311551148 lab3 Report

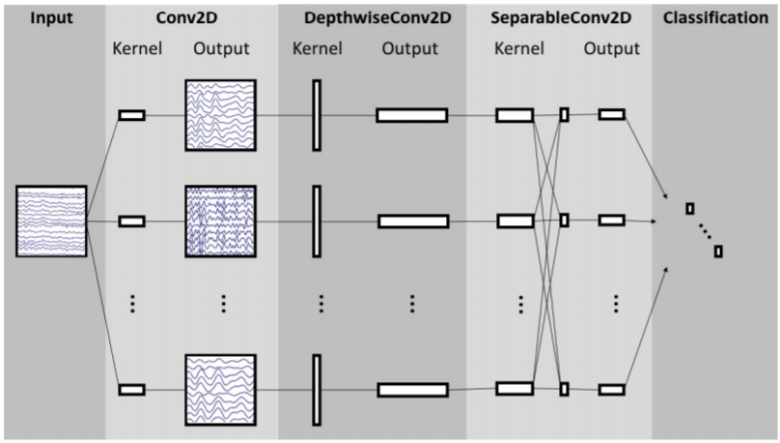
1. Introduction

   In recent years, CNN have been widely applied to electroencephalogram(EEG) for feature extraction and classification. In this lab, we are asked to implement two simple models for EEG classification, which are EEGNet and DeepConvNet. This model are train and evaluated on a BCI competition dataset. We also need to compare the result using different activation function, including ReLU, Leaky ReLU and ELU.

2. Experiment set up

   甲、The details of your model

   i. EEGNet



上圖是 EEGNet 的模型架構，先使用普通的 Conv2D 得到 feature map，接著使用 DepthwiseConv2D 以及 separableConv2D 最後使用一層 linear layer 得到 classification output，參數細節為下圖

```
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)
  )
)
```

用 pytorch 可以很簡單的實作出來，所有 layer 都有在 torch.nn 裡，在最後的 linear layer 前把 feature map flatten 就好

```python
class EEGNet(nn.Module):
    def __init__(self, activation="relu", dropout_p=None):
        super(EEGNet, self).__init__()

        ## Select activation function
        if activation == "relu":
            self.activation = nn.ReLU()
        elif activation == "leaky_relu":
            self.activation = nn.LeakyReLU()
        elif activation == "elu":
            self.activation = nn.ELU()

        ## Set dropout probability
        self.dropout_p = 0.25 if dropout_p is None else dropout_p

        ## EEGNet layers
        self.firstConv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(1, 51), padding=(0, 25), bias=False),
            nn.BatchNorm2d(16)
        )
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1), groups=16, bias=False),
            nn.BatchNorm2d(32),
            self.activation,
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4)),
            nn.Dropout(p=self.dropout_p)
        )
        self.separableConv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=(1, 15), padding=(0, 7), bias=False),
            nn.BatchNorm2d(32),
            self.activation,
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8)),
            nn.Dropout(p=self.dropout_p)
        )
        self.classify = nn.Sequential(
            nn.Linear(736, 2)
        )
```

```python
def forward(self, x):
    ## x.shape = (bs, 1, 2, 750)
    x = self.firstConv(x) ## (bs, 16, 2, 750)
    x = self.depthwiseConv(x) ## (bs, 32, 1, 187)
    x = self.separableConv(x) ## (bs, 32, 1, 23)

    ## Flatten
    x = x.view(x.shape[0], -1) ## (bs, 736)
    y = self.classify(x) ## (bs, 2)
    return y
```

