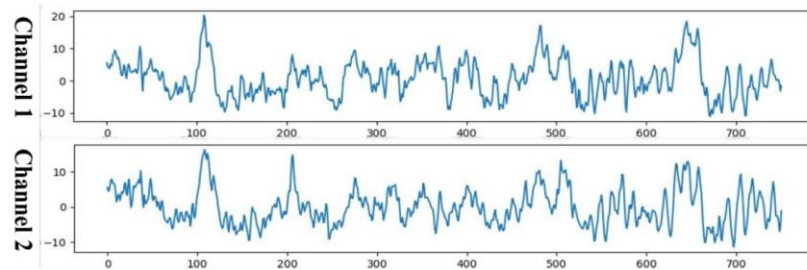


## Lab2: EEG classification

系級:智能系統 學號:312581006 姓名:張宸瑋

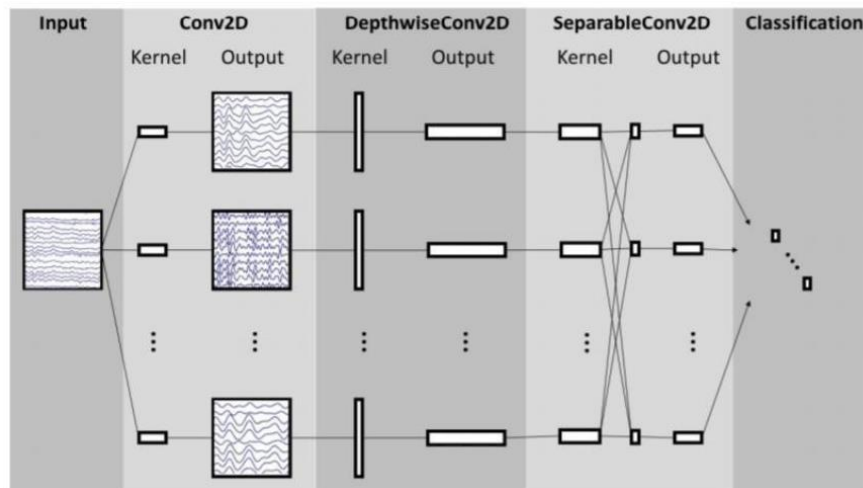
### 1. Introduction

這次 lab 要實作出 EGGNet 以及 DeepConvNet 來分類 EEG 訊號，並且在這兩種不同結構的網路中，使用不同的激活函數，並且展示兩種結構的模型在不同的激勵函數下的最高準確率，並且為了更清楚的看到訓練中準確率變化的趨勢，我們必須可視化在訓練階段以及測試階段，每個 epoch 的準確率曲線。



訓練資料為 BCI dataset

這兩種不同架構的網路會根據下面兩張架構圖分別來做實作



EEGNet 模型架構

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 模型架構

## 2. Experiment setups

### A. The detail of your model

下面兩張圖是我在程式中實現的網路架構

```
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): None
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (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): None
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

EEGNet 實現細節

```

DeepConvNet(
  (layers): ModuleList(
    (0): Sequential(
      (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1))
      (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): None
      (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
      (4): Dropout(p=0.5, inplace=False)
    )
    (1): Sequential(
      (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1))
      (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): None
      (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
      (4): Dropout(p=0.5, inplace=False)
    )
    (2): Sequential(
      (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1))
      (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): None
      (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
      (4): Dropout(p=0.5, inplace=False)
    )
  )
  (firstconv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): None
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)

