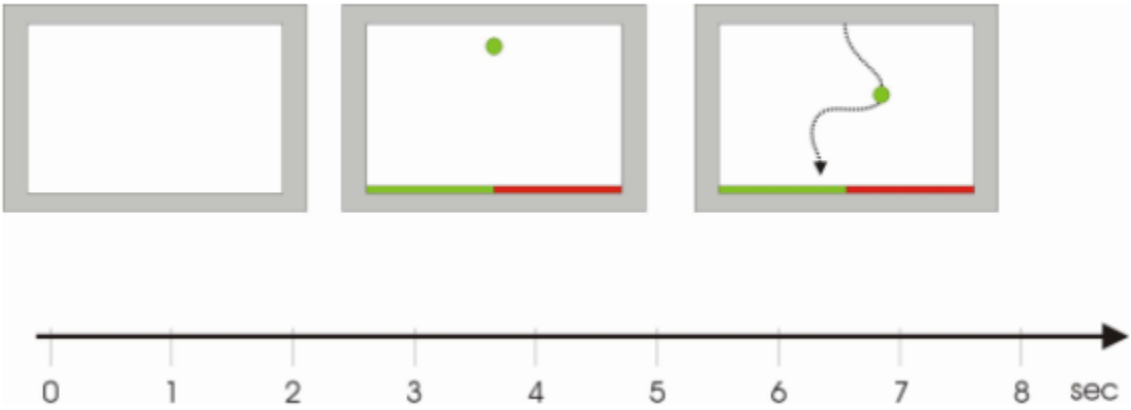
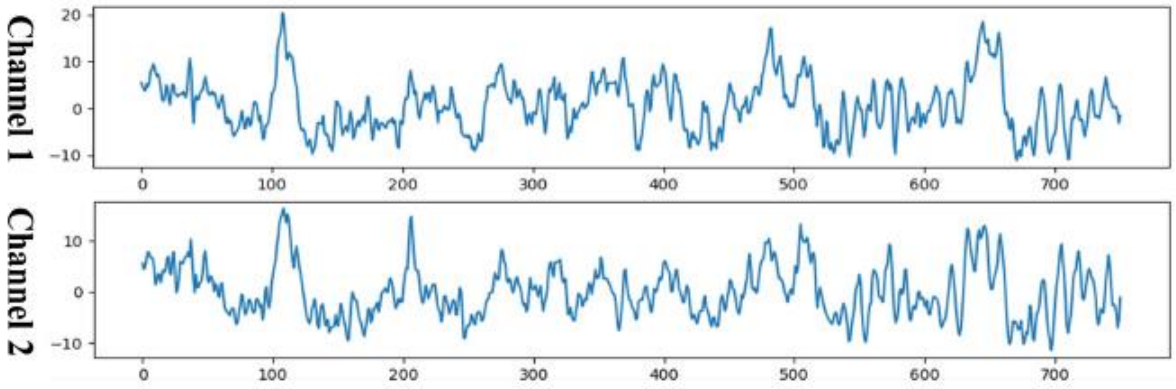
1. Introduction (20%)

在這個實驗，我們需要建立並運行 EEGNet 和 DeepConvNet，當中使用不同的激活函數 (ELU, ReLU, LeakyReLU)。然後通過調節不同的 Hyperparameter 使網絡達到高的準確率。

EEG 全名 Electroencephalography，是一種將電極置於頭皮檢測腦電波，並將腦電波紀錄起來的圖形。相同的想法會檢測出類似的腦電波，藉此來判斷同一類型的腦波並將其分類，此次的數據集是根據圖中的綠球來預測其會落下的點會是左邊還是右邊。



根據腦電波的輸入模型來判斷受試者的預測，其中以 EEGNet 和 DeepConvNet 來 evaluate 不同的激活函數，比較後得出最好的模型和激活函數。



上圖為兩個腦電波訊號的 Channel，可以看得出不同 Channel 其實也有著類似的 pattern。這次實驗的一個輸入裡分別有 2 個 Channel，750 長，然後輸出是 2(左或右)來分別出的綠球落點。