ii. DeepConvNet

使用下圖的參數實作 DeepConvNet，C=2, T=750, N=2

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

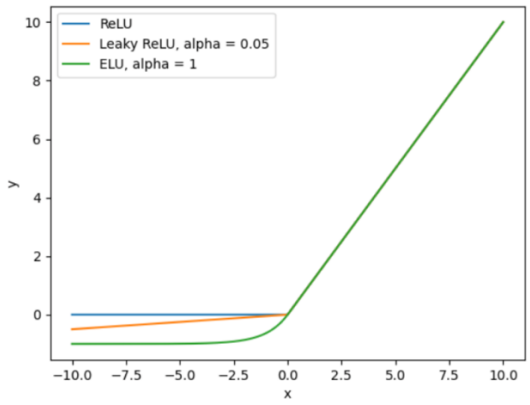
DeepConvNet 可以拆解成第一層 convolution，中間有四坨 convolution，裡面分別都是 conv2d, batchnorm, activation, maxpool, dropout，最後也是 flatten 後連接 linear layer 得到 classification output。

```python
class DeepConvNet(nn.Module):
    def __init__(self, activation="relu", dropout_p=None):
        super(DeepConvNet, self).__init__()

        ## Select activation function
        if activation == "relu":
            self.activation = nn.ReLU()
        elif activation == "leaky_relu":
            self.activation = nn.LeakyReLU()
        elif activation == "elu":
            self.activation = nn.ELU()

        ## Set dropout probability
        self.dropout_p = 0.5 if dropout_p is None else dropout_p

        ## Parameters for convs
        out_channels = [25, 25, 50, 100, 200]
        kernel_sizes = [(2, 1), (1, 5), (1, 5), (1, 5)]

        ## DeepConvNet layers
        self.conv0 = nn.Conv2d(1, 25, kernel_size=(1, 5))
        self.convs = nn.ModuleList()
        for idx in range(4):
            conv_i = nn.Sequential(
                nn.Conv2d(out_channels[idx], out_channels[idx + 1], kernel_size=kernel_sizes[idx]),
                nn.BatchNorm2d(out_channels[idx + 1]),
                self.activation,
                nn.MaxPool2d(kernel_size=(1, 2)),
                nn.Dropout(p=self.dropout_p)
            )
            self.convs.append(conv_i)
        self.classify = nn.Linear(8600, 2)
```

```python
def forward(self, x):
    ## x.shape = (bs, 1, 2, 750)
    x = self.conv0(x) ## (bs, 25, 2, 746)
    for conv_i in self.convs: ## (bs, 25, 1, 373) ## (bs, 50, 1, 184) ## (bs, 100, 1, 90) ## (bs, 200, 1, 43)
        x = conv_i(x)

    ## Flatten
    x = x.view(x.shape[0], -1) ## (bs, 8600)
    y = self.classify(x) ## (bs, 2)
    return y
```

乙、Explain the activation function(ReLU, Leaky ReLU, ELU)

這三個 activation function 的函數圖形為下圖，各自要代的數學公式也放在下面，ReLU 是最簡單的，但是有一個問題就是當 x<0 時，他會直接把他設成 0，導致 gradient=0，這個問題叫做 dying ReLU problem，為了解決這個問題所以延伸出 Leaky ReLU 以及 RLU 兩種 ReLU 的變形，他們在 x<0 時也會有 gradient，可以解決前面提到的 Dying ReLU problem，可以注意到 ELU 跟 ReLU, Leaky ReLU 不同，他在 x=0 時也可以微分。



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

$$\text{LeakyReLU}(x, \alpha) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}, \text{ where } 0 \leq \alpha \leq 1.$$

$$\text{ELU}(x, \alpha) = \begin{cases} x, & \text{if } x > 0 \\ \alpha e^x - 1, & \text{if } x \leq 0 \end{cases}, \text{ where } 0 \leq \alpha \leq 1.$$

3. Experimental Result

Lr	EGGNet			DeepConvNet		
	ReLU	Leaky ReLU	ELU	ReLU	Leaky ReLU	ELU
0.01	87.13	84.81	83.43	82.78	82.22	81.67
0.005	86.57	86.48	81.76	82.78	81.11	80.00
0.002	87.22	86.20	83.61	81.57	82.87	81.39
0.001	87.22	86.39	84.54	80.56	83.43	81.11
0.0005	86.11	87.41	84.72	81.48	82.04	82.22
0.0002	87.59	86.20	84.63	82.87	81.48	80.74
0.0001	86.85	86.48	83.61	82.87	83.33	81.20