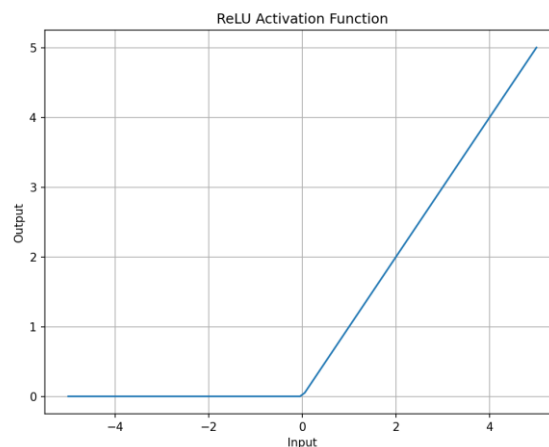
```

DeepConvNet 實現細節

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

ReLU 函數：

ReLU 是一個簡單且廣泛使用的激活函數。它將所有負數的輸入值設置為 0，而保留所有非負數的輸入值不變。數學表示為： $f(x) = \max(0, x)$ 。ReLU 的優點是計算高效，且能夠解決梯度消失的問題。然而，它存在一個稱為"神經元死亡"的問題，即某些神經元可能在訓練過程中永遠不會被激活，導致其梯度始終為 0，無法更新權重。

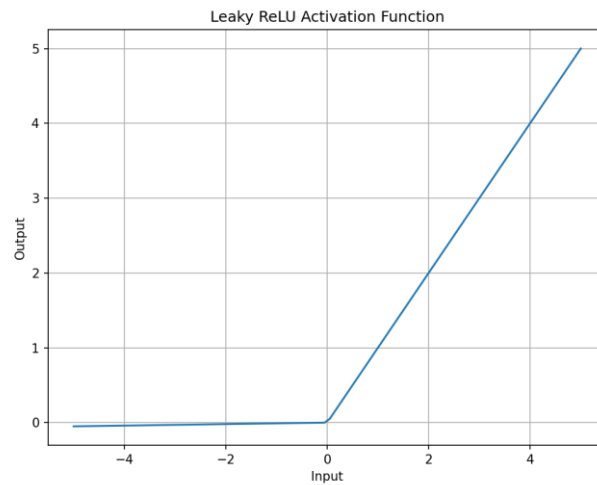


ReLU Function 示意圖

Leaky ReLU 函數：

為了解決 ReLU 的"神經元死亡"問題，Leaky ReLU 被提出。Leaky ReLU 在輸入值為負時引入一個小的斜率，使得當輸入值為負時，其梯度不再為 0。數學表示為： $f(x) = \max(ax, x)$ ，其中  $a$  是一個小

