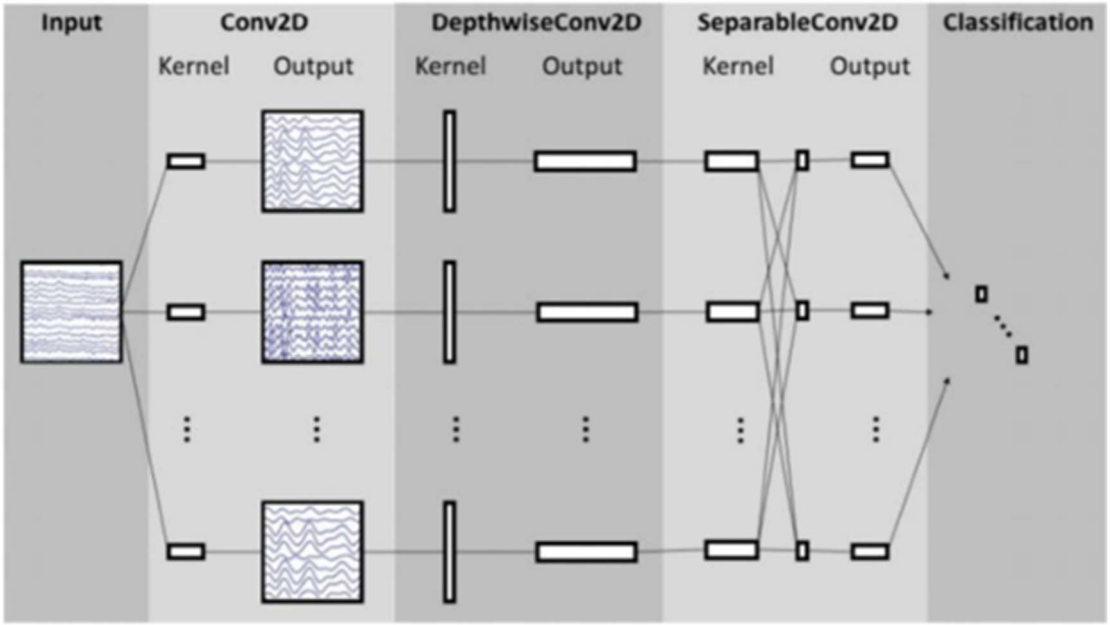
1. Introduction

本次實驗實作了 **EEGNet** 與 **DeepConvNet** 兩種神經網路架構，用於對 **BCI Competition** 所提供的資料集進行腦電訊號分類。該資料集由固定時間區段的 **雙通道 EEG 訊號** 組成，如圖一所示。在實驗設計中，我們分別以 **ReLU**、**Leaky ReLU** 及 **ELU** 三種 activation functions 進行測試，並比較兩種模型在分類準確率上的表現差異。



▲圖一、BCI competition dataset 第三筆資料

A. Introduce EEGNet



▲圖二、EEGNet model 之架構

EEGNet 是一種專為腦電訊號（Electroencephalography, EEG）設計的輕量級深度學習模型，由 Lawhern 等人於 2018 年提出。該模型基於卷積神經網路（Convolutional Neural Network, CNN）架構，能夠有效學習 EEG 資料中的時域與空域特徵，同時維持相對較低的參數量與運算成本。EEGNet 採用 Depthwise Convolution 與 Separable Convolution 的設計，使其在捕捉不同頻段與通道間特徵時更加高效。Depthwise Convolution 用於提取各通道的空間特徵，而 Separable Convolution 則用於進一步壓縮特徵維度並降低參數量。透過這樣的結構，EEGNet 能在小樣本的腦電資料上仍保持良好的泛化能力。由於其輕量化與高效特性，EEGNet 已被廣泛應用於多種腦機介面（Brain–Computer Interface, BCI）任務中，如運動想像分類、事件相關電位（ERP）辨識與情緒分析等。本研究亦採用 EEGNet 作為基礎架構之一，用以比較其在 BCI competition 資料集上的分類效能。

