Lab3 EEG classification

邱以中

311551040

1. Introduction

這次的作業是要使用 EEGNET 與 DeepConvNet 兩個 network 來做分類任務，並使用 3 種不同的 activation function，分別是: Relu , Leaky relu, Elu 來進行訓練，並比較訓練結果的差異。

2. Experiment setup

- **EEG net**

```python
class EEGNet(nn.Module):
    def __init__(self,activate) -> None:
        super().__init__()

        if activate == "Relu":
            self.activate = nn.ReLU()
        elif activate == "LeakyRelu":
            self.activate = nn.LeakyReLU()
        elif activate == "Elu":
            self.activate = nn.ELU()

        self.firstconv = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=(1, 51),
                stride=(1, 1),
                padding=(0, 25),
                bias=False
            ),
            nn.BatchNorm2d(16)
        )
```

```
        self.depthwiseconv = nn.Sequential(
            nn.Conv2d(
                in_channels=16,
                out_channels=32,
                kernel_size=(2, 1),
                stride=(1, 1),
                groups=16,
                bias=False
            ),
            nn.BatchNorm2d(32),
            self.activate,
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(p=0.25)
        )
```

```
        self.separableconv = nn.Sequential(
            nn.Conv2d(
                in_channels=32,
                out_channels=32,
                kernel_size=(1, 15),
                stride=(1, 1),
                padding=(0, 7),
                bias=False
            ),
            nn.BatchNorm2d(32),
            self.activate,
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p=0.25)
        )
```

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


    def forward(self,x):
        x = self.firstconv(x)
        x = self.depthwiseconv(x)
        x = self.separableconv(x)
        x = self.classify(x)
        return x
```

- EEGNet implementation details

```
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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25)
  )
  (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): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

上圖為我對 EEGNet 的實踐，我會先在 init 的部分定義好 network，並在 forward function 執行 network 的預測。

EEGNet 使用 depthwise-separable 的架構來構建網路，可以有效減少網路的參數量，並且不影響 accuracy 的準確度。

- **DeepConvNet**

```python
class DeepConvNet(nn.Module):
    def __init__(self,activate) -> None:
        super().__init__()
        if activate == "Relu":
            self.activate = nn.ReLU()
        elif activate == "LeakyRelu":
            self.activate = nn.LeakyReLU()
        else:
            self.activate = nn.ELU()

        self.conv_0 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=25,
                kernel_size=(1, 5),
            ),
            nn.Conv2d(
                in_channels=25,
                out_channels=25,
                kernel_size=(2, 1),
            ),
            nn.BatchNorm2d(25),
            self.activate,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )

        self.conv_1 = nn.Sequential(
            nn.Conv2d(
                in_channels=25,
                out_channels=50,
                kernel_size=(1, 5),
            ),
            nn.BatchNorm2d(50),
            self.activate,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
```

```python
        self.conv_2 = nn.Sequential(
            nn.Conv2d(
                in_channels=50,
                out_channels=100,
                kernel_size=(1, 5),
            ),
            nn.BatchNorm2d(100),
            self.activate,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )

        self.conv_3 = nn.Sequential(
            nn.Conv2d(
                in_channels=100,
                out_channels=200,
                kernel_size=(1, 5),
            ),
            nn.BatchNorm2d(200),
            self.activate,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )

        self.classify = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=8600, out_features=2)
        )


    def forward(self,input):
        x = self.conv_0(input)
        x = self.conv_1(x)
        x = self.conv_2(x)
        x = self.conv_3(x)
        x = self.classify(x)
        return x
```

- You need to implement the DeepConvNet architecture by using the following table, where C = 2, T = 750 and N = 2. **The max norm term is ignorable.**

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 * 5 + 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 * 5 + 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 * 5 + 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

The input data has reshaped to [B, 1, C, T]

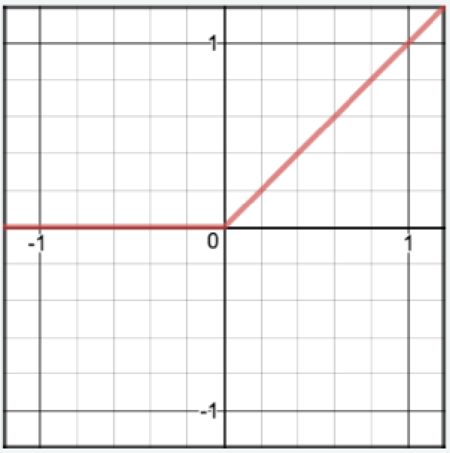
上圖為我對 DeepConvNet 的實踐，一樣是在 init 時構建網路，並在 forward function 執行。DeepConvNet 就是一般的深度網路，構建了多層的 Conv -> batchnorm -> activate -> pooling -> dropout 的 block。

