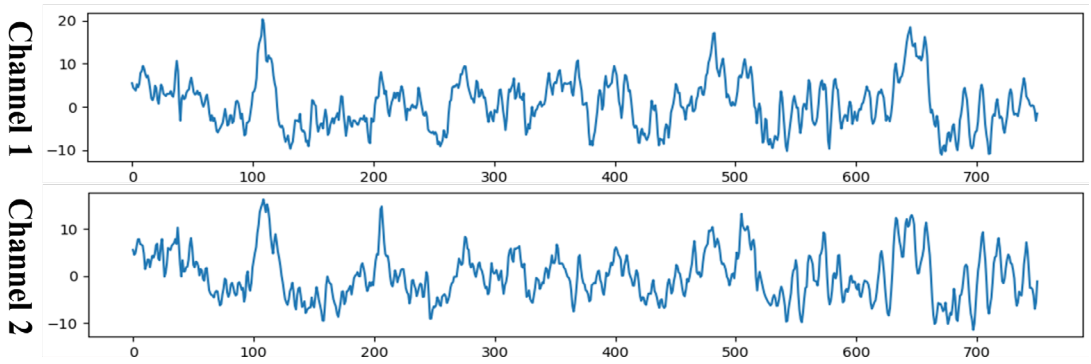
# Lab2 : EEG classifications

311511043 李承翰

1. Introduction

在這次 Lab，需要我們使用 pytorch 去實作 EEGNet 還有 DeepConvNet 兩種分類模型，用來分類 BCI 資料集，而這個資料集會有兩個 channel，分別代表著左手與右手，而總共會分別有 1080 筆的訓練資料及測試資料



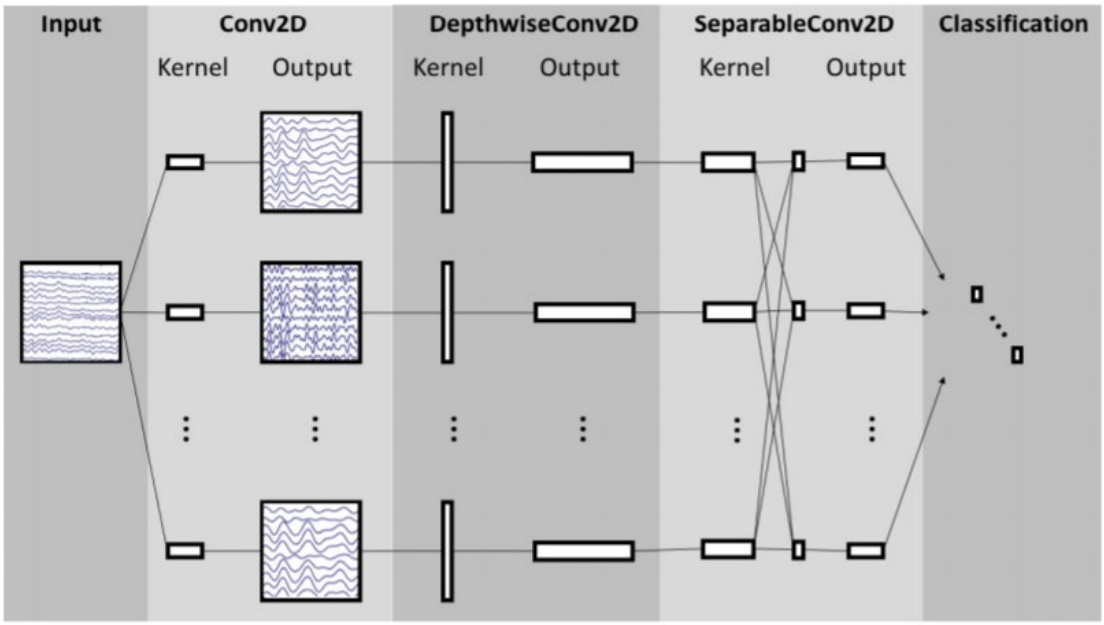

上圖為資料範例

此外，我們需要分別嘗試三種的 activation function(ReLU, Leaky_ReLU, ELU)，並且探討、比較實驗結果。

2. Experiment setup

A. The detail of models:

- EEGNet




EEGNet 是一個小型的卷積神經網路，常用於腦電波類的資料，因為大幅降低了模型當中的參數數量，所以可以在有限的硬體當中還是有不錯的訓練速度。

而整個模型的架構如上圖所示，大致上可以分成三個 convolutional layer，其設定如下:

```
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)
  )
)
```