2. Experiment set up (30%)

A. The detail of your model

- EEGNet

```
class EEGNet(nn.Module):
    def __init__(self, activation_function, dropout):
        super(EEGNet, self).__init__()  # run init at Module
        self.first_conv = nn.Sequential(  # By sequence to form module 1 after 1
            # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1,
            # padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros',
device=None, dtype=None)
            nn.Conv2d(1, 16, (1, 51), (1, 1), (0, 25), bias=False),
            # torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1,
            # affine=True, track_running_stats=True, device=None, dtype=None)
            nn.BatchNorm2d(16))  # out_channel = 16
        self.depthwise_conv = nn.Sequential(
            nn.Conv2d(16, 32, (2, 1), (1, 1), groups=16, bias=False),
            nn.BatchNorm2d(32),
            activation_function,
            nn.AvgPool2d((1, 4), (1, 4)),  # take avg_sample of input, 750 -> 187
# torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False,
            # count_include_pad=True, divisor_override=None)
            nn.Dropout(dropout))
        # torch.nn.Dropout(p=0.5, inplace=False)
        self.separable_conv = nn.Sequential(
            nn.Conv2d(32, 32, (1, 15), (1, 1), (0, 7), bias=False),
            nn.BatchNorm2d(32),
            activation_function,
            nn.AvgPool2d((1, 8), (1, 8)),  # take avg_sample of input, 187 -> 23
            nn.Dropout(dropout))
        self.classify = nn.Sequential(
            nn.Linear(736, 2))  # 23 x 32 = 736
        # torch.nn.Linear(in_features, out_features, bias=True, device=None,
dtype=None)

    def forward(self, x):
        x = self.first_conv(x)
        x = self.depthwise_conv(x)
        x = self.separable_conv(x)
        x = x.flatten(1)  # flatten the shape to 2D by 32*1*23 = 736 as linear input
        x = self.classify(x)
        return x
```

- DeepConvNet

```
class DeepConvNet(nn.Module):
    def __init__(self, activation_function, dropout):
        super(DeepConvNet, self).__init__()
        self.Conv1 = nn.Sequential(
            nn.Conv2d(1, 25, (1, 5)),
            nn.Conv2d(25, 25, (2, 1)),
            nn.BatchNorm2d(25),
```

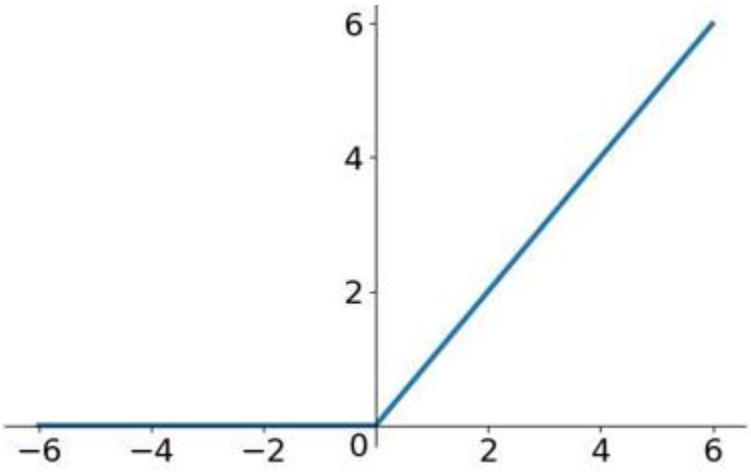
```
            activation_function,
            nn.MaxPool2d((1, 2)),
            # torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False)
            # default_stride = kernel_size
            nn.Dropout(p=dropout))
        self.Conv2 = nn.Sequential(
            nn.Conv2d(25, 50, (1, 5)),
            nn.BatchNorm2d(50),
            activation_function,
            nn.MaxPool2d((1, 2)),
            nn.Dropout(p=dropout))
        self.Conv3 = nn.Sequential(
            nn.Conv2d(50, 100, (1, 5)),
            nn.BatchNorm2d(100),
            activation_function,
            nn.MaxPool2d((1, 2)),
            nn.Dropout(p=dropout))
        self.Conv4 = nn.Sequential(
            nn.Conv2d(100, 200, (1, 5)),
            nn.BatchNorm2d(200),
            activation_function,
            nn.MaxPool2d((1, 2)),
            nn.Dropout(p=dropout),
            nn.Flatten())
        self.classify = nn.Sequential(
            nn.Linear(8600, 2))

    def forward(self, x):
        x = self.Conv1(x)  # 750 -> 746 => 746/2=373
        # (25, 50, (1, 5)) ([40, 25, 1, 373]) => (373-4)/2=184
        x = self.Conv2(x) # (50, 100, (1, 5)) ([40, 50, 1, 184]) => (184-4)/2=90
        x = self.Conv3(x) # (100, 200, (1, 5)) ([40, 100, 1, 90]) => (90-4)/2=43
        x = self.Conv4(x) # [40, 8600])=> 200 x 43 = 8600
        x = x = x.view(-1, self.classify[0].in_features)
        x = self.classify(x)
        return x
```