- **Activation function**
  - **Relu**

    Relu 是一種常用的 activation function，在輸入<=0 輸出 0，輸入>0 時輸出=輸入，relu 的優點是運算較快並且可以解決梯度消失的問題，缺點是當輸入<0 時有可能會引發 ”dead relu” 的問題

Function

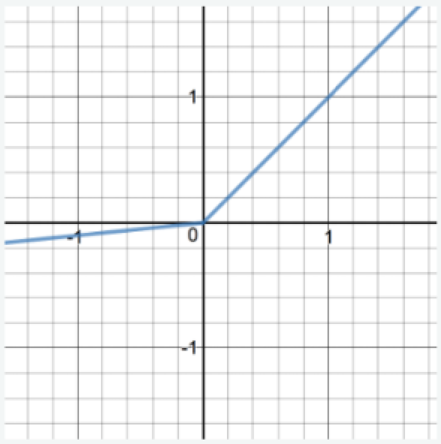
$$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$$



  - **Leaky Relu**

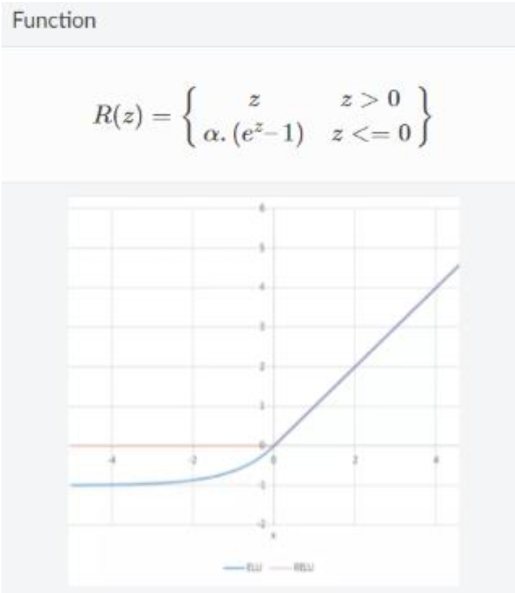
Function

$$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha z & z <= 0 \end{Bmatrix}$$



Leaky relu 是 relu 的改進版本，他會將小於 0 的輸入乘上一個 alpha，目的是為了解決 relu 可能造成的 “dead relu”問題。

■ ELU



ELU 相較於 relu 來說可以有負的輸出，並且在 0 的地方變得更平滑。

3. Experiment results

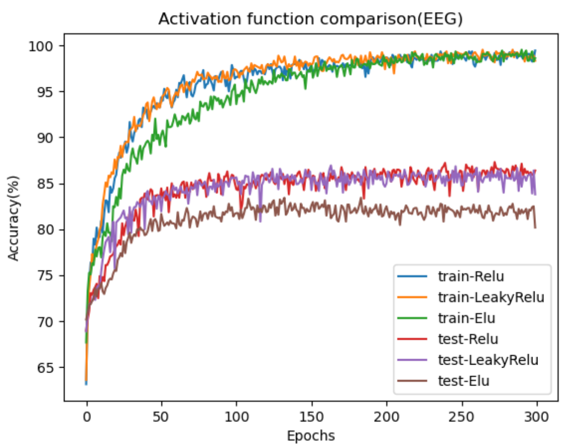
● Highest test accuracy

```
 highest testing accuracy
EEG_Relu: 87.31481481481481
EEG_LeakyRelu: 86.94444444444444
EEG_Elu: 83.42592592592592
Deep_Relu: 81.11111111111111
Deep_LeakyRelu: 81.01851851851852
Deep_Elu: 81.01851851851852
```