第一層的 FistConv layer 負責的是將 input 的資料以頻率的方式分成 16 個 channel

第二層的 Depthwise layer 負責將第一層經過 frequency filter 的各個通道再經過 spatial filter，而因為我們是直接將第一層的 output 分別當作第二層的 input，讓使用上減少了參數數量，不過後果就是我們沒有辦法獲取不同頻率(通道)下同個空間的資料

而最後一層則是總結了第二層過來的資料，彌補了第二層的時候看不見不同頻率下相同空間的資料，將資料特徵都完整取出，交由下一層的 linear classification 來做判斷。

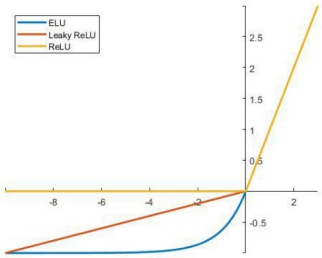
- DeepConvNet

DeepConvNet 是傳統的深度學習架構，由一層捲積層，在後面再接上 convolution、batch normalization、activation function、pooling、dropout，以這樣的順序重複 4 次，最後再用一個 fully connected 配合 softmax 進行分類，而每一層的參數設定如下圖: 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 * 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 |

B. Explain the activation function:

激勵函數的存在是為了讓神經網路模型可以判斷非線性的資料，如果沒有加上激勵函數，那麼整個神經網路的模型在判斷資料時就會出現困難。而這次我們分別要使用 ReLU,Leaky_ReLU,ELU 三個激勵函數




- ReLU

$$f(x) = \max(0,x),\ f'(x) = \begin{cases} 1, & (x \geq 0) \\ 0, & (x < 0) \end{cases}$$

ReLU 函數是現在常用的激勵函數，因為他運算速度快，而且不會有梯度消失的問題存在。不過因為她在 x < 0 的時候梯度為 0，因此在某些情況下面有可能完全不激發，而這種問題被稱為 dead ReLU 問題。

- Leaky_ReLU

$$f(x) = \max(0,x) + \min(0.001x,0)\ ,\ f'(x) = \begin{cases} 1, & (x \geq 0) \\ 0.001, & (x < 0) \end{cases}$$

Leaky_ReLU 對 ReLU 函數進行了改良，針對 x < 0 的部分提供了解決方案，在本來梯度為零的部分增加一個 0.001 的項，讓模型可能未被激活的情況得到改善，不過 Leaky_ReLU 依然有在 x = 0 的點無法微分的情況。

- ELU

$$f(x) = \max(0,x) + \min(\alpha * (exp(x) - 1),0),\ f'(x) = \begin{cases} 1, & (x \geq 0) \\ f(x) + \alpha, & (x < 0) \end{cases}$$