甲、The highest testing accuracy

i. Screenshot with two model

EGGNet best test accuracy 是使用 learning rate = 0.0002 搭配 ReLU

activation function，DeepConvNet best test accuracy 是使用 learning rate = 0.001 搭配 Leaky ReLU activation function，下方為訓練時的的截圖，裡面有 training loss, training accuracy, testing accuracy

EGGNet

```
Using relu as activation function...
Start training...
Epoch:    0, Train Loss: 0.6180, Train Accuracy: 0.6454, Test Accuracy: 0.6741
Epoch:   10, Train Loss: 0.4146, Train Accuracy: 0.8167, Test Accuracy: 0.7370
Epoch:   20, Train Loss: 0.3267, Train Accuracy: 0.8491, Test Accuracy: 0.7852
Epoch:   30, Train Loss: 0.2474, Train Accuracy: 0.9037, Test Accuracy: 0.7769
Epoch:   40, Train Loss: 0.1851, Train Accuracy: 0.9250, Test Accuracy: 0.8083
Epoch:   50, Train Loss: 0.1611, Train Accuracy: 0.9380, Test Accuracy: 0.8278
Epoch:   60, Train Loss: 0.1349, Train Accuracy: 0.9509, Test Accuracy: 0.8361
Epoch:   70, Train Loss: 0.1110, Train Accuracy: 0.9565, Test Accuracy: 0.8463
Epoch:   80, Train Loss: 0.0871, Train Accuracy: 0.9694, Test Accuracy: 0.8537
Epoch:   90, Train Loss: 0.0965, Train Accuracy: 0.9685, Test Accuracy: 0.8509
Epoch:  100, Train Loss: 0.0791, Train Accuracy: 0.9759, Test Accuracy: 0.8556
Epoch:  110, Train Loss: 0.0798, Train Accuracy: 0.9676, Test Accuracy: 0.8620
Epoch:  120, Train Loss: 0.0738, Train Accuracy: 0.9778, Test Accuracy: 0.8306
Epoch:  130, Train Loss: 0.0496, Train Accuracy: 0.9870, Test Accuracy: 0.8583
Epoch:  140, Train Loss: 0.0757, Train Accuracy: 0.9694, Test Accuracy: 0.8583
Epoch:  150, Train Loss: 0.0587, Train Accuracy: 0.9815, Test Accuracy: 0.8611
Epoch:  160, Train Loss: 0.0617, Train Accuracy: 0.9713, Test Accuracy: 0.8593
Epoch:  170, Train Loss: 0.0647, Train Accuracy: 0.9759, Test Accuracy: 0.8556
Epoch:  180, Train Loss: 0.0452, Train Accuracy: 0.9843, Test Accuracy: 0.8546
Epoch:  190, Train Loss: 0.0444, Train Accuracy: 0.9843, Test Accuracy: 0.8611
Epoch:  200, Train Loss: 0.0452, Train Accuracy: 0.9870, Test Accuracy: 0.8537
Epoch:  210, Train Loss: 0.0363, Train Accuracy: 0.9889, Test Accuracy: 0.8630
Epoch:  220, Train Loss: 0.0352, Train Accuracy: 0.9889, Test Accuracy: 0.8741
Epoch:  230, Train Loss: 0.0413, Train Accuracy: 0.9843, Test Accuracy: 0.8639
Epoch:  240, Train Loss: 0.0329, Train Accuracy: 0.9889, Test Accuracy: 0.8648
Epoch:  250, Train Loss: 0.0246, Train Accuracy: 0.9935, Test Accuracy: 0.8620
Epoch:  260, Train Loss: 0.0403, Train Accuracy: 0.9898, Test Accuracy: 0.8657
Epoch:  270, Train Loss: 0.0338, Train Accuracy: 0.9861, Test Accuracy: 0.8667
Epoch:  280, Train Loss: 0.0313, Train Accuracy: 0.9907, Test Accuracy: 0.8620
Epoch:  290, Train Loss: 0.0399, Train Accuracy: 0.9870, Test Accuracy: 0.8741
Best Epoch: 295, Test Accuracy: 0.8759
```

