1. Introduction

   這次的 Lab 使用 pytorch 實作 Deep Neural network 來做 EEG 腦電圖訊號的分類，分別使用兩個 model，第一個是 EEGNet，已經有好的參數以及架構，照著寫出來即可，DeepConvNet 則是使用許多層的 Convolutional layer 實作 Convolution, BatchNorm, activation, 以及 Pooling 和 dropout，最後再經過 flatten 以及 linear layer 得出 classifier 的結果。

   此次因為需要常常更改訓練參數，於是提供了 parser 功能可以直接下參數修改訓練參數

2. Experiment setup

   A. The detail of your model

      - EEGNet

```python
class EEGNet(torch.nn.Module):
    def __init__(self,activation):
        super().__init__()

        self.firstConv = Sequential(Conv2d(1, 16, kernel_size = (1,51), stride=(1,1), padding = (0,25), bias=False),
                                    BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats =True))
        self.depthWiseConv = Sequential(Conv2d(16,32,kernel_size = (2,1), stride=(1,1),groups=16, bias=False),
                                        BatchNorm2d(32, eps=1e-05, momentum=0.1, affine = True, track_running_stats=True),
                                        activation(),
                                        AvgPool2d(kernel_size=(1,4),stride=(1,4),padding=0),
                                        Dropout(p=0.25))
        self.separableConv = Sequential(Conv2d(32,32,kernel_size=(1,15),stride=(1,1),padding=(0,7),bias=False),
                                        BatchNorm2d(32, eps=1e-05, momentum=0.1, affine = True, track_running_stats=True),
                                        activation(),
                                        AvgPool2d(kernel_size=(1,8),stride=(1,8),padding = 0),
                                        Dropout(p=0.25))
        self.classify = Sequential(Flatten(),Linear(in_features = 736, out_features =2 ,bias = True))
    def forwardPass(self, inputs):
        NetResult = self.separableConv(self.depthWiseConv(self.firstConv(inputs)))
        return self.classify(NetResult)
```

實作上使用了 pytorch，讓 EEGNet 繼承 torch.nn.module，然後依照 spec 上面的內容 implement， 並在最後用 forward 將 input pass 到 classifier 之前，最後再送進 classifier 並回傳答案

- DeepConvNet

```
class DeepConvNet(torch.nn.Module):
    def __init__(self, activation, dropout):
        super().__init__()
        self.Conv1 = Sequential(Conv2d(1,25,kernel_size= (1,5), bias = False),
                                Conv2d(25,25,kernel_size = (2,1), bias = False),
                                BatchNorm2d(25, eps=1e-05, momentum=0.1),
                                activation(),
                                MaxPool2d(kernel_size = (1,2)),
                                Dropout(p=dropout))
        self.Conv2 = Sequential(Conv2d(25,50,kernel_size= (1,5), bias = False),
                                BatchNorm2d(50, eps=1e-05, momentum=0.1),
                                activation(),
                                MaxPool2d(kernel_size = (1,2)),
                                Dropout(p=dropout))
        self.Conv3 = Sequential(Conv2d(50,100,kernel_size= (1,5), bias = False),
                                BatchNorm2d(100, eps=1e-05, momentum=0.1),
                                activation(),
                                MaxPool2d(kernel_size = (1,2)),
                                Dropout(p=dropout))
        self.Conv4 = Sequential(Conv2d(100,200,kernel_size= (1,5), bias = False),
                                BatchNorm2d(200, eps=1e-05, momentum=0.1),
                                activation(),
                                MaxPool2d(kernel_size = (1,2)),
                                Dropout(p=dropout))
        flatten_size = 8600
        output_size = 2
        self.classify = Sequential(Flatten(),Linear(in_features = flatten_size, out_features = output_size,bias = True))
    def forwardPass(self, inputs):
        for i in range(1,5):
            inputs = getattr(self, f'Conv{i}')(inputs)
        return self.classify(inputs)
```

實作上用了 4 層的 Convolutional layer，Convolution 的部分取 kernel size = (1,5)，Pooling 的部分使用 kernel size = (1,2)

算起來 data 經過 layer 的 tensor 大小就會如下:

[1,1,2,750] -> conv1 -> [1,25,2,373] -> conv2 -> [1,50,2,184] -> conv3 -> [1,100,2,90] -> conv4 -> [1,200,1,43]

於是, flatten size = 1*200*1*43 = 8600

Output size 則是 2

最後經 flatten()再經 Linear layer 得到 classify 結果

B. Explain activation function (ReLU, LeakyReLU, ELU)

- ReLU: Rectified Linear Unit(修正線性單元)