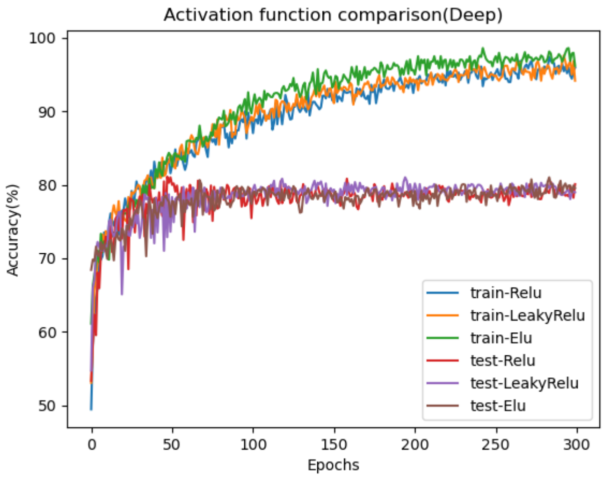
上圖為執行不同 network + 不同 activation function 在 testing 時最高的 accuracy。參數設置為 epochs: 300, batch size: 64, learning rate: 0.001

● Comparison figure

■ EEGNet



- ■ **DeepConvNet**



4. Discussion

- **Depthwise Separable Convolution**

  根據 https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728 網站的介紹，我們可以知道使用 Depthwise Separable Convolution，透過對不同 channel 分開做 convolution，最後再合併的操作可以有效的減少 convolution 的參數量，以此來提高運算的速度。