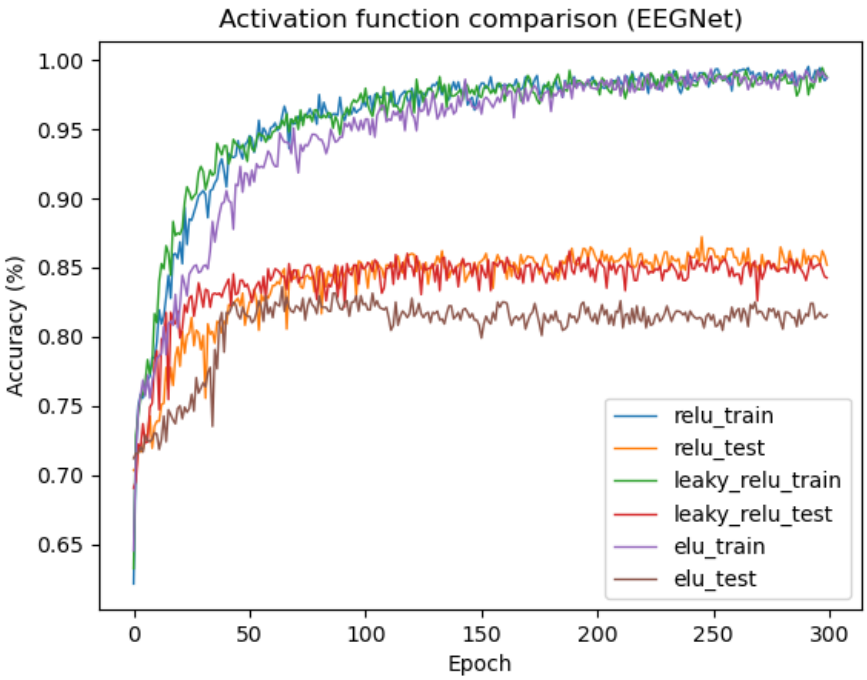
DeepConvNet

```
Using leaky_relu as activation function...
Start training...
Epoch:    0, Train Loss: 1.4836, Train Accuracy: 0.5009, Test Accuracy: 0.5426
Epoch:   10, Train Loss: 0.5376, Train Accuracy: 0.7435, Test Accuracy: 0.7213
Epoch:   20, Train Loss: 0.4693, Train Accuracy: 0.7824, Test Accuracy: 0.7852
Epoch:   30, Train Loss: 0.4413, Train Accuracy: 0.7972, Test Accuracy: 0.7796
Epoch:   40, Train Loss: 0.4083, Train Accuracy: 0.8250, Test Accuracy: 0.7676
Epoch:   50, Train Loss: 0.3611, Train Accuracy: 0.8398, Test Accuracy: 0.7694
Epoch:   60, Train Loss: 0.3258, Train Accuracy: 0.8546, Test Accuracy: 0.7935
Epoch:   70, Train Loss: 0.3348, Train Accuracy: 0.8648, Test Accuracy: 0.8056
Epoch:   80, Train Loss: 0.3119, Train Accuracy: 0.8676, Test Accuracy: 0.8028
Epoch:   90, Train Loss: 0.2314, Train Accuracy: 0.9000, Test Accuracy: 0.7889
Epoch:  100, Train Loss: 0.2429, Train Accuracy: 0.8935, Test Accuracy: 0.8056
Epoch:  110, Train Loss: 0.2220, Train Accuracy: 0.9111, Test Accuracy: 0.8037
Epoch:  120, Train Loss: 0.2651, Train Accuracy: 0.8889, Test Accuracy: 0.7556
Epoch:  130, Train Loss: 0.2225, Train Accuracy: 0.9111, Test Accuracy: 0.8009
Epoch:  140, Train Loss: 0.2016, Train Accuracy: 0.9222, Test Accuracy: 0.8083
Epoch:  150, Train Loss: 0.2225, Train Accuracy: 0.9139, Test Accuracy: 0.7824
Epoch:  160, Train Loss: 0.1994, Train Accuracy: 0.9222, Test Accuracy: 0.8139
Epoch:  170, Train Loss: 0.1644, Train Accuracy: 0.9380, Test Accuracy: 0.8093
Epoch:  180, Train Loss: 0.1605, Train Accuracy: 0.9333, Test Accuracy: 0.8120
Epoch:  190, Train Loss: 0.1419, Train Accuracy: 0.9407, Test Accuracy: 0.8074
Epoch:  200, Train Loss: 0.1264, Train Accuracy: 0.9481, Test Accuracy: 0.8213
Epoch:  210, Train Loss: 0.1388, Train Accuracy: 0.9444, Test Accuracy: 0.8074
Epoch:  220, Train Loss: 0.1551, Train Accuracy: 0.9435, Test Accuracy: 0.8213
Epoch:  230, Train Loss: 0.1110, Train Accuracy: 0.9593, Test Accuracy: 0.8176
Epoch:  240, Train Loss: 0.1616, Train Accuracy: 0.9370, Test Accuracy: 0.8120
Epoch:  250, Train Loss: 0.1431, Train Accuracy: 0.9407, Test Accuracy: 0.8194
Epoch:  260, Train Loss: 0.1331, Train Accuracy: 0.9481, Test Accuracy: 0.8194
Epoch:  270, Train Loss: 0.1548, Train Accuracy: 0.9370, Test Accuracy: 0.8102
Epoch:  280, Train Loss: 0.1403, Train Accuracy: 0.9519, Test Accuracy: 0.8148
Epoch:  290, Train Loss: 0.0992, Train Accuracy: 0.9630, Test Accuracy: 0.8269
Best Epoch: 282, Test Accuracy: 0.8343
```