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

RelU 是一個激活函數其公式表達為：

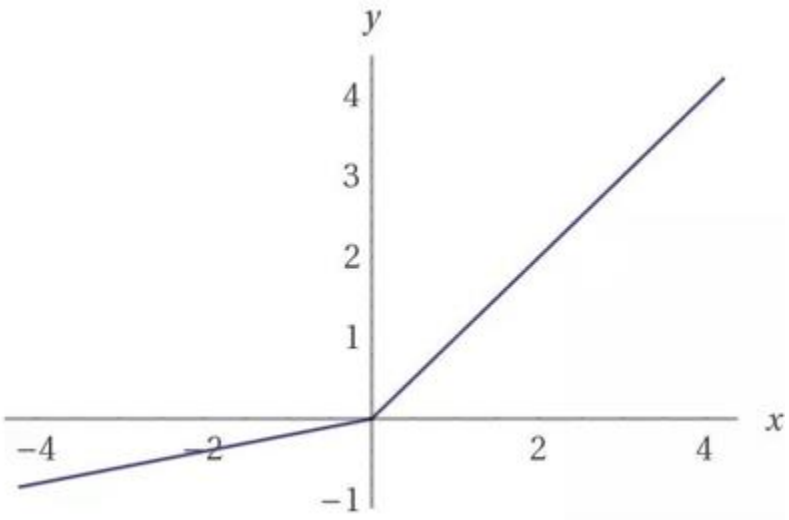
$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \text{ 或 } f(x) = max(0, x)$$



其作用為修正線性單一斜率的特性，以得到不同的線性組合達到單一線性不能達到的區分，並相對 sigmoid 有較好的 back propagation 表現，在深度學習中應用廣泛。

Leaky ReLU (LReLU)：

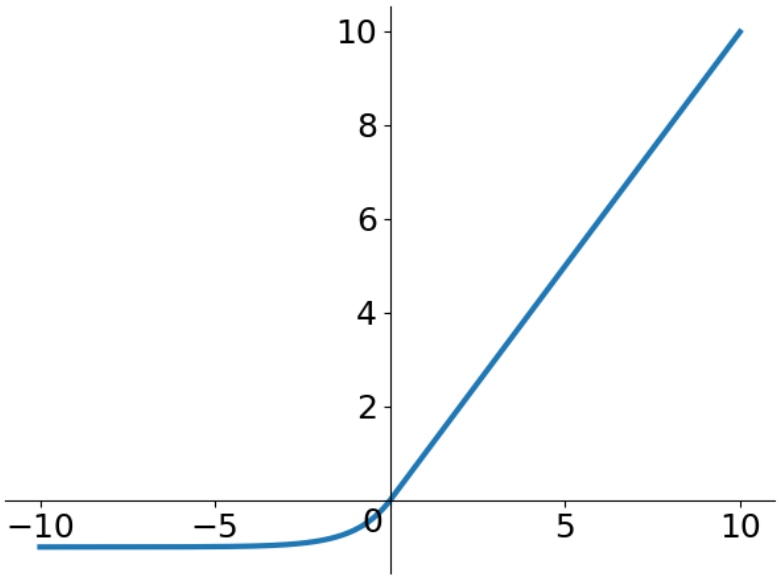
$$f(x) = \begin{cases} \alpha x, & x \leq 0 \\ x, & x > 0 \end{cases} \text{ 或 } f(x) = max(\alpha x, x)$$