2. Experiment setups and result

A. Highest testing accuracy

EEGNet

```
Train_ELU Max Accuracy:94.16666666666667
Train_ReLU Max Accuracy:98.79629629629629
Train_LeakyReLU Max Accuracy:99.16666666666667
Test_ELU Max Accuracy:85.18518518518519
Test_ReLU Max Accuracy:87.22222222222223
Test_LeakyReLU Max Accuracy:87.77777777777777
```

```
EEGNet — Activation Accuracy Summary
-----------------------------------
Activation   Train Max (%)  Test Max (%)  Best Saved (%)
-----------------------------------------------------
LeakyReLU          99.17         87.78          87.78
ReLU               98.80         87.22          87.22
ELU                94.17         85.19          85.19
```

DeepConvNet

```
Train_ELU Max Accuracy:99.16666666666667
Train_ReLU Max Accuracy:97.03703703703704
Train_LeakyReLU Max Accuracy:97.22222222222223
Test_ELU Max Accuracy:81.94444444444444
Test_ReLU Max Accuracy:84.16666666666667
Test_LeakyReLU Max Accuracy:85.0925925925926
```

```
DeepConvNet — Activation Accuracy Summary
----------------------------------------
Activation   Train Max (%)  Test Max (%)  Best Saved (%)
-----------------------------------------------------
LeakyReLU          97.22         85.09          85.09
ReLU               97.04         84.17          84.17
ELU                99.17         81.94          81.94
```

▲表一、testing accuracy results

Hyper Parameters

EEGNet

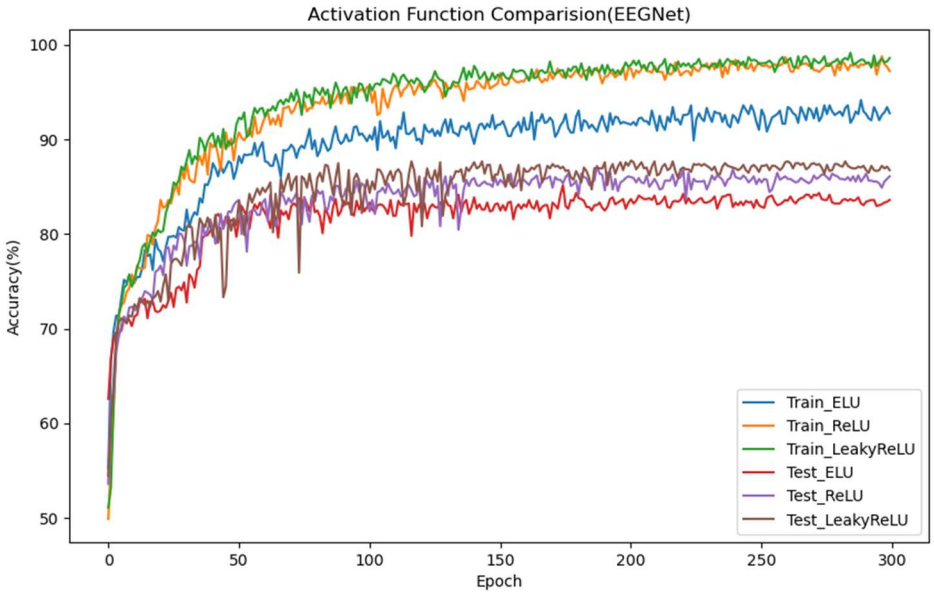
- Batch size :256
- Learning rate :1.824e-3
- Optimizer: Adam
- Weight_decay:1e-2