ELU 也是 ReLU 函數的改良版本，他解決了上述兩種函數遇到的各種問題，但是因為在函數運算當中牽涉了指數函數的運算，因此有了運算量較大的問題。

3. Experimental results

A. The highest testing accuracy:

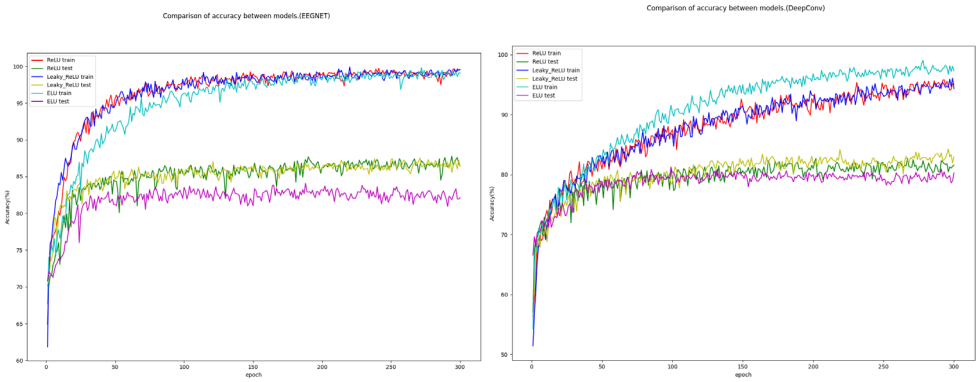
表格中與截圖數據有些微不同，因為是不同次實驗當中擷取的

| | ReLU | Leaky_ReLU | ELU |
|---|---|---|---|
| EEGNet | 88.98% | 89.07% | 83.61% |
| DeepConvNet | 81.11% | 81.38% | 80.46% |

```
now training network: EEGNET
now using activation fcn type: relu
{'now epoch: ': 50, 'loss: ': 0.2593009471893310S, 'training accuracy: ': 0.9101851851851852, 'testing accuracy: ': 0.8342592592592593}
{'now epoch: ': 100, 'loss: ': 0.2857321798801422, 'training accuracy: ': 0.9268518518518518, 'testing accuracy: ': 0.8611111111111112}
{'now epoch: ': 150, 'loss: ': 0.13416670262813568, 'training accuracy: ': 0.950925925925926, 'testing accuracy: ': 0.862037037037037}
{'now epoch: ': 200, 'loss: ': 0.060605939477682114, 'training accuracy: ': 0.9592592592592593, 'testing accuracy: ': 0.8657407407407407}
{'now epoch: ': 250, 'loss: ': 0.021002808585762978, 'training accuracy: ': 0.9694444444444444, 'testing accuracy: ': 0.8564814814814815}
{'now epoch: ': 300, 'loss: ': 0.2023935467004776, 'training accuracy: ': 0.9694444444444444, 'testing accuracy: ': 0.8666666666666667}
Best epoch: 203
Best_acc: 0.871296296296963
now using activation fcn type: leaky_relu
{'now epoch: ': 50, 'loss: ': 0.25554978647503666, 'training accuracy: ': 0.9138888888888889, 'testing accuracy: ': 0.8453703703703703}
{'now epoch: ': 100, 'loss: ': 0.17239254713058472, 'training accuracy: ': 0.937037037037037, 'testing accuracy: ': 0.862037037037037}
{'now epoch: ': 150, 'loss: ': 0.30566126108169556, 'training accuracy: ': 0.9564814814814815, 'testing accuracy: ': 0.8629629629629629}
{'now epoch: ': 200, 'loss: ': 0.038130883121745094, 'training accuracy: ': 0.9629629629629629, 'testing accuracy: ': 0.8740740740740741}
{'now epoch: ': 250, 'loss: ': 0.07777546858136177, 'training accuracy: ': 0.9675925925925926, 'testing accuracy: ': 0.8638888888888889}
{'now epoch: ': 300, 'loss: ': 0.022262761369347572, 'training accuracy: ': 0.9814814814814815, 'testing accuracy: ': 0.8638888888888889}
Best epoch: 287
Best_acc: 0.8777777777777778
now using activation fcn type: elu
{'now epoch: ': 50, 'loss: ': 0.09983541816473007, 'training accuracy: ': 0.8833333333333333, 'testing accuracy: ': 0.825}
{'now epoch: ': 100, 'loss: ': 0.05296963453292847, 'training accuracy: ': 0.9296296296296296, 'testing accuracy: ': 0.8416666666666667}
{'now epoch: ': 150, 'loss: ': 0.06384535878896713, 'training accuracy: ': 0.9490740740740741, 'testing accuracy: ': 0.8416666666666667}
{'now epoch: ': 200, 'loss: ': 0.13001899228858948, 'training accuracy: ': 0.9574074074074074, 'testing accuracy: ': 0.8435185185185186}
{'now epoch: ': 250, 'loss: ': 0.13234852254390717, 'training accuracy: ': 0.9648148148148148, 'testing accuracy: ': 0.8342592592592593}
{'now epoch: ': 300, 'loss: ': 0.04849542677402496, 'training accuracy: ': 0.9805555555555555, 'testing accuracy: ': 0.8444444444444444}
Best epoch: 193
Best_acc: 0.850925925925926
```

```
now training network: DeepConv
now using activation fcn type: relu
{'now epoch: ': 50, 'loss: ': 0.7612078785896301, 'training accuracy: ': 0.7796296296296297, 'testing accuracy: ': 0.7611111111111111}
{'now epoch: ': 100, 'loss: ': 0.3593970835208893, 'training accuracy: ': 0.8305555555555556, 'testing accuracy: ': 0.7703703703703704}
{'now epoch: ': 150, 'loss: ': 0.369075208902359, 'training accuracy: ': 0.8435185185185186, 'testing accuracy: ': 0.8}
{'now epoch: ': 200, 'loss: ': 0.23085010051722795, 'training accuracy: ': 0.8870370370370371, 'testing accuracy: ': 0.8055555555555556}
{'now epoch: ': 250, 'loss: ': 0.30168393254288099, 'training accuracy: ': 0.8916666666666667, 'testing accuracy: ': 0.7842592592592592}
{'now epoch: ': 300, 'loss: ': 0.2923013865947234, 'training accuracy: ': 0.9083333333333333, 'testing accuracy: ': 0.8064814814814815}
Best epoch: 233
Best_acc: 0.8111111111111111
now using activation fcn type: leaky_relu
{'now epoch: ': 50, 'loss: ': 0.6943282748500824, 'training accuracy: ': 0.7490740740740741, 'testing accuracy: ': 0.7712962962962963}
{'now epoch: ': 100, 'loss: ': 0.5116204619407654, 'training accuracy: ': 0.8398148148148148, 'testing accuracy: ': 0.7657407407407407}
{'now epoch: ': 150, 'loss: ': 0.41861259937286377, 'training accuracy: ': 0.8592592592592593, 'testing accuracy: ': 0.8018518518518518}
{'now epoch: ': 200, 'loss: ': 0.22823567688465118, 'training accuracy: ': 0.8824074074074074, 'testing accuracy: ': 0.7833333333333333}
{'now epoch: ': 250, 'loss: ': 0.42626525516360505, 'training accuracy: ': 0.8861111111111111, 'testing accuracy: ': 0.8009259259259259}
{'now epoch: ': 300, 'loss: ': 0.3327280282974243, 'training accuracy: ': 0.9083333333333333, 'testing accuracy: ': 0.7879629629629963}
Best epoch: 230
Best_acc: 0.8138888888888889
now using activation fcn type: elu
{'now epoch: ': 50, 'loss: ': 0.6062850952148438, 'training accuracy: ': 0.7907407407407407, 'testing accuracy: ': 0.7537037037037037}
{'now epoch: ': 100, 'loss: ': 0.3347131013870239, 'training accuracy: ': 0.825925925925926, 'testing accuracy: ': 0.7666666666666667}
{'now epoch: ': 150, 'loss: ': 0.22238485361060333, 'training accuracy: ': 0.8287037037037037, 'testing accuracy: ': 0.7916666666666666}
{'now epoch: ': 200, 'loss: ': 0.42508581280708313, 'training accuracy: ': 0.8768518518518519, 'testing accuracy: ': 0.7916666666666666}
{'now epoch: ': 250, 'loss: ': 0.19181646406650543, 'training accuracy: ': 0.8972222222222223, 'testing accuracy: ': 0.7842592592592592}
{'now epoch: ': 300, 'loss: ': 0.23682548105716705, 'training accuracy: ': 0.9361111111111111, 'testing accuracy: ': 0.7851851851851852}
Best epoch: 189
Best_acc: 0.8046296296296296
```