其為 ReLU 的變種，將斜率為 0 的部份改為有一個較小的系數來使 gradient 不為零，改善斜率為 0 時不能夠學習的情況。

ELU：

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



在 ReLU 為 0 的斜率時，ELU 不為 0，類似 LReLU 的原理，但會慢慢趨向於-1，斜率亦慢慢接近 0。

3. Experimental results (30%)

A. The highest testing accuracy

	ELU	ReLU	Leaky ReLU
EEGNet	92.50%	95.00%	95.00%
DeepConvNet	90.00%	90.00%	95.00%

在以下環境設定中得出最大的準確率：

Epochs: 1001, batch size: 40, drop out: 0.5, learning rate (default): 0.01

基本上可以得出 Leaky ReLU 最為優秀的結論，下面為程式在 1000 個 epoch 中截取最大的 accuracy 的結果。

```
EEGNet_ELU
Max: test_epoch:692.0, test_acurracy:92.50%
EEGNet_ReLU
Max: test_epoch:28.0, test_acurracy:95.00%
EEGNet_Leaky_ReLU
Max: test_epoch:122.0, test_acurracy:95.00%
DeepConvNet_ELU
Max: test_epoch:37.0, test_acurracy:90.00%
DeepConvNet_ReLU
Max: test_epoch:91.0, test_acurracy:90.00%
DeepConvNet_Leaky_ReLU
Max: test_epoch:88.0, test_acurracy:95.00%
```

- Screenshot with two models

```
EEGNet Architecture
EEGNet(
  (first_conv): 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)
  )
  (depthwise_conv): 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=0.5, inplace=False)
  )
  (separable_conv): 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=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

```
DeepConvNet(
  (Conv1): 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): ELU(alpha=1.0)
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (Conv2): 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): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (Conv3): 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): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (Conv4): 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): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
    (5): Flatten(start_dim=1, end_dim=-1)
  )
  (classify): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```

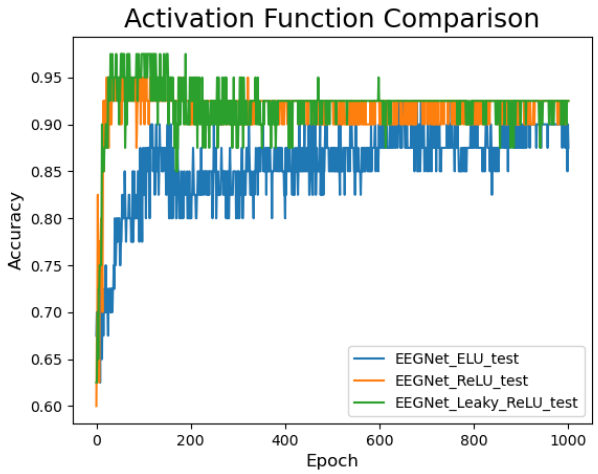
- anything you want to present

就網絡結構而言，DeepConvNet 有著更複雜的結構，其相對的運算時間也會更長，EEGNet 利用 Depthwise Separable Convolution 簡化了模型的複雜性，相對的運算成本也降低了。

B. Comparison figures

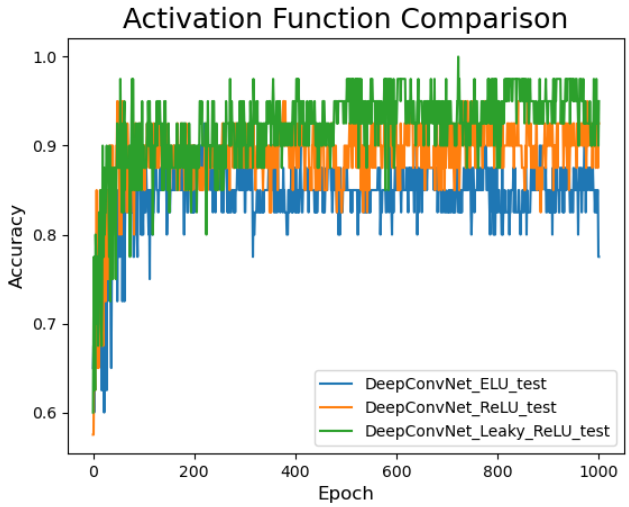
- EEGNet

在通過 1000 個 Epoches 訓練後，EEGNet 在用激活函數 ELU 平均落在 87.50%，用激活函數 ReLU 平均落在 92.50%，Leaky ReLU 平均也一樣落在 92.50%。



- DeepConvNet

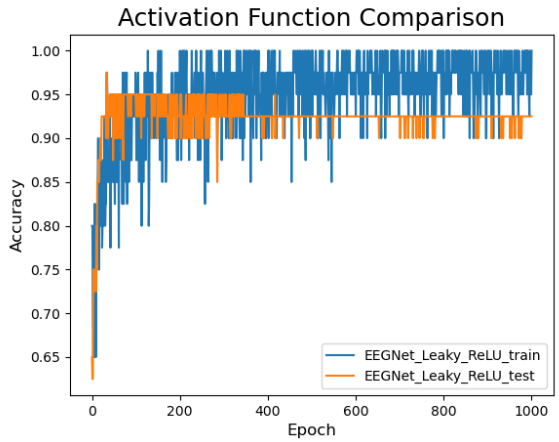
在通過 1000 個 Epoches 訓練後，EEGNet 在用激活函數 ELU 平均落在 85.00%，用激活函數 ReLU 平均落在 90.00%，Leaky ReLU 平均落在 93.75%。