DeepConvNet

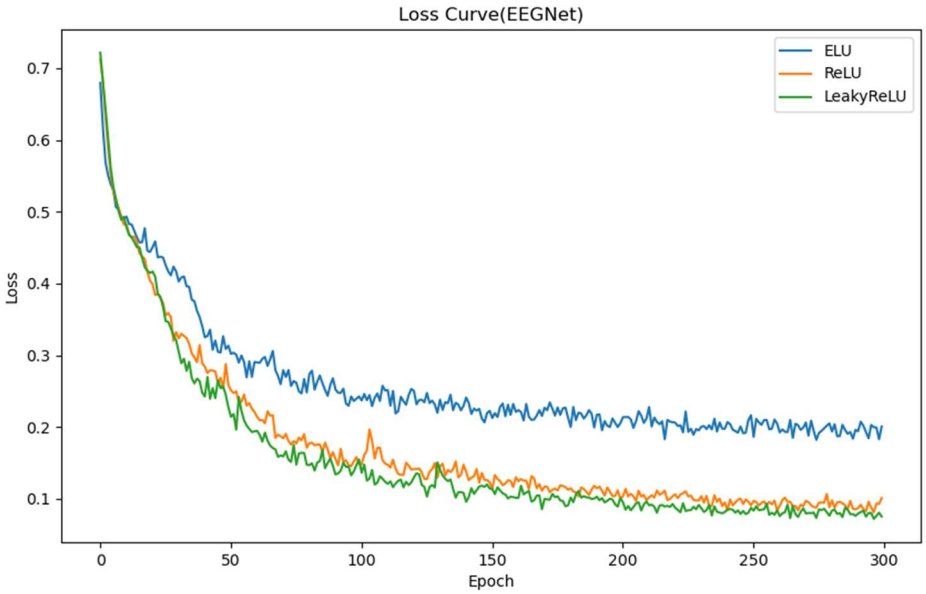
- Batch size :256
- Learning rate :2e-3
- Optimizer: Adam
- Weight_decay:1e-2

B. Train/ Test loss and accuracy curve

■ **EEGNet**



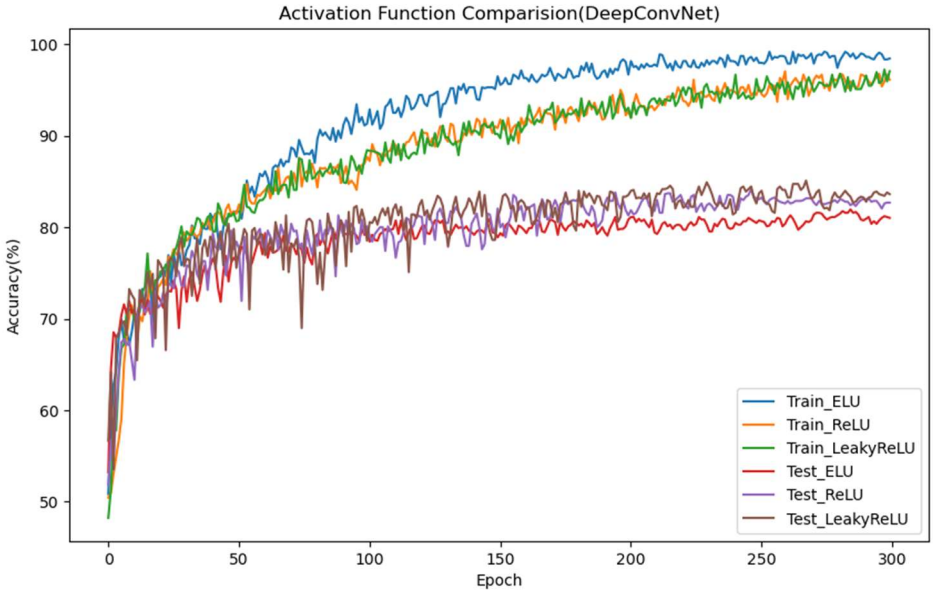
➤圖三、EEGNet accuracy curve with different activation functions



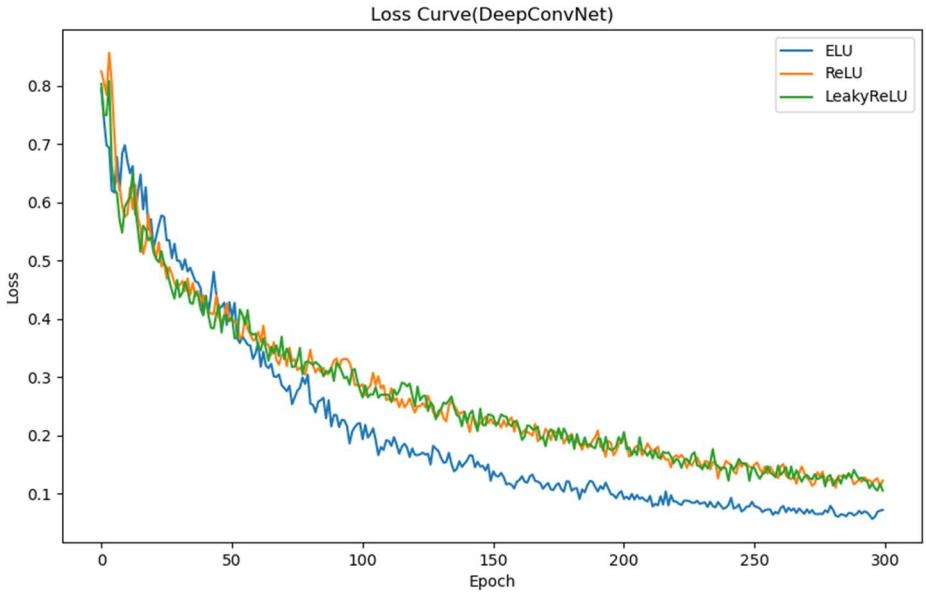
➤圖四、EEGNet loss curve with different activation functions

