Lab2: EEG classification

Student id / name: A113599 / 楊淨富

I. Introduction:

分別訓練兩個 models，一個為 EEG net，另一個為 Deep convolutional net。並將 BCI(brain-computer interface)腦波圖的 dataset 作為 input 輸入至兩個 models，並進行 2-class 的 classification。

II. Experiment setups

A. The detail of your model

- EEGNet

```
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)
  )
)
```

先對 input data 進行 pre-processing，用 torch.utils.data 這個 module 的 DataLoader，將 train_data, train_label, test_data, test_label 分別轉換成 tensor 後包在一起。

之後是 network 的 architecture，除了 input layer 與 output layer 外，中間的 hidden layer 分為三層。

- 第一層為 CNN
- 第二層為 Depthwise convolution

- 第三層為 Separable convolution

而 Depthwise convolution 及 Separable convolution 的計算是希望在不影響輸出結構的狀況下減少運算量，具體參數皆按照 spec。

此外，此 model 有一些 Hyper parameters，以下是我 tune 的參數:

- Batch size = 108
- Learning rate = 0.001
- Epochs = 1201
- Optimizer: torch.optim.Adam()
- Loss function: torch.nn.CrossEntropyLoss()

- DeepConvNet

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 * 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

Input 和 EEGNet 一樣，都是先用 DataLoader 包成一包。

Network 的 architecture，除了 input layer 與 output layer 外，中間的 hidden layer 共四層，每層皆為 CNN。但除了第一層內部用到了 2 層的 CNN 外，其餘第二到第四層皆使用一層的 CNN，並且再透過 Batch normalization、Activation funcion、Max pooling 2D 和 Dropout 來做為該層最後的 output。

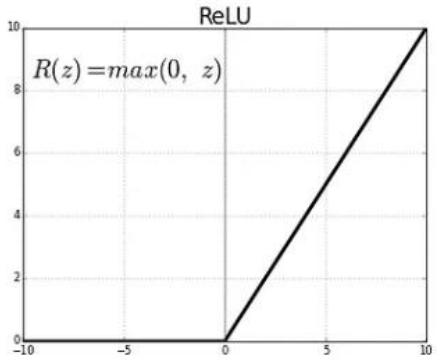
- Batch normalization:透過對每個 batch 進行標準化處理，有助於加速訓練過程(因為能更順利傳遞 gradient 從而加快收斂速度)，並提高模型的泛化性(generalization)。
- Activation function: Non-linearity
- Max pooling 2D:降維操作，透過保留最顯著的特徵並丟棄不重要的細節來減少計算量。
- Dropout: 在訓練過程中隨機地丟棄或關閉一部份 neuron 來防止 overfitting，並且可以提高模型的泛化性(generalization)。

此外，此 model 有一些 Hyper parameters，以下是我 tune 的參數:

- Batch size = 1080
- Learning rate = 0.001
- Epochs = 1201
- Optimizer: torch.optim.Adam()
- Loss function: torch.nn.CrossEntropyLoss()

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

- ReLU:

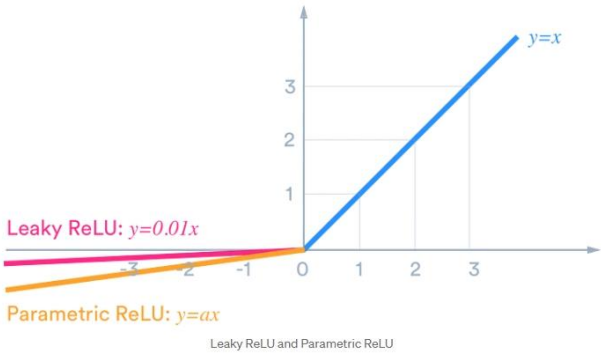


  - 如果輸入小於 0，則它的輸出為 0，否則輸出原始輸入。

  - 優點:
    - 避免並修正梯度消失的問題。
    - 計算的 cost 比 tanh 和 sigmoid 更低。

  - 缺點:
    - ReLU 應該用於 hidden layer，不適用於 output layer。
    - 在訓練過程中，某些梯度可能會變得脆弱並消失。這可能導致權重更新，使得某些神經元永遠不再對任何數據點產生活性，即 ReLu 可能導致神經元失效。
    - 對於 ReLu 的負區域（x < 0），梯度將為 0，這意味著在梯度下降過程中權重不會進行調整。這意味著處於該狀態的神經元將不再對誤差/輸入的變化做出響應（因為梯度為 0，什麼都不會變化）。這稱為「dying ReLu problem」。
    - ReLU 的輸出範圍是[0,∞)，這意味著它可能使 activation 的值爆炸。