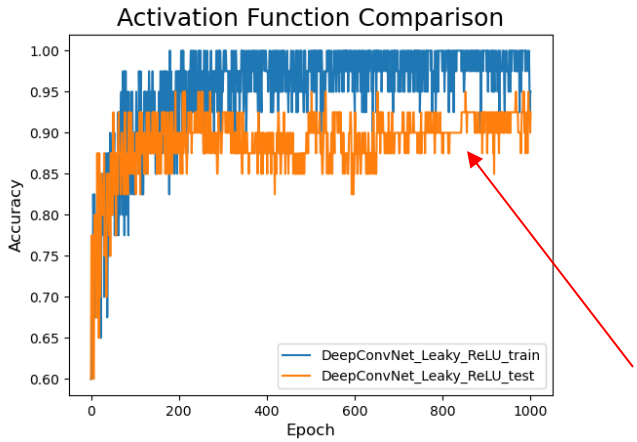


4. Discussion (20%)

A. Anything you want to share

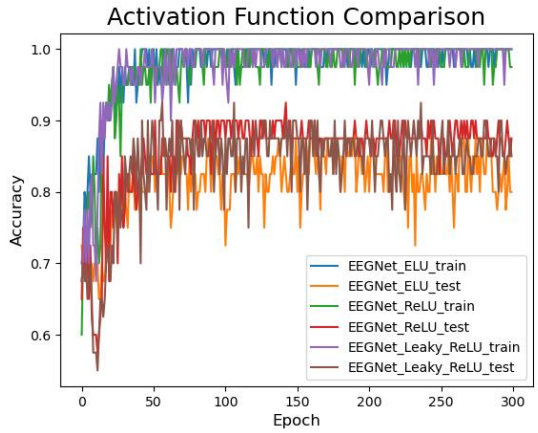
在訓練上，我們可以觀察到，模型在訓練資料上雖然已經多次達到 100%的準確率，但在 DeepConvNet 仍有進步的空間，可見 DeepConvNet 在大量 Training 後得到了些微上升。不過，在大約 100epoches 時，模型已經趨向穩定了。