➤表二、ELU alpha 0.1 ~ 0.9 results

■ **DeepConvNet**



➤圖五、DeepConvNet accuracy curve with different activation functions

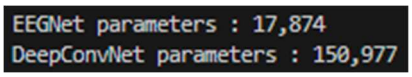


➤圖六、DeepConvNet loss curve with different activation functions

3. Discussion

A. Discuss your findings or share anything you want to share

根據實驗結果可觀察到，EEGNet 在測試集上的準確率普遍高於 DeepConvNet，顯示其在處理 EEG 信號時具有較佳的泛化能力。我認為是因為 EEGNet 採用了 Depthwise 與 Separable Convolution 結構，使每個卷積核僅針對單一通道進行特徵擷取，進而大幅減少參數量並提高計算效率。相較之下，DeepConvNet 為傳統的多層卷積設計，每層卷積核與所有輸入特徵圖相連，因此雖能提取較多高階特徵，但參數量與運算成本顯著增加（EEGNet 約 17k 參數，DeepConvNet 約 151k 參數）。



▲圖七、model parameters

在 activation function 的比較中，LeakyReLU 在兩個模型中皆表現最佳，其測試準確率於 EEGNet 為 **87.78%**、DeepConvNet 為 **85.09%**。我認為是因為 LeakyReLU 允許負區域保留小梯度，可避免 ReLU 在負輸入下出現梯度消失問題，因此能保持更穩定的學習與更高的泛化能力。而 ELU 雖在訓練集上收斂較快，但測試準確率略低，可能為 ELU 開銷較大，在這個任務比較沒有優勢。

從 Loss 曲線可見，兩種模型皆能隨訓練穩定下降，其中 EEGNet 的 loss 變化更平滑，顯示其網路結構在正則化與穩定性方面較佳。結論，EEGNet 以較少的參數量達成與甚至超越 DeepConvNet 的準確率，證實其為一種高效且適用於腦電訊號分類的輕量化模型。

EEGNet	DeepConvNet
EEGNet_ELU_a0.1_87.31% -> 87.31% EEGNet_ELU_a0.2_86.85% -> 86.85% EEGNet_ELU_a0.3_86.3% -> 86.30% EEGNet_ELU_a0.4_85.0% -> 85.00% EEGNet_ELU_a0.5_85.46% -> 85.46% EEGNet_ELU_a0.6_86.02% -> 86.02% EEGNet_ELU_a0.7_82.78% -> 82.78% EEGNet_ELU_a0.8_83.8% -> 83.80% EEGNet_ELU_a0.9_85.65% -> 85.65%	DeepConvNet_ELU_a0.1_84.81% -> 84.81% DeepConvNet_ELU_a0.2_84.63% -> 84.63% DeepConvNet_ELU_a0.3_83.98% -> 83.98% DeepConvNet_ELU_a0.4_84.17% -> 84.17% DeepConvNet_ELU_a0.5_83.24% -> 83.24% DeepConvNet_ELU_a0.6_83.8% -> 83.80% DeepConvNet_ELU_a0.7_83.24% -> 83.24% DeepConvNet_ELU_a0.8_82.96% -> 82.96% DeepConvNet_ELU_a0.9_82.31% -> 82.31%