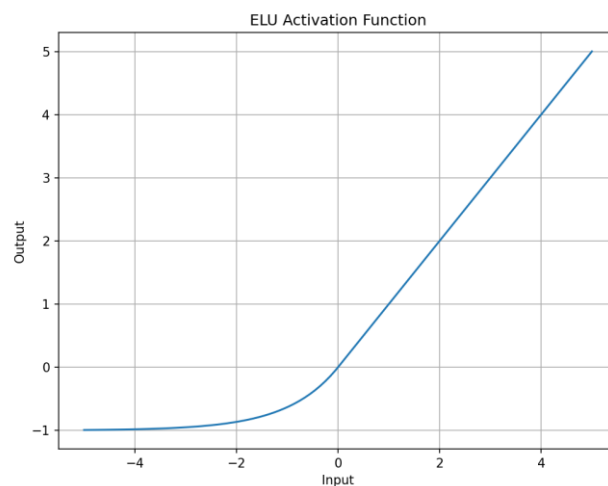
的正數，通常很小，比如 0.01。Leaky ReLU 在 ReLU 的基礎上增加了一個超參數  $\alpha$ ，可以部分解決"神經元死亡"問題。



Leaky ReLU Function 示意圖

ELU 函數：

ELU 是另一種解決"神經元死亡"問題的激活函數。ELU 在輸入值為負時引入了一個負指數項，使得當輸入值為負時，其梯度不僅不會為 0，還會保持平滑。數學表示為： $f(x) = x$  ( $x \geq 0$ )； $f(x) = \alpha(e^x - 1)$  ( $x < 0$ )，其中  $\alpha$  是一個可選的超參數，控制輸入值為負時的曲率。ELU 的主要優點是它能夠解決"神經元死亡"問題，並且在一些情況下可能比 ReLU 和 Leaky ReLU 表現更好，但計算成本較高。



ELU Function 示意圖

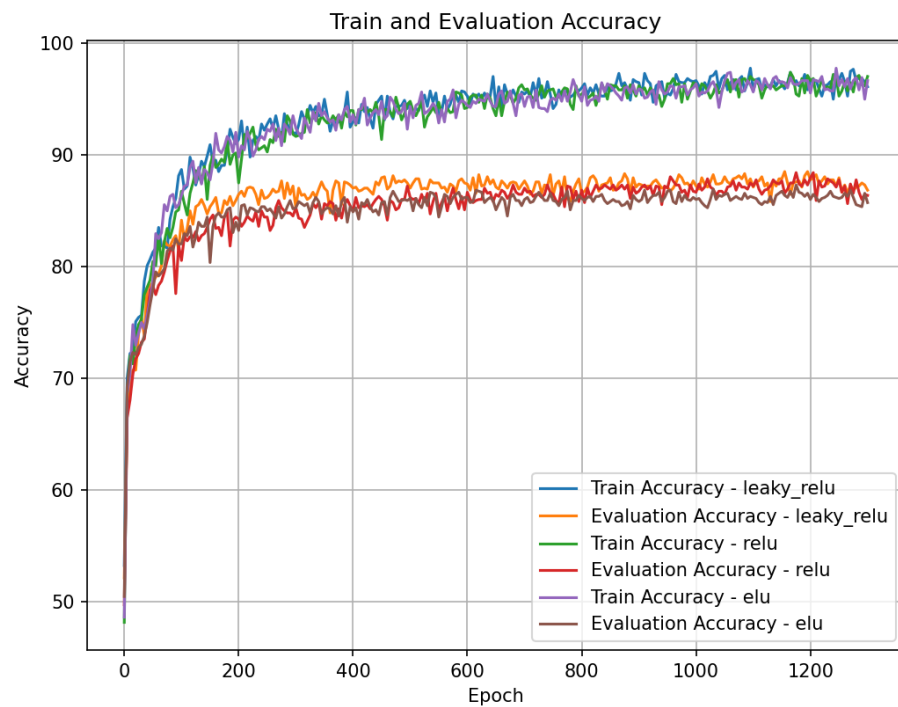
### 3. Experimental results

#### A. The highest testing accuracy