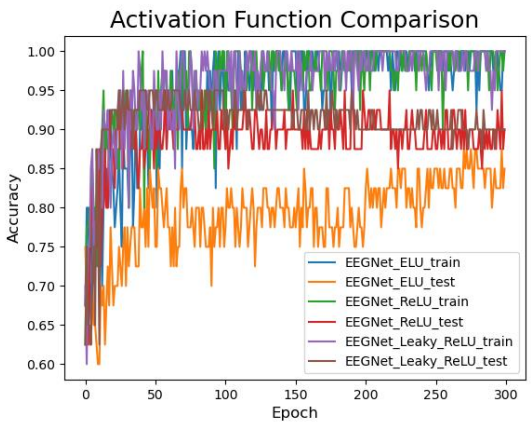


在得出最好的結果前，分別用了幾個方法：

Shuffle：將輸入資料打亂，讓模型不要學習到不必要的順序輸入 Pattern。可以明顯地看到，除了 ELU 外，ReLU 都得到了可觀的提升。

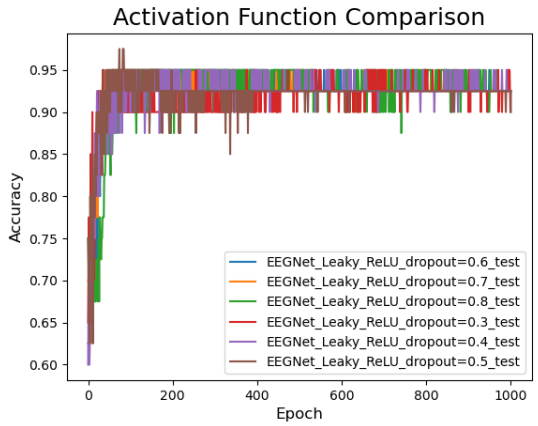


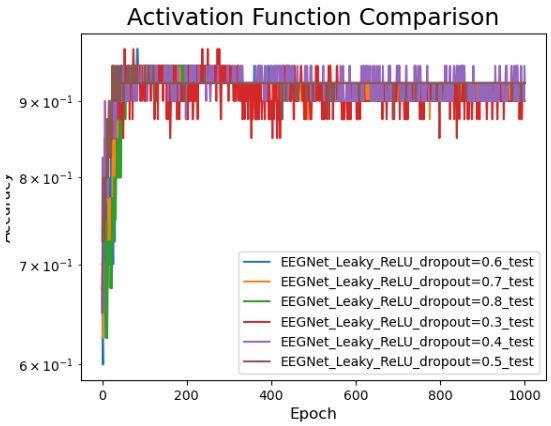
Shuffle: False



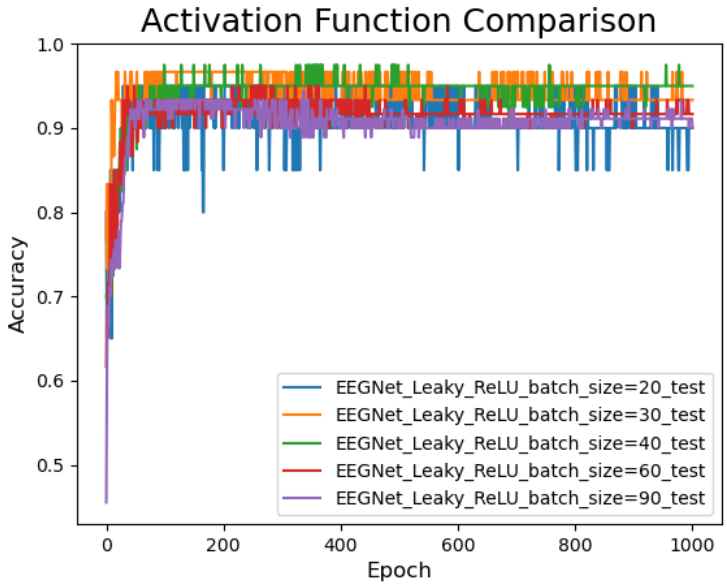
Shuffle: True

Dropout：通過忽略掉一定比例的特徵來防止模型過擬合，不同的 dropout 準確率平均值幾乎一樣，但 0.5 最為穩定，最大值亦是 0.5 最好，有 97.50%，所以選取了 0.5。



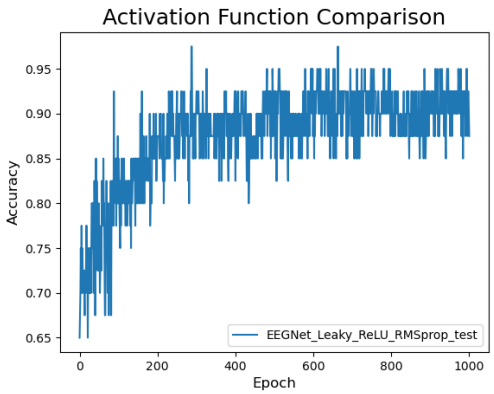


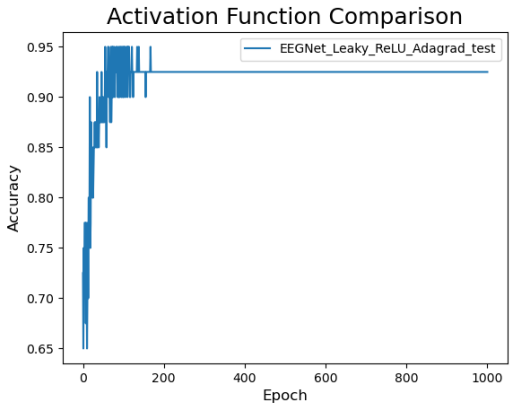
Batch Size：用不同大小的 Batch 來訓練資料， 令模型每次更新的 Step 有所不同。嘗試後得出 40 為最優的大小。