EEGNet — ELU Alpha Sweep Summary

alpha	Train Max (%)	Test Max (%)	Best Saved (%)
0.1	98.70	87.31	87.31
0.2	98.24	86.85	86.85
0.3	97.87	86.30	86.30
0.6	96.39	86.02	86.02
0.9	93.24	85.65	85.65
0.5	97.22	85.46	85.46
0.4	97.50	85.00	85.00
0.8	95.93	83.80	83.80
0.7	95.46	82.78	82.78

DeepConvNet — ELU Alpha Sweep Summary

alpha	Train Max (%)	Test Max (%)	Best Saved (%)
0.1	98.33	84.81	84.81
0.2	98.43	84.63	84.63
0.4	99.26	84.17	84.17
0.3	99.07	83.98	83.98
0.6	98.80	83.80	83.80
0.7	99.17	83.24	83.24
0.5	99.26	83.24	83.24
0.8	99.17	82.96	82.96
0.9	99.35	82.31	82.31

不同的 ELU 參數值(α 值)對模型的表現具有明顯影響。以實驗結果觀察，EEGNet 在 α=0.1 時達到最高準確率 87.31%，而 DeepConvNet 在相同設定下的準確率為 84.81%，略低於 EEGNet。這顯示不同的網路架構對激活函數參數的敏感程度並不相同，需根據具體的模型結構與任務特性進行調整。此外，ELU 參數的微小變化也可能造成顯著的性能差異。例如，當 α 從 0.1 增加至 0.2 時，EEGNet 的準確率由 87.31% 降至 86.85%，顯示適度的 α 值能更有效地平衡負區域的非線性輸出與梯度流動，進而提升學習穩定性與泛化能力。整個實驗來看，EEGNet 的表現普遍優於 DeepConvNet，並且對 ELU 參數的變化展現出更穩定的反應。

```
# --- Warmup + Cosine ---
warmup_epochs = min(warmup_epochs, max(0, epochs-1))
warmup = LinearLR(optimizer, start_factor=0.1, total_iters=warmup_epochs)
T_cos = max(1, epochs - warmup_epochs)
cosine = CosineAnnealingLR(optimizer, T_max=T_cos, eta_min=lr * 0.1)
scheduler = SequentialLR(optimizer, schedulers=[warmup, cosine], milestones=[warmup_epochs])
```

▲圖八、Cosine lr + Warmup

在本次實驗中，為了使模型訓練更加穩定並提升收斂效率，加入了 Warmup 結合 Cosine Annealing 學習率調整策略。其中，Warmup 採用線性增長方式，於前幾個 epoch 將學習率由初始值的 0.1 緩慢提升至設定的最終學習率，以避免訓練初期梯度不穩定導致的收斂問題；之後則採用 CosineAnnealingLR 使學習率隨訓練進程逐漸下降。讓模型在早期階段加速模型收斂，而在後期提供較小的學習率以防止過度擬合，使模型在穩定性與最終性能之間達到平衡。

B. Explain how you implement the model

■ **EEGNet**

```python
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.separableConv = 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.flatten  = nn.Flatten()
        self.classify = nn.LazyLinear(2, bias=True)

    def forward(self, input):
        output = self.firstconv(input)
        output = self.depthwiseConv(output)
        output = self.separableConv(output)
        output = self.flatten(output)
        output = self.classify(output)
        return output
```

▲圖八、EEGNet model

在本次實作中使用 PyTorch 建立了 EEGNet 類別，輸入資料張量形狀為((N,1,C,T))，其中 N 為批次大小、C 為 EEG 通道數、T 為時間長度。model 共包含三個主要卷積區塊與一個分類層。第一層 Temporal Convolution 採用 Conv2d(1→16, kernel=(1,51))，僅沿時間軸進行卷積以擷取時間特徵，並接上 BatchNorm2d(16)。第二層 Depthwise Convolution 使用 Conv2d(16→32, kernel=(2,1), groups=16)，讓每個輸入通道以獨立濾波器學習空間特徵，再接上 BatchNorm、activation、AvgPool 與 Dropout。第三層 Separable Convolution 以 Conv2d(32→32, kernel=(1,15)) 進一步抽取時間資訊，並同樣接續 BatchNorm、activation、平均池化與 Dropout。最後透過 Flatten() 展平特徵，並以 nn.LazyLinear(2) 輸出二分類結果，其中 LazyLinear 會自動推斷輸入維度，提升模型的靈活性。