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

  ReLU 會將所有負值改成 0, 會使得負值的 neuron 不會繼續修正 error, 因為 gradient 會變成 0

- Leaky ReLU:

  $\mathrm{Leaky\_ReLU(x)} = \max(\alpha\mathrm{x}, \mathrm{x})$

  Leaky_ReLU 相較於 ReLU 並沒有那麼極端, 會使負值的 neuron 變的沒負那麼多, 會乘上一個 α, 這個 α 基本上會設成 0.1

- ELU: Exponential linear unit

  $$\mathrm{ELU(x)} = \begin{cases} \mathrm{x} & ,\mathrm{x} \geq 0 \\ \alpha(e^{\mathrm{x}}\text{-}1), & \mathrm{x} < 0 \end{cases}$$

  在負值的地方使用 exponential 來處理而得名, 可以將負值限制在> - α

以上，並且函數在 0 附近比較光滑

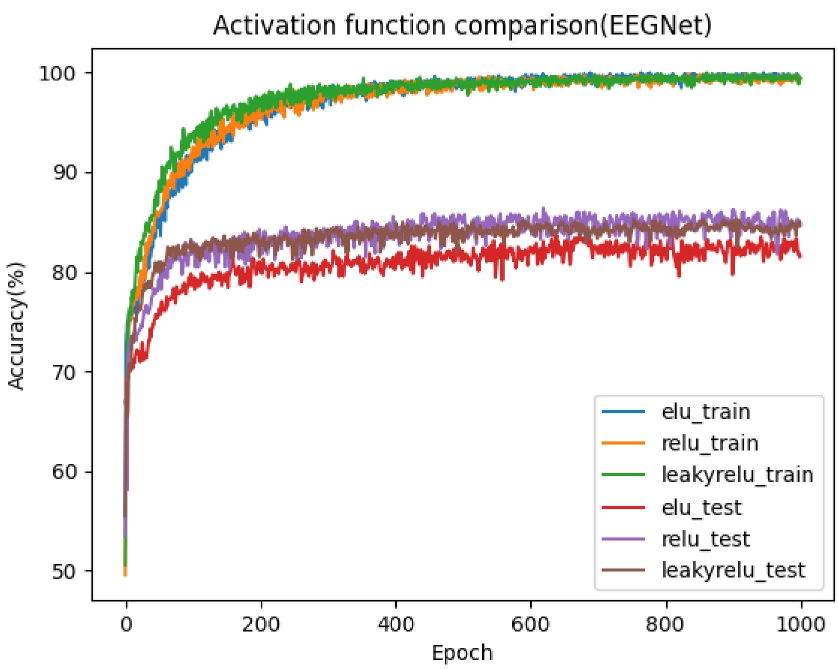
3. Experimental result
   A. The highest testing accuracy
      a. DeepConvNet (with epoch 1000, batch size = 512, optimizer = adamax, learning rate = 0.001)

```
elu_train: 100.00 %
relu_train: 100.00 %
leakyrelu_train: 100.00 %
elu_test: 81.02 %
relu_test: 81.20 %
leakyrelu_test: 81.02 %
```

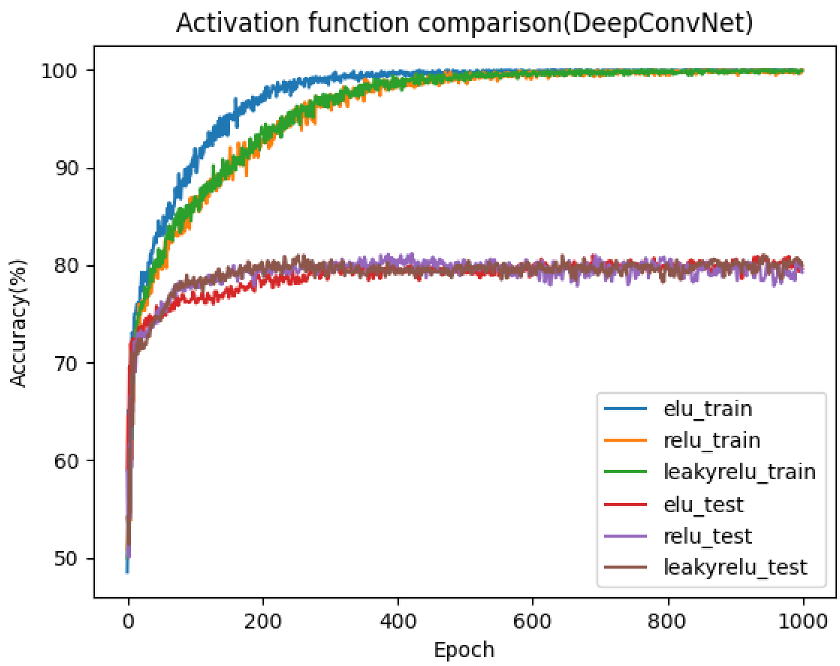
      b. EEGNet(with epoch 10000, batch size = 512, optimizer = adamax, learning rate = 0.001)

```
elu_train: 100.00 %
relu_train: 100.00 %
leakyrelu_train: 100.00 %
elu_test: 84.91 %
relu_test: 86.57 %
leakyrelu_test: 86.30 %
```