B. Comparison figures




4. Discussion

這一次的實驗當中有幾個主要的可以供我們調整的超參數:

1. Learning_rate
2. Momentum
3. Epoch
4. Dropout rate
5. Batch size

雖然在模型當中還有很多我們可以調整的超參數，但我主要有修改到的超參數就是這幾個，而 learning rate、momentum、epoch 會有什麼樣的影響在上一次 lab 當中都有提及，這次我新學到了 dropout rate，以及 batch size。

在這次的模型當中我們在許多地方加了一層 dropout layer，這一層相當的神奇，她會以機率 = dropout rate 的方式，在訓練過程當中適當的使一些神經元不被觸發，以這樣的方式來防止 overfit。而在 EEGNet 這種比較沒那麼深的模型當中，跟在 DeepConvNet 這種深度的模型比起來，兩者就需

要不一樣的 dropout rate，在 DeepConvNet 當中需要較大的 dropout rate 才可以有效地防止 overfit。

而 batch size 的部分，雖然在 lab1 當中有學到，但當初的資料我並沒有特別針對 batch size 進行調整。這一個參數代表著一個 epoch 當中，要分幾次將所有的訓練資料丟到模型當中去訓練。如果將 batch size 調大，那麼因為矩陣的乘法次數會減少，所以整體來說運算效率也會提升，但因為在進行 gradient 運算的時候，將各個訓練資料所產生的 gradient 進行了平均的動作，因此每一次 error 造成的 gradient 特徵會在這一個過程當中被犧牲，在模型學習上會相對而言不直觀。因為理論上我們對每一個權重的修正量，應該是看此次 error 的 gradient，但這個一平均下去就會稍微變得不同。所以在學習效率上我們需要稍微取捨。