- LeakyReLU:



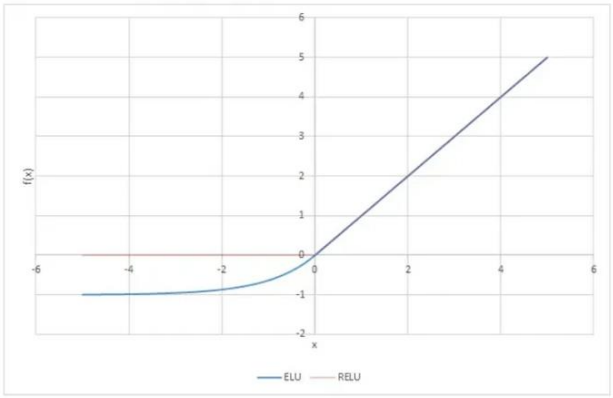
Leaky ReLU and Parametric ReLU

  - LeakyReLU 是 ReLU 的一種變形。不同於 ReLU 在 y < 0 時輸出為 0，LeakyReLU 允許一個小的、非零的常數斜率 α（通常為 α=0.01）。然而，目前這種優勢在所有任務中是否都能一致取得更好的結果仍然不確定。

  - 優點:
    - 透過具有一個小的負斜率(通常為 0.01 左右)來解決 dying ReLU problem。

  - 缺點:
    - 由於它具有線性特性，它不能用於複雜的分類任務。對於某些例子，LeakyReLU 的表現不如 Sigmoid 和 Tanh。

- ELU(Exponential LU):



ELU blue ReLU brown

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ a\left(e^x - 1\right) & \text{otherwise,} \end{cases}$$

alpha is a hyper parameter, with positive value constraint

- ELU 產生的結果比 ReLU 更準確，並且收斂速度更快。對於>0 的正輸入，ELU 和 ReLU 相同，但對於<0 的負輸入，ELU 緩慢地平滑到-α 而 ReLU 則急遽平滑。

- 優點:
  - ELU 的輸出在平滑到-α 時變化較慢，而 ReLU 的平滑程度較快。
  - 與 ReLU 不同，ELU 可以產生負輸出。

- 缺點:

  - 與 ReLU 相同，因為輸出範圍是[0,∞)，這意味著它可能使 activation 的值爆炸。

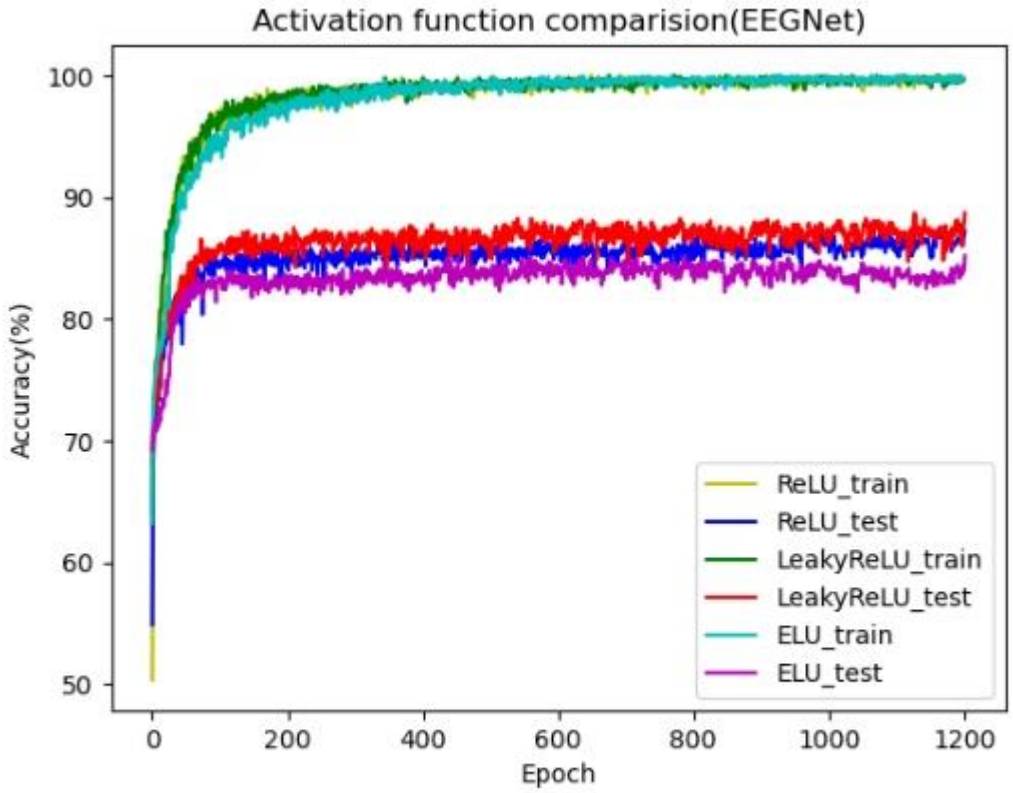
III. Experimental results

A. The highest testing accuracy

	EEGNet	DeepConvNet
ReLU	87.22%	85.09%
LeakyReLU	88.70%	83.43%
ELU	85.19%	83.06%

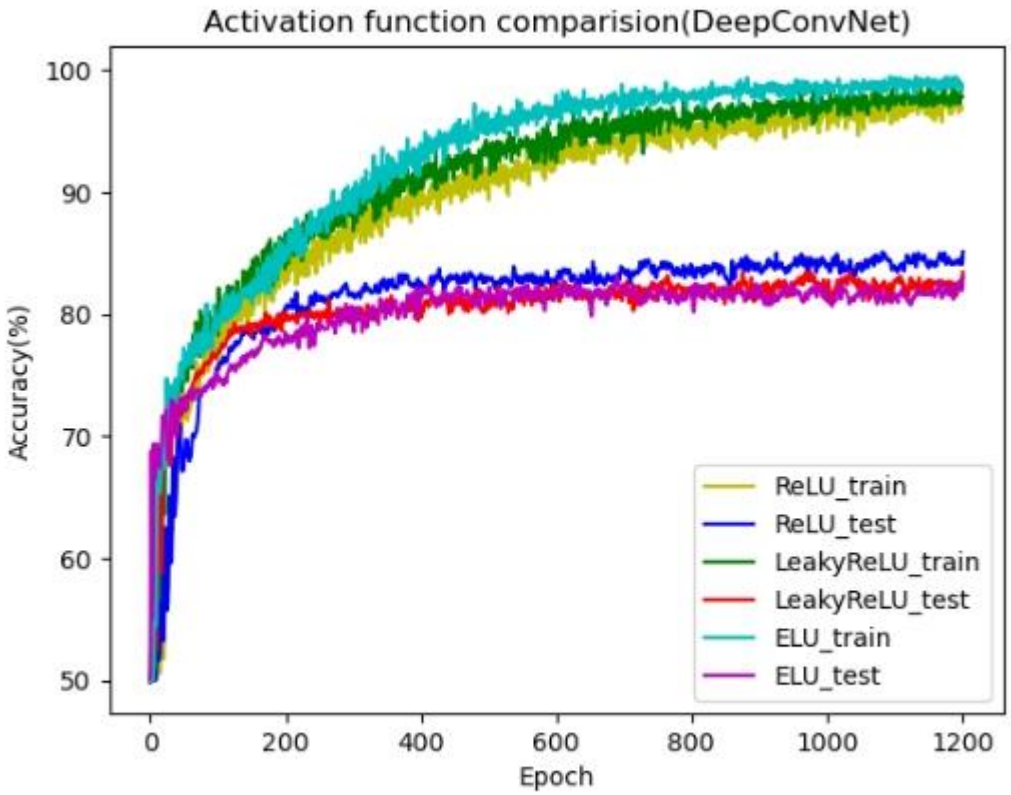
B. Comparison figures

- EEGNet



ReLU acc: 87.22%, LeakyReLU acc: 88.70%, ELU acc: 85.19%

- DeepConvNet




```
ReLU acc: 85.09%, LeakyReLU acc: 83.43%, ELU acc: 83.06%
```

IV. Discussion

A. Anything you want to share

1. 一開始不太會使用 DataLoader()，查詢資料後才知道要把 data 放至 TensorDataset()，然後才能夠再設定 DataLoader()的 dataset 這個參數。此外，一開始我想說 DataLoader()的 batch_size 這個參數設越大越好(在 GPU VRAM 足夠的情況下)，因為可以整批放進去 model 訓練，加快訓練速度，但後來發現，如果設小一點，因為一次 epoch 中，更新參數的次數更多，反而讓訓練過程更穩定。