- **DeepConvNet**

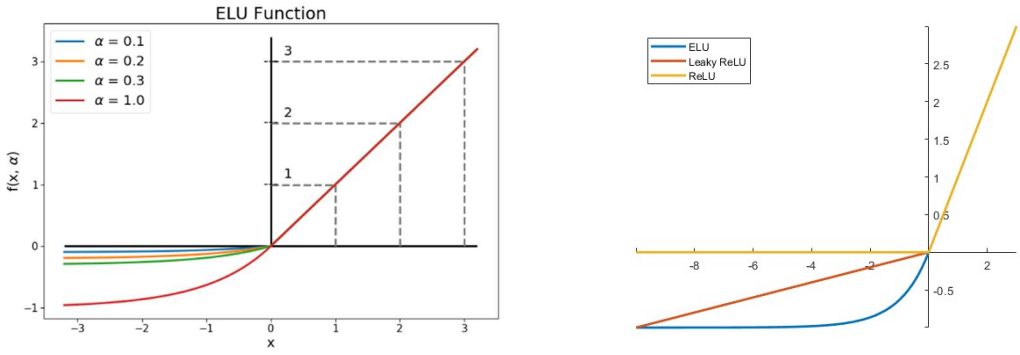
```python
class DeepConvNet(nn.Module):
    def __init__(self, activation):
        super(DeepConvNet, self).__init__()
        self.conv0 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=(0, 0), bias=True),
            nn.Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 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.flatten  = nn.Flatten()
        self.classify = nn.LazyLinear(2, bias=True)

    def forward(self, input):
        output = self.conv0(input)
        output = self.conv1(output)
        output = self.conv2(output)
        output = self.conv3(output)
        output = self.flatten(output)
        output = self.classify(output)
        return output
```

▲圖九、DeepConvNet model

在本次實作中使用 PyTorch 建立了 DeepConvNet，輸入張量形狀為((N,1,C,T))。模型依序包含四個卷積區塊與一個分類層：conv0 先以 Conv2d(1→25, kernel=(1,5)) 做時間卷積，再接 Conv2d(25→25, kernel=(2,1)) 擷取通道資訊，後接 BatchNorm2d(25)、指定的 activation function（ReLU/LeakyReLU/ELU）、MaxPool2d(kernel=(1,2)) 與 Dropout(0.5)；conv1、conv2、conv3 分別以 Conv2d(25→50), Conv2d(50→100), Conv2d(100→200)（皆為 kernel=(1,5)）逐層加深特徵抽取，每層後均接 BatchNorm2d、相同 activation、MaxPool2d(1,2) 與 Dropout(0.5) 以下採樣並正則化。最後使用 Flatten() 展平成一維，透過 nn.LazyLinear(2) 產生二分類 logits（LazyLinear 會在首次 forward