```
The performance of the DeepConvNet model with different activation functions is as follows:  
●leaky_relu Activation  
Accuracy: 85.74%  
●relu Activation  
Accuracy: 85.19%  
●elu Activation  
Accuracy: 85.83%  
The performance of the EGGNet model with different activation functions is as follows:  
●leaky_relu Activation  
Accuracy: 88.70%  
●relu Activation  
Accuracy: 88.61%  
●elu Activation  
Accuracy: 87.31%
```

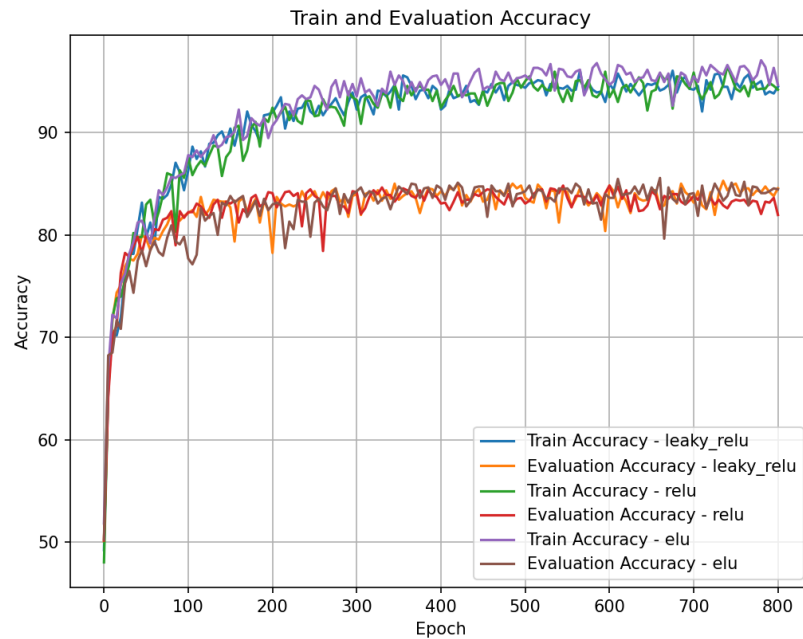
#### B. Comparison figures

EGGNet:



設定参数: epoch:1300, learning rate:0.001, batch size:256,  
Optimizer: Adam, Loss function: torch.nn.BCEWithLogitsLoss()

## DeepConvNet:



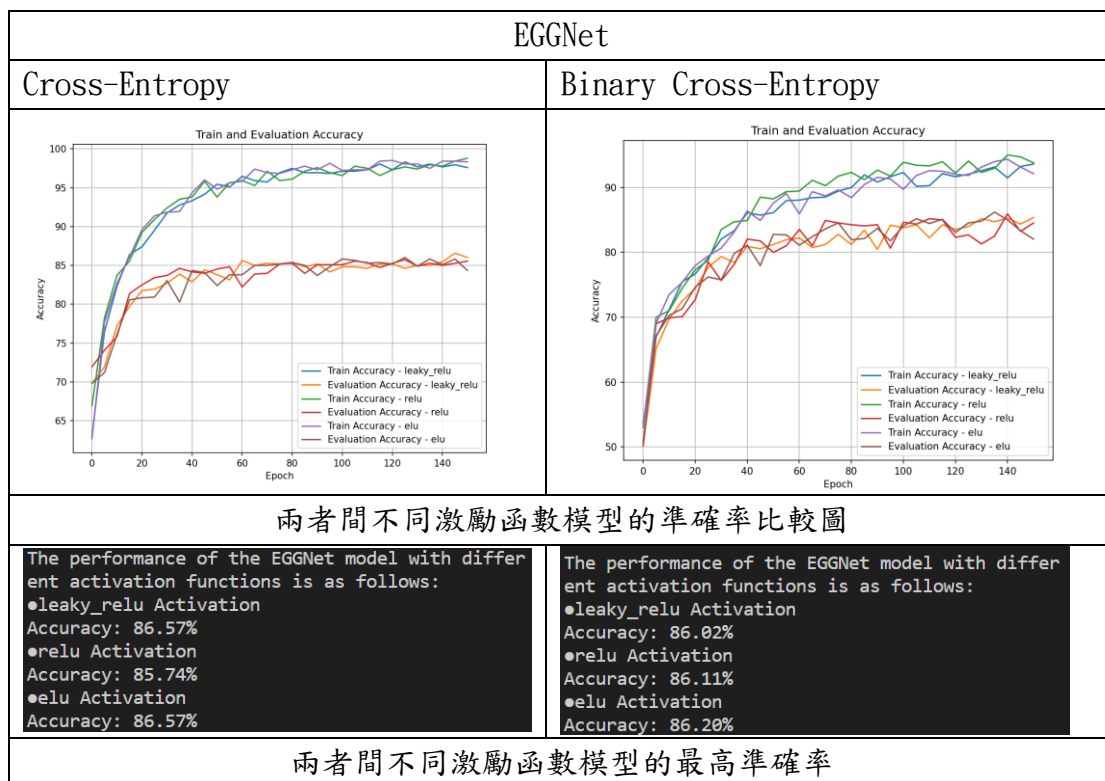
設定參數: epoch: 800, learning rate: 0.001, batch size: 256

Optimizer: Adam, Loss function: torch.nn.BCEWithLogitsLoss()

## 4. Discussion

### 1. 調整 Loss Function

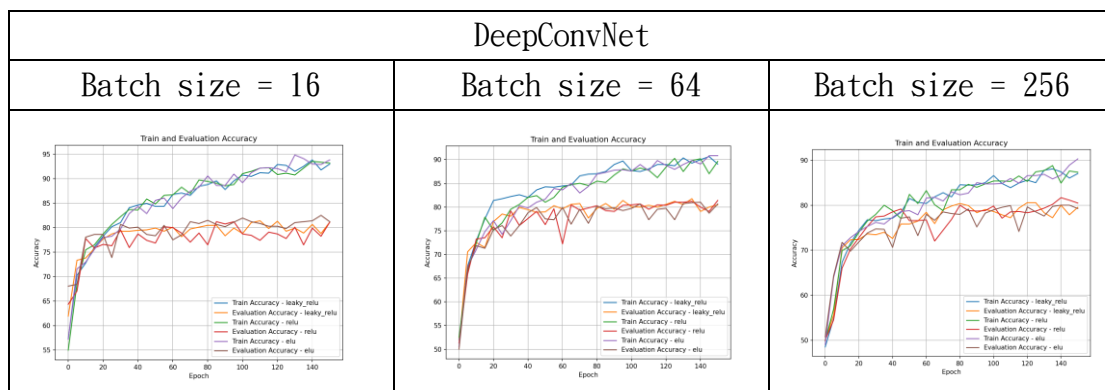
DeepConvNet	
Cross-Entropy	Binary Cross-Entropy
<p>Train and Evaluation Accuracy</p>	<p>Train and Evaluation Accuracy</p>
兩者間不同激勵函數模型的準確率比較圖	