除此之外，DataLoader()還有一個參數叫 num_workers，我一開始是設為我的 CPU 個數，即 14，但發現訓練速度反而變成超慢，後來查詢資料才發現 num_workers 設太高可能會導致資源競爭和調度開銷，從而導致訓練速度變慢。對於較小的 dataset，設 0 或 1 就好。
2. 在 feed forward 的過程中，data 和 model 都必須使用.to(device)才能放到 GPU 上運行。而且 label 必須是 torch.long 的 type。

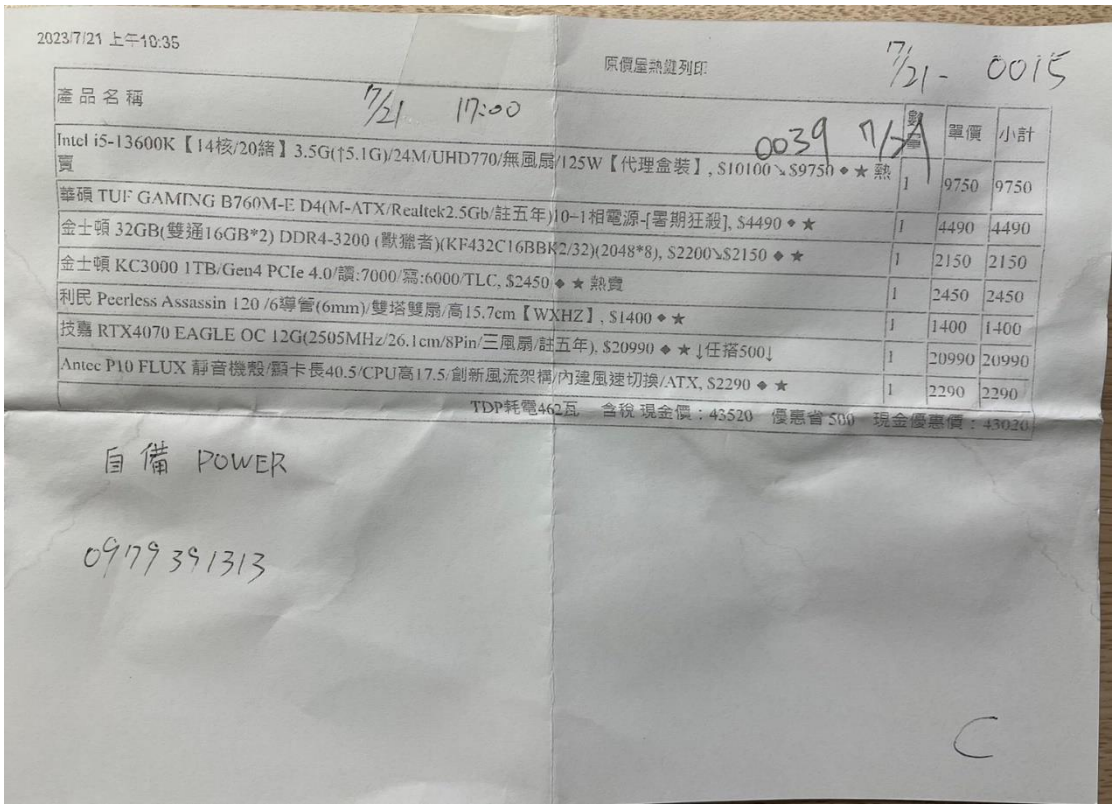
```
def forward(self, x):
    x = x.to(device)
    first_feature = self.firstConv(x)
    second_feature = self.depthwiseConv(first_feature)
    third_feature = self.separableConv(second_feature)
    flatten_feature = torch.flatten(third_feature, start_dim=1)
    pred = self.classify(flatten_feature)

    return pred
```

```
for i, (train_data, train_label) in enumerate(train_loader):
    train_data = train_data.to(device)
    train_label = train_label.to(device)
```