自動推斷輸入維度）。

C. Explain the ELU function



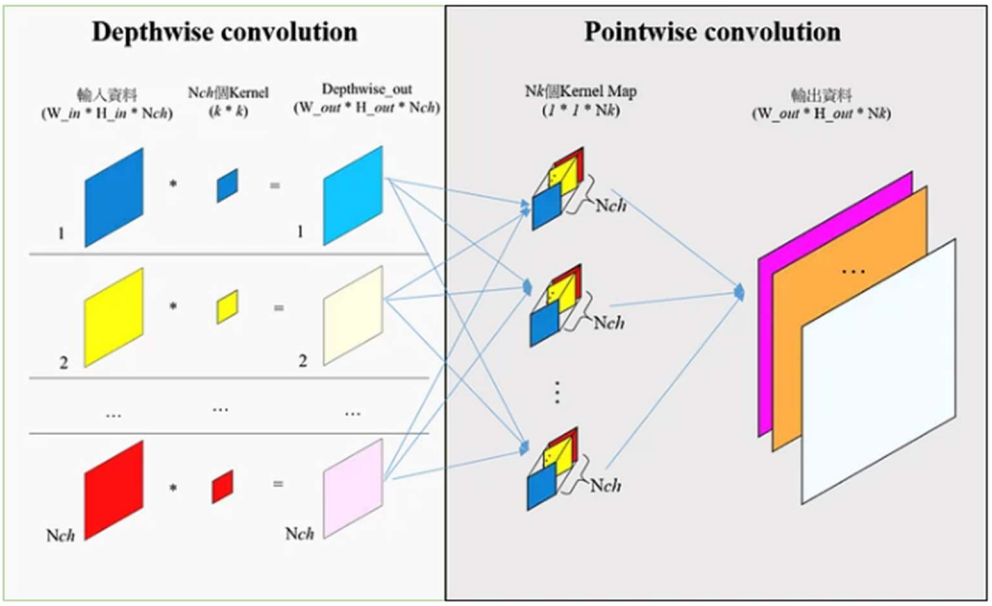
▲圖十、ELU and ReLU and Leaky ReLU

ELU（Exponential Linear Unit） 是一種改良型的 activation function，相較於傳統的 ReLU，ELU 在輸入為負值時不會直接輸出 0，而是使用指數函數的形式，使輸出平滑地接近一個負常數。其數學定義如下：

$$f(x) = \begin{cases} x, & x > 0 \\ \propto (e^{x} - 1), & x \leq 0 \end{cases}$$

其中 α 為常數（一般取 1）。對正值輸入時，ELU 與 ReLU 相同；對負值輸入時，輸出會逐漸趨近於 $-\alpha$，因此不會出現 ReLU 的神經元死亡問題，同時也能使輸出的平均值更接近零，減少梯度消失並加速收斂。

D. Explain what is the depthwise convolution and separable convolution



▲圖 、Separable Convolution 之整體架構

- 輸入資料: W_in * H_in * Nch (輸入圖的寬*高*輸入 channel 數)
- Kernel Map: k * k * Nk (Kernel map 寬*高*kernel 數，kernel 寬高假設一樣)
- Kernel Map：k * k * Nk（Kernel Map 寬*高*內核數 ，kernel 寬高假設一樣）
- 輸出資料: W_out * H_out * Nk (輸出圖的寬*高*輸出 channel 數)

針對輸入資料的每一個 Channel 都建立一個 k*k 的 Kernel，然後每一個 Channel 針對對應的 Kernel 都各自做 convolution，當輸入資料的每個 channel 做完 depthwise convolution 後，針對每個點的所有 channel 做 pointwise convolution。

計算量比較:

$$\frac{W_{in} \cdot H_{in} \cdot N_{ch} \cdot k^2 + N_{ch} \cdot N_k \cdot W_{in} \cdot H_{in}}{W_{in} \cdot H_{in} \cdot N_{ch} \cdot N_k \cdot k^2} = \frac{1}{N_k} + \frac{1}{k^2}$$

分子為 Depthwise separable convolution 計算量，分母為一般卷積計算量，由此公式看出，當 kernel map 越大及數量越多，Depthwise separable convolution 可以節省越多計算量。

■ Depthwise convolution

傳統的卷積運算會在所有輸入通道上同時應用同一組卷積核，而 Depthwise Convolution 則將卷積核拆分成與輸入通道數相同的個體，使每個卷積核只作用於對應的一個輸入通道。若輸入資料具有 C 個通道，則會使用 C 個單通道的卷積核，分別對各通道進行卷積，從而產生 C 張輸出特徵圖。這種方法能大幅降低參數量與計算量，因為不再在通道之間共享權重。然而，其缺點是各通道間缺乏信息交互，僅能獨立學習每個通道的空間特徵。

■ Pointwise convolution

Pointwise Convolution 又稱為 1×1 Convolution，即卷積核的大小為 1×1。此運算會將所有通道上相同空間位置的像素值做加權求和，以產生新的輸出特徵圖。也就是說，Pointwise Convolution 在空間上不改變特徵圖的尺寸，而是在通道維度上進行線性組合，實現跨通道的信息整合。

■ Separable convolution

Separable Convolution（可分離卷積）是以上兩種操作的組合，先進行 Depthwise Convolution 以提取空間特徵，再接著 Pointwise Convolution 將不同通道的特徵進行整合。這樣的設計能在大幅減少參數量與計算成本的同時，仍保留傳統卷積的特徵提取能力，因此被廣泛應用於輕量化網路架構中（例如 EEGNet、MobileNet 等）。

4. Github Link

https://github.com/Ianuyu/AIMI/tree/main