<p>The performance of the DeepConvNet model with different activation functions is as follows:</p> <ul style="list-style-type: none"> <li>leaky_relu Activation Accuracy: 81.30%</li> <li>relu Activation Accuracy: 81.85%</li> <li>elu Activation Accuracy: 81.94%</li> </ul>	<p>The performance of the DeepConvNet model with different activation functions is as follows:</p> <ul style="list-style-type: none"> <li>leaky_relu Activation Accuracy: 83.98%</li> <li>relu Activation Accuracy: 83.33%</li> <li>elu Activation Accuracy: 83.98%</li> </ul>
兩者間不同激勵函數模型的最高準確率	



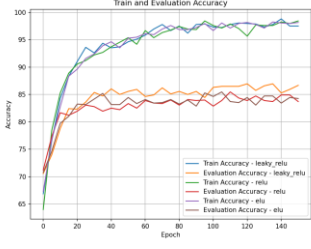
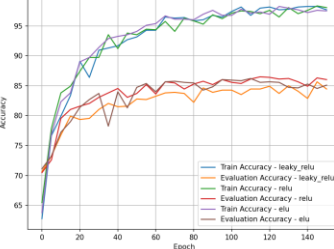
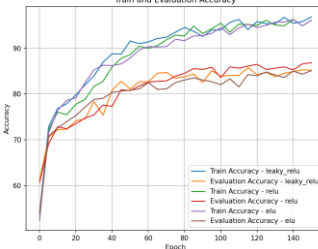
討論：

這裡我將損失函數做了更換，將原本的 Cross-Entropy 更改為 Binary Cross-Entropy，會有這個想法是因為網路的輸出僅僅是 1 跟 0，因此相當於我可以把它看成一個二分類問題，因此我認為 Binary Cross-Entropy 在做法上比較直觀，而這裡我們可以觀察到，再像 DeepConvNet 這樣複雜的網路中，Binary Cross-Entropy 的表現是更好的，我認為是 Binary Cross-Entropy 只需要計算兩個類別之間的交叉熵，而不需要對所有類別進行計算，因此在計算效率上更高。尤其是當網路比較複雜。這裡要注意的是，因為改成二分類問題，所以在 `nn.Linear()` 的 `out_features` 是要更改成 1 的，並且也要對 labels 的 shape 做轉換才行。

## 2. 調整 batch size



不同 batch size 的準確率比較圖		