  當我們實際去看 EEGNet 與 DeepConvNet 也可以發現 EEGNet 的參數量確實有明顯的較少。

  - ■ **EEGNet**

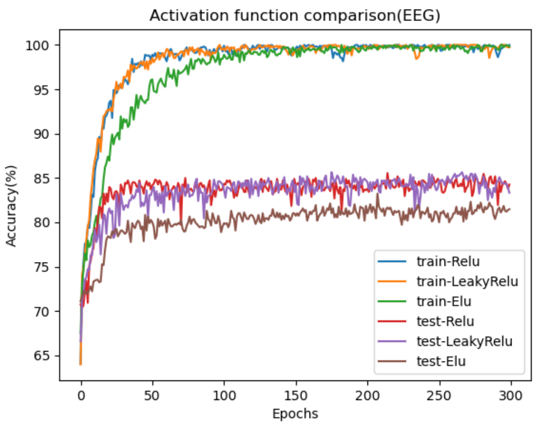
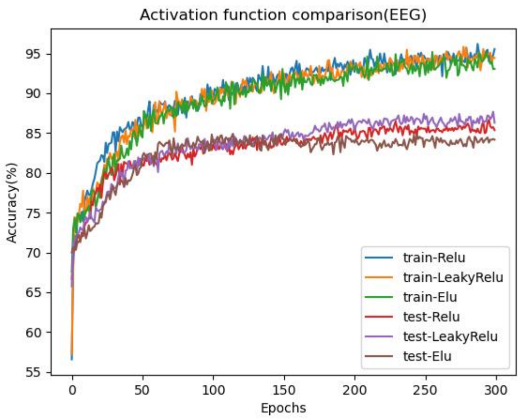
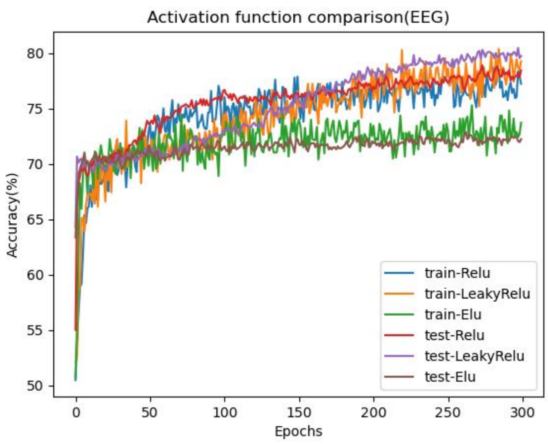
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 2, 750]             816
       BatchNorm2d-2           [-1, 16, 2, 750]              32
            Conv2d-3           [-1, 32, 1, 750]              64
       BatchNorm2d-4           [-1, 32, 1, 750]              64
              ReLU-5           [-1, 32, 1, 750]               0
              ReLU-6           [-1, 32, 1, 750]               0
              ReLU-7           [-1, 32, 1, 750]               0
         AvgPool2d-8           [-1, 32, 1, 187]               0
           Dropout-9           [-1, 32, 1, 187]               0
           Conv2d-10           [-1, 32, 1, 187]          15,360
      BatchNorm2d-11           [-1, 32, 1, 187]              64
             ReLU-12           [-1, 32, 1, 187]               0
             ReLU-13           [-1, 32, 1, 187]               0
             ReLU-14           [-1, 32, 1, 187]               0
        AvgPool2d-15            [-1, 32, 1, 23]               0
          Dropout-16            [-1, 32, 1, 23]               0
          Flatten-17                  [-1, 736]               0
           Linear-18                    [-1, 2]           1,474
================================================================
Total params: 17,874
Trainable params: 17,874
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.62
Params size (MB): 0.07
Estimated Total Size (MB): 1.69
----------------------------------------------------------------
```

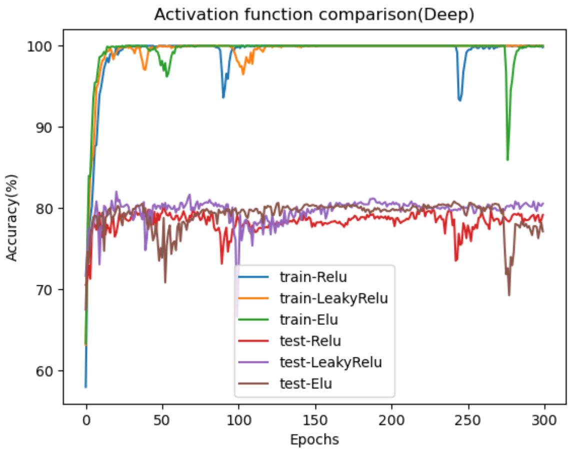
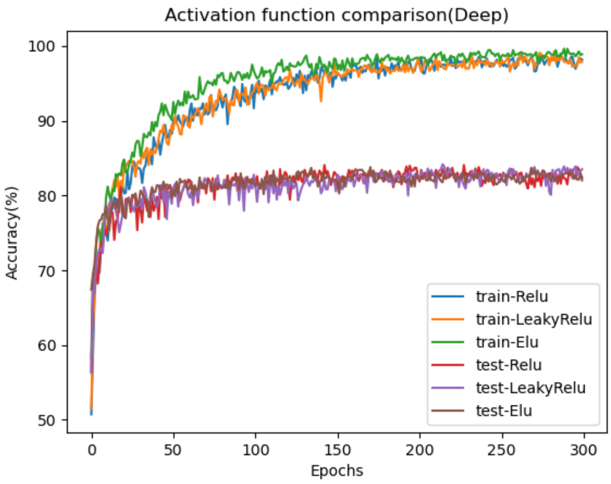
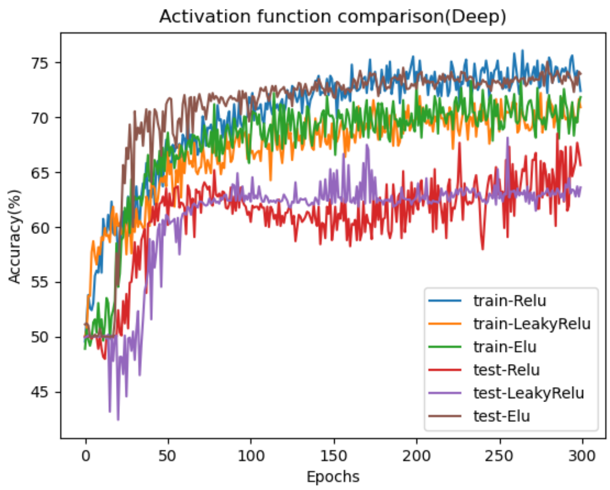
■ **DeepConvNet**

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 25, 2, 746]             150
            Conv2d-2           [-1, 25, 1, 746]           1,275
       BatchNorm2d-3           [-1, 25, 1, 746]              50
              ReLU-4           [-1, 25, 1, 746]               0
              ReLU-5           [-1, 25, 1, 746]               0
              ReLU-6           [-1, 25, 1, 746]               0
              ReLU-7           [-1, 25, 1, 746]               0
              ReLU-8           [-1, 25, 1, 746]               0
         MaxPool2d-9           [-1, 25, 1, 373]               0
          Dropout-10           [-1, 25, 1, 373]               0
           Conv2d-11           [-1, 50, 1, 369]           6,300
      BatchNorm2d-12           [-1, 50, 1, 369]             100
             ReLU-13           [-1, 50, 1, 369]               0
             ReLU-14           [-1, 50, 1, 369]               0
             ReLU-15           [-1, 50, 1, 369]               0
             ReLU-16           [-1, 50, 1, 369]               0
             ReLU-17           [-1, 50, 1, 369]               0
        MaxPool2d-18           [-1, 50, 1, 184]               0
          Dropout-19           [-1, 50, 1, 184]               0
           Conv2d-20          [-1, 100, 1, 180]          25,100
      BatchNorm2d-21          [-1, 100, 1, 180]             200
             ReLU-22          [-1, 100, 1, 180]               0
             ReLU-23          [-1, 100, 1, 180]               0
             ReLU-24          [-1, 100, 1, 180]               0
             ReLU-25          [-1, 100, 1, 180]               0
             ReLU-26          [-1, 100, 1, 180]               0
        MaxPool2d-27           [-1, 100, 1, 90]               0
          Dropout-28           [-1, 100, 1, 90]               0
           Conv2d-29           [-1, 200, 1, 86]         100,200
      BatchNorm2d-30           [-1, 200, 1, 86]             400
             ReLU-31           [-1, 200, 1, 86]               0
             ReLU-32           [-1, 200, 1, 86]               0
             ReLU-33           [-1, 200, 1, 86]               0
             ReLU-34           [-1, 200, 1, 86]               0
             ReLU-35           [-1, 200, 1, 86]               0
        MaxPool2d-36           [-1, 200, 1, 43]               0
          Dropout-37           [-1, 200, 1, 43]               0
          Flatten-38                 [-1, 8600]               0
           Linear-39                    [-1, 2]          17,202
================================================================
Total params: 150,977
Trainable params: 150,977
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.76
Params size (MB): 0.58
Estimated Total Size (MB): 5.34
----------------------------------------------------------------
```