   B. Comparison figures
      a. EEGNet



b. DeepConvNet

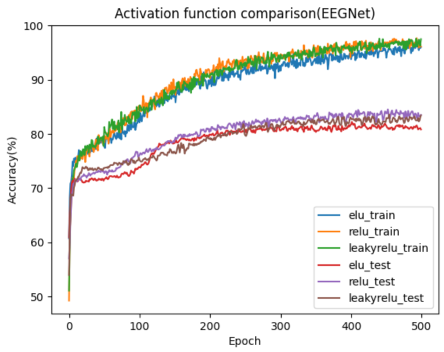


4. Discusssion

I try to implement different optimizers to get the different comparison results.

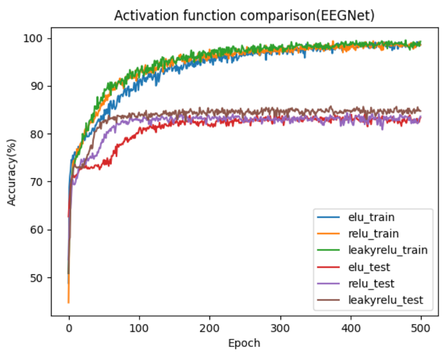
A. With epoch=500, batch size = 512, learning rate = 0.001, model = EEG

a. With adamax optimizer



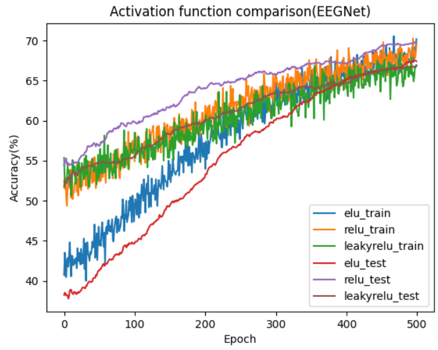

```
elu_train: 96.76 %
relu_train: 97.69 %
leakyrelu_train: 97.59 %
elu_test: 82.22 %
relu_test: 84.54 %
leakyrelu_test: 83.61 %
```

b. With adam optimizer



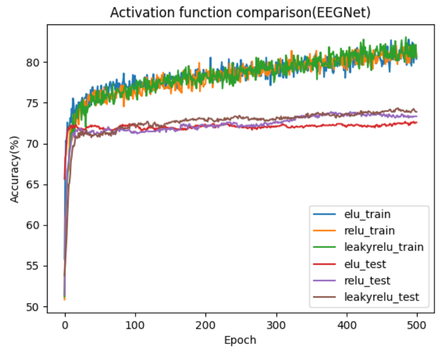

```
elu_train: 99.17 %
relu_train: 99.35 %
leakyrelu_train: 99.54 %
elu_test: 84.07 %
relu_test: 84.35 %
leakyrelu_test: 85.74 %
```

c. With adadelta optimizer



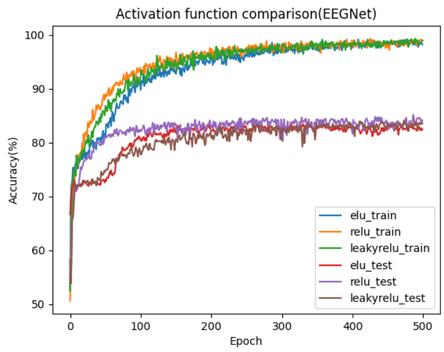

```
elu_train: 70.56 %
relu_train: 70.28 %
leakyrelu_train: 68.52 %
elu_test: 67.50 %
relu_test: 69.81 %
leakyrelu_test: 66.85 %
```

d. With adagrad optimizer




```
elu_train: 82.78 %
relu_train: 82.04 %
leakyrelu_train: 83.06 %
elu_test: 72.69 %
relu_test: 73.89 %
leakyrelu_test: 74.35 %
```