<ul style="list-style-type: none"> <li>●leaky_relu Activation Accuracy: 82.04%</li> <li>●relu Activation Accuracy: 81.20%</li> <li>●elu Activation Accuracy: 82.50%</li> </ul>	<ul style="list-style-type: none"> <li>●leaky_relu Activation Accuracy: 81.76%</li> <li>●relu Activation Accuracy: 81.94%</li> <li>●elu Activation Accuracy: 81.30%</li> </ul>	<ul style="list-style-type: none"> <li>●leaky_relu Activation Accuracy: 80.83%</li> <li>●relu Activation Accuracy: 81.67%</li> <li>●elu Activation Accuracy: 81.11%</li> </ul>
不同 batch size 的最高準確率		

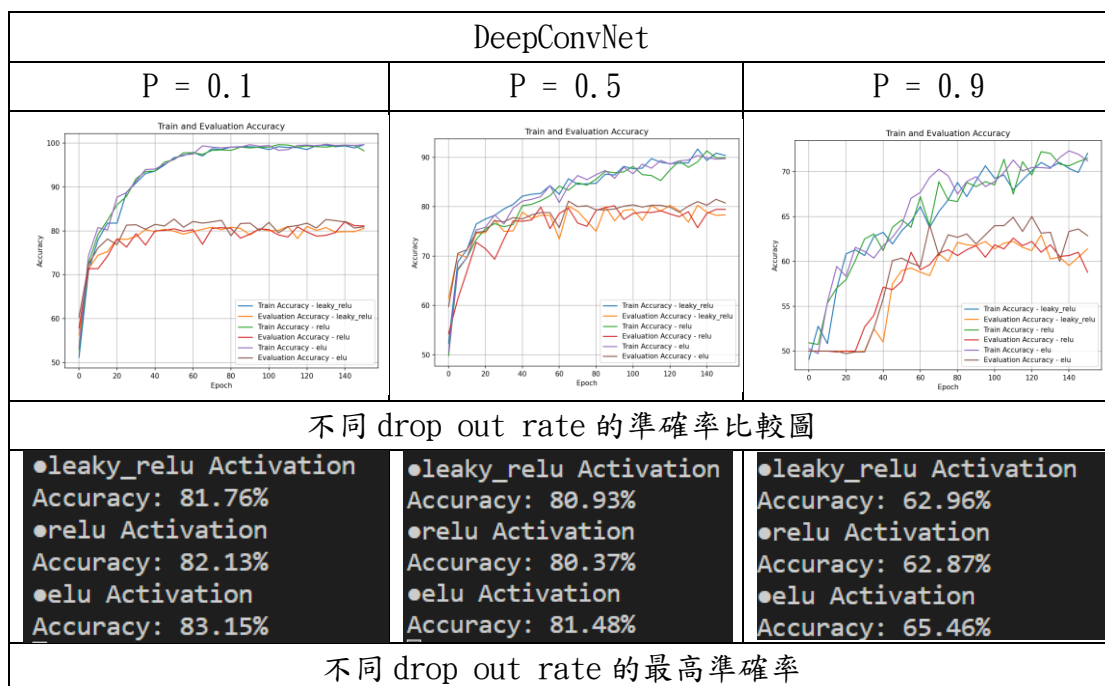
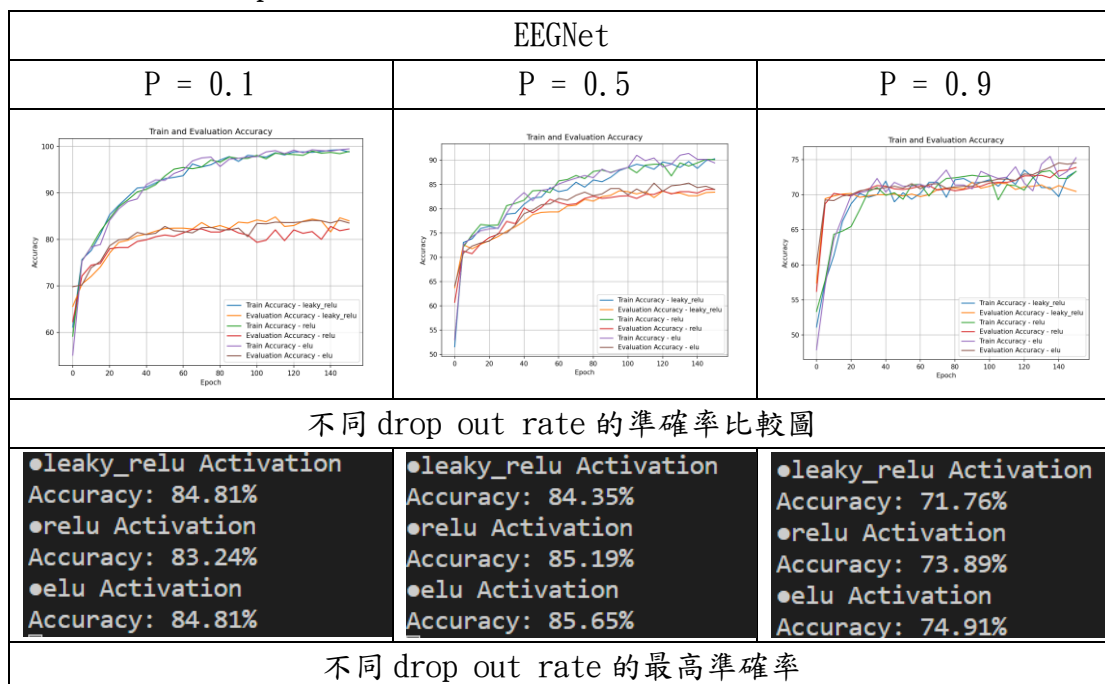
EGGNet		
Batch size = 16	Batch size = 64	Batch size = 256
		
不同 batch size 的準確率比較圖		
<ul style="list-style-type: none"> <li>●leaky_relu Activation Accuracy: 87.59%</li> <li>●relu Activation Accuracy: 85.46%</li> <li>●elu Activation Accuracy: 85.46%</li> </ul>	<ul style="list-style-type: none"> <li>●leaky_relu Activation Accuracy: 85.93%</li> <li>●relu Activation Accuracy: 86.76%</li> <li>●elu Activation Accuracy: 86.48%</li> </ul>	<ul