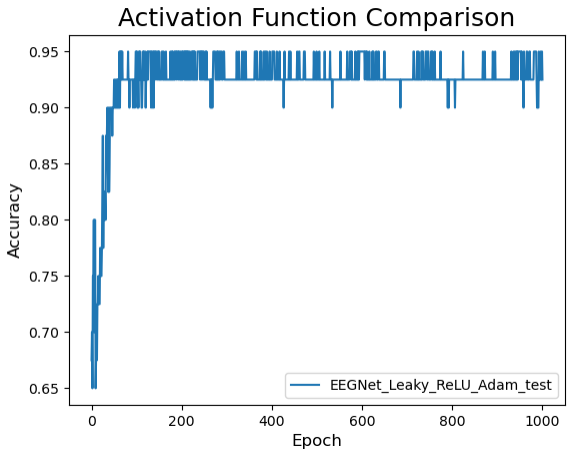


進一步嘗試不同的 optimizer 會不會有更好的適性：

可以從下圖得出 RMSprop 所需訓練時間較長，準確率亦沒有提升。Adagrad 則較為穩定，保持一定水平。但 Adam 仍然是此實驗中，準確值平均最好的 optimizer。(如下圖)

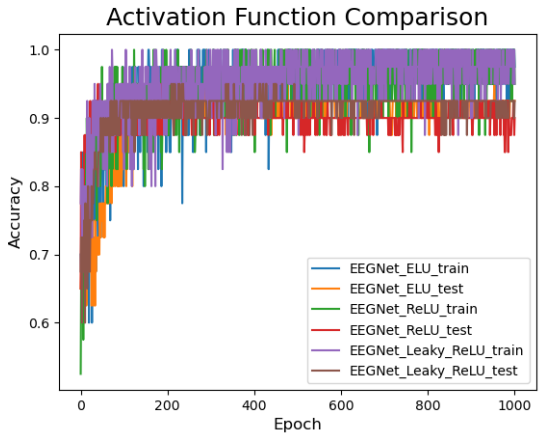


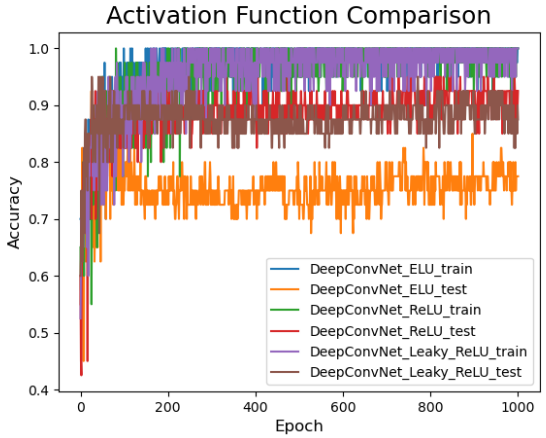




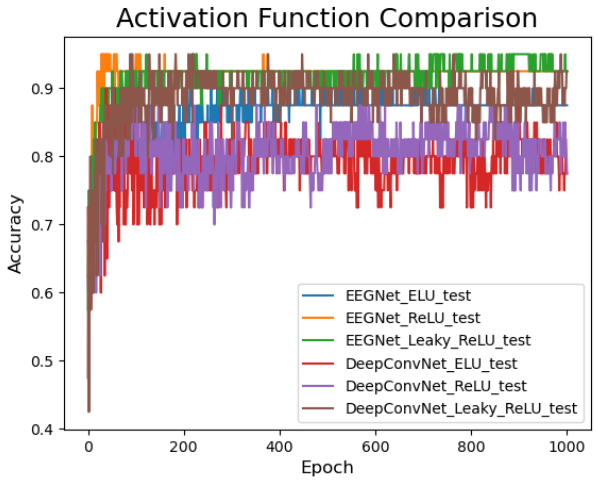
EEGNet 和 DeepConvNet 的訓練和測試結果比較：

雖然訓練多次達到 100%，但測試仍是達不到 100%。





在通過兩個模型的測試比較後(下圖)，可以得知 DeepConvNet 中的 ELU 和 ReLU 得出較差的值，而相對的 EEGNet 則無論在 ELU 或是 ReLU 都明顯優於 DeepConvNet，事實上，連 Leaky ReLU 也是以些微差距超過 DeepConvNet，這很容易就可以得知 EEGNet 在此類型的數據上是優於 DeepConvNet。(如下圖)



結論：EEGNet 明顯是優於 DeepConvNet 的模型，無論在運算量還是準確度都高於 DeepConvNet，Depthwise Separable Convolution 是一個很有用的模型簡化結構。在 3 個激活函數的比較上，我們得知 Leaky ReLU 為最好的一個，相信原因就是 gradient 不為 0 使每一次訓練都會有一定幅度的更新。