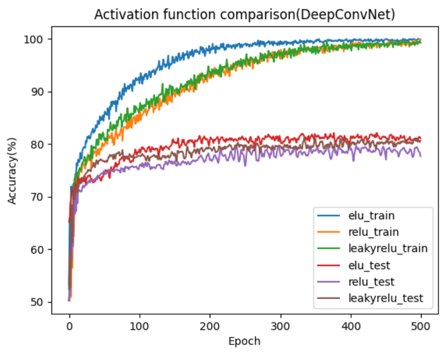
e. With adamw optimizer




```
elu_train: 99.26 %
relu_train: 99.35 %
leakyrelu_train: 99.35 %
elu_test: 83.89 %
relu_test: 85.19 %
leakyrelu_test: 84.17 %
```

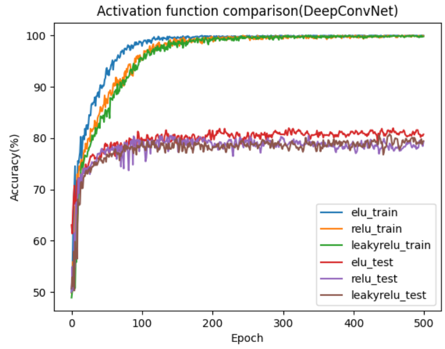
B. With With epoch=500, batch size = 512, learning rate = 0.001, model = “DeepConv”

a. With adamax optimizer



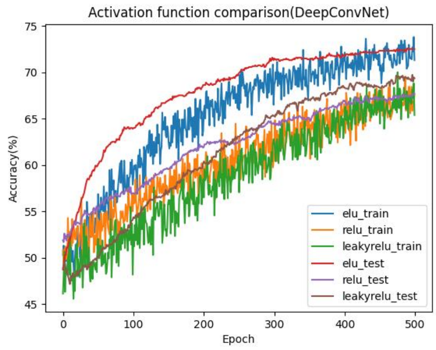

```
elu_train: 100.00 %
relu_train: 99.91 %
leakyrelu_train: 99.63 %
elu_test: 82.13 %
relu_test: 79.72 %
leakyrelu_test: 81.11 %
```

b. With adam optimizer



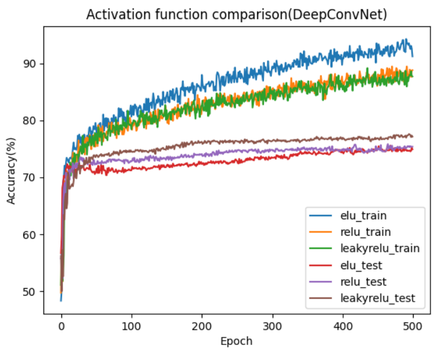

```
elu_train: 100.00 %
relu_train: 100.00 %
leakyrelu_train: 100.00 %
elu_test: 81.94 %
relu_test: 80.46 %
leakyrelu_test: 80.83 %
```

c. With adadelta optimizer



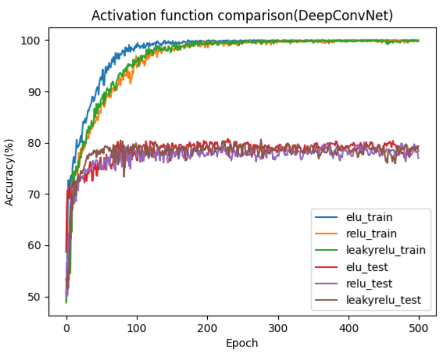

```
elu_train: 73.80 %
relu_train: 69.54 %
leakyrelu_train: 70.00 %
elu_test: 72.59 %
relu_test: 67.69 %
leakyrelu_test: 69.72 %
```

d. With adagrad optimizer




```
elu_train: 94.26 %
relu_train: 89.44 %
leakyrelu_train: 89.17 %
elu_test: 75.56 %
relu_test: 75.83 %
leakyrelu_test: 77.59 %
```

e. With adamw optimizer




```
elu_train: 100.00 %
relu_train: 100.00 %
leakyrelu_train: 100.00 %
elu_test: 80.74 %
relu_test: 80.09 %
leakyrelu_test: 80.65 %
```

C. result

a. EEG: 利用此五種 optimizer 做 EEG 的預測發現 adam, adamw, adamax 的表現較好, adadelta 以及 adagrad 的效果較差

b. DeepConv: 利用此五種 optimizer 做 DeepConv 的預測一樣是 adadelta 以及 adagrad 的效果比較差，尤其是 adadelta 甚至 training set 也無法足夠有效的預測成功，而 adagrad 則是較差一些但不會差到太嚴重