style="list-style-type: none"> <li>●leaky_relu Activation Accuracy: 85.74%</li> <li>●relu Activation Accuracy: 87.31%</li> <li>●elu Activation Accuracy: 85.09%</li> </ul>
不同 batch size 的最高準確率		

討論：

這裡我們可以看到，當 batch size 較小時，其準確率的表現看起來比較好，而原因我認為是較小的 batch size，使更新頻率更高，可以更及時的更新模型參數，使模型更快收斂，但也有可能因為較小的 batch size 造成更不穩定的梯度估計，造成訓練過程不穩定。而 batch size 較大，會加快訓練速度，因為每一次迭代處理的樣本數量更多，能更有效利用運算資源，但也不是越大越快，如果太大的話可能會導致內存空間不夠，出現記憶體不足的問題，因此根據狀況選擇一個適合的 batch size 是很重要的



### 3. 調整 drop out



討論：

這裡我們可以看到，如果 drop out rate 太大，代表大部分的神經元都關閉，可能會導致模型更難訓練複雜的模型，導致欠擬合，而如果 drop out rate 太小，代表用大多數的神經元來訓練模型，這樣可能會造成模型過擬合，泛化能力會比較差，因此選擇適當的 drop out rate 是很重要的。

總結：

這此實驗我們實作兩個應用於腦電圖(EEG)信號分類的兩個網路，從架構上看 DeepConvNet 是一個相對較深的神經網路，由多個卷積層跟池化層交錯組成，最後透過全連結層進行分類，而 EEGNet 是一個相對較簡單的網路架構，只由一個混合卷積層，和深度可分離卷積層組合而成，並使用短時間平均池化(STAP)來獲得特徵，而在訓練的資源消耗上，DeepConvNet 因為架構較為複雜，參數量較大，需要更多的計算資源以及時間訓練。

而在足夠的訓練次數下，我們可以看到 EEGNet 相對於 DeepConvNet 表現更好，那是因為 EEGNet 架構較為簡單，參數量較少，可以避免過度擬合的問題，但我認為這也是取決於具體的數據集跟任務，在某些方面 DeepConvNet 可能因為複雜的架構，造成它的表現能力更好。

而我們也可以看到，對於各個不同的網路，其參數的調整是很重要的，參數的調整會直接影響到模型的性能和效果，因此仔細調整參數，找到最適合的配置，也是很重要的一環。