3. 一開始還沒存 model，所以我把 train 跟 test 寫在同一個 function，但後來要存 model 時發現這樣如果 load 回 model，再跑一次這個 function 的話，等於又再經過一次 train 也就是參數會被多更新一次，所以把 train 跟 test 拆分出來。換言之，存好 model 後，重新 load 並再跑一次 test 就會是 train 出來的最好結果。
4. 畫圖的時候發現我存的 acc 都是 single-element tensors，必須利用.item()這個 function 轉成 Python scalar values，所以放進 acc list 時，都必須先進行轉換。舉例: 原本算出來的 acc 可能是 tensor(5.)但經過.item()後會變成 5.0。
5. 我一開始覺得 epoch 越大，理論上 loss 會越低，acc 會越高，但實則不然，因為有時候可能會在最低點來回震盪。所以稍微測了 epoch=1000-100000，最好實驗結果是大概設在 1200 左右會比較理想。

6. 我一開始是用筆電 train 的，費了一番工夫把 cuda，anaconda，pytorch 裝好後，發現 train model 實在太慢了，所以我就買了一台桌機，搞了一張 4070(原本筆電是 1060)，後來發現真的快很多 XD(廢話)，以下是我去原價屋的菜單。然後買了主機，沒有螢幕好像也不行，碰巧看到原價屋有特價一台 Acer 的螢幕，2k 240Hz 才 7k，也手刀下單，用了 60Hz 好幾年的我一看到這畫面細膩度真的是超爽的。



2023/7/21 上午10:35 原價屋熱賣列印 7/21 - 0015

產品名稱 7/21 17:00 0039 7/21	數量	單價	小計
Intel i5-13600K【14核/20緒】3.5G(↑5.1G)/24M/UHD770/無風扇/125W【代理盒裝】, $10100↘$9750 ◆★熱賣	1	9750	9750
華碩 TUF GAMING B760M-E D4(M-ATX/Realtek2.5Gb/註五年)/0-1相電源-[暑期狂殺], $4490 ◆★	1	4490	4490
金士頓 32GB(雙通16GB*2) DDR4-3200 (獸獵者)(KF432C16BBK2/32)(2048*8), $2200↘$2150 ◆★	1	2150	2150
金士頓 KC3000 1TB/Gen4 PCIe 4.0/讀:7000/寫:6000/TLC, $2450 ◆★熱賣	1	2450	2450
利民 Peerless Assassin 120 /6導管(6mm)/雙塔雙扇/高15.7cm【WXHZ】, $1400 ◆★	1	1400	1400
技嘉 RTX4070 EAGLE OC 12G(2505MHz/26.1cm/8Pin/三風扇/註五年), $20990 ◆★↓任搭500↓	1	20990	20990
Antec P10 FLUX 靜音機殼/顯卡長40.5/CPU高17.5/創新風流架構/內建風速切換/ATX, $2290 ◆★	1	2290	2290

TDP耗電462瓦 含稅 現金價：43520 優惠省500 現金優惠價：43020

自備 POWER

0917 391313

C