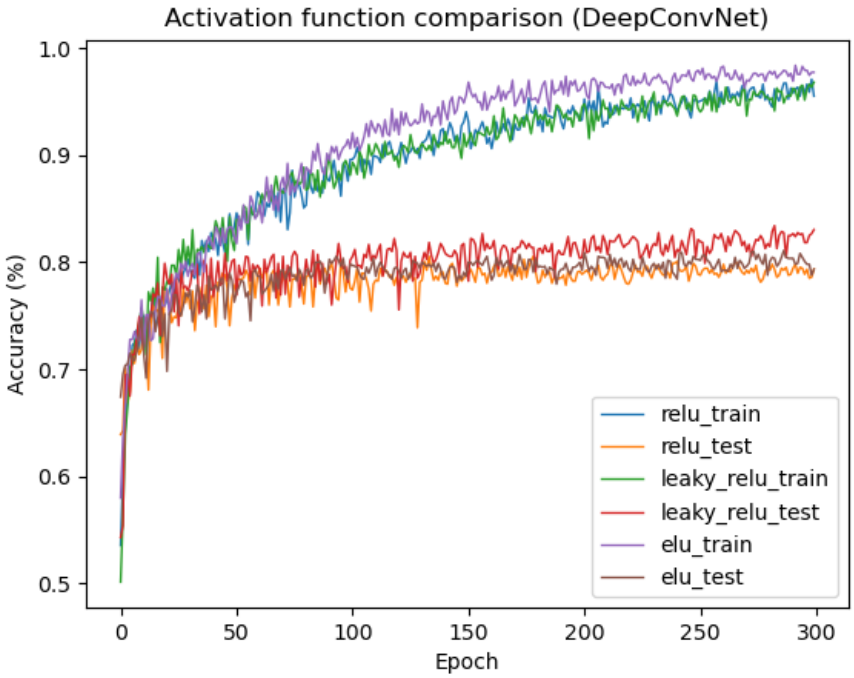
乙、Comparison figures

可以注意到大概 50 epoch 後 testing accuracy 就不太上升，但是 training accuracy 卻持續上升，代表模型開始 overfitting training data

i. EEGNet



ii. DeepConvNet



4. Discussion

甲、Anything you want to share

觀察前面 experiment result 的表可以發現，EGGNet 的結果比 DeepConvNet 好很多，可以看出來 depth wise convolution 可以更好的

解析 EEG data，相比一般的 deep convolution network。
另外從 comparison figure 可以看出來，ELU activation function 看起來有比較嚴重的 overfitting problem，相比 ReLU, Leaky ReLU，他的 training accuracy 與 testing accuracy 在模型收斂後相差更大