● **Dropout**

在這次的實作當中，我也嘗試去改變 dropout 的值，來看他對於訓練的影響，可以發現在 dropout 很小的時候，會稍微有點 overfitting 的情況發生，training 的 accuracy 能夠接近到 100%，但是 testing 只有到 85% 左右。另一方面，我們可以發現在 dropout 很大的時候，因為刪除掉太多神經元，導致模型能力不足，只能得到較低的 accuracy。因此選擇適當的 dropout 對 model 的訓練是非常重要的。

<table>
<tr>
<td>EEG
Droupout
= 0.1</td>
<td>
</td>
<td>highest testing accuracy
EEG_Relu: 85.55555555555556
EEG_LeakyRelu: 85.64814814814815
EEG_Elu: 83.24074074074075
Deep_Relu: 80.37037037037037
Deep_LeakyRelu: 82.03703703703704
Deep_Elu: 80.83333333333333</td>
</tr>
<tr>
<td>EEG
Droupout
= 0.5</td>
<td>
</td>
<td>highest testing accuracy
EEG_Relu: 86.66666666666667
EEG_LeakyRelu: 87.68518518518519
EEG_Elu: 84.9074074074074
Deep_Relu: 84.07407407407408
Deep_LeakyRelu: 84.16666666666667
Deep_Elu: 83.79629629629629</td>
</tr>
<tr>
<td>EEG
Droupout
= 0.9</td>
<td>
</td>
<td>highest testing accuracy
EEG_Relu: 78.88888888888889
EEG_LeakyRelu: 80.46296296296296
EEG_Elu: 72.87037037037037
Deep_Relu: 68.51851851851852
Deep_LeakyRelu: 68.14814814814815
Deep_Elu: 75.0925925925926</td>
</tr>
</table>

<table>
<tr>
<td>Deep
Dropout
=0.1</td>
<td>
</td>
<td>highest testing accuracy
EEG_Relu: 85.55555555555556
EEG_LeakyRelu: 85.64814814814815
EEG_Elu: 83.24074074074075
Deep_Relu: 80.37037037037037
Deep_LeakyRelu: 82.03703703703704
Deep_Elu: 80.83333333333333</td>
</tr>
<tr>
<td>Deep
Dropout
=0.5</td>
<td>
</td>
<td>highest testing accuracy
EEG_Relu: 86.66666666666667
EEG_LeakyRelu: 87.68518518518519
EEG_Elu: 84.9074074074074
Deep_Relu: 84.07407407407408
Deep_LeakyRelu: 84.16666666666667
Deep_Elu: 83.79629629629629</td>
</tr>
<tr>
<td>Deep
Dropout
=0.9</td>
<td>
</td>
<td>highest testing accuracy
EEG_Relu: 78.88888888888889
EEG_LeakyRelu: 80.46296296296296
EEG_Elu: 72.87037037037037
Deep_Relu: 68.51851851851852
Deep_LeakyRelu: 68.14814814814815
Deep_Elu: 75.0925925925926</td>
</tr>
</table>