Introduction:

這次作業要對 EEG (Electroencephalography)圖進行二元分類，腦波圖（EEG）是一種腦電波的紀錄方法，放置電極於頭皮記錄腦神經元的電壓變化。而相同動作或想法，可能會有相似的 pattern，因此如果可以依據腦波圖來分析受試者可能的相應動作，便可以依此做出應對與回饋。此作業便是要利用 CNN 網路，分析其中的 pattern 進而再做出分類。

而且 CNN (convolutional neural network、卷積神經網路)，在深度學習中，擅長於處理圖片的資訊，層與層之間會有 filter 提取相關特徵，filter 會是一個矩形，通常會是正方形，但也可以根據你對問題的了解（也就是所謂的 domain knowledge），自行定義認為有助於的提取特徵的形狀。

而 filter 向右或向下移動的步伐稱為 stride，而全部的 filter 掃出來的結果即為下一層的 layer。此外如果 filter 超出 layer 的長與寬可以利用 padding 決定是否要填補。除了有一般的卷積層其中可以還可以安插 batch normalization 將參數調整以致於易於訓練。也可以加入激活函數，或是 pool 層對資料做處理。重複疊加幾層，最後再連結全連接層，轉乘所需要的 class 數量，就可以利用 CNN + fully connected 處理圖片分類的問題。

Experiment set up:

EEGNet:

```
EGGNet_ELU(
  (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, inplace=False)
  )
  (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, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

圖一、EEG 模型架構，

```
class EGGNet_ELU(nn.Module):
  def __init__(self):
    super( EGGNet_ELU, self).__init__()
    self.firstconv = nn.Sequential(
        nn.Conv2d(1, 16, kernel_size=(1,51), stride=1, padding=(0,25) , bias=False),
        nn.BatchNorm2d(16)
    )
    self.depthwiseConv = torch.nn.Sequential(
        nn.Conv2d(16, 32, kernel_size=(2,1), stride=1, groups=16, bias=False),
        nn.BatchNorm2d(32),
        nn.ELU(),
        nn.AvgPool2d(kernel_size=(1,4), stride=(1,4), padding=0),
        nn.Dropout(p=0.25)
    )
    self.separableConv = torch.nn.Sequential(
        nn.Conv2d(32, 32, kernel_size=(1,15), stride=1, padding=(0,7), bias=False),
        nn.BatchNorm2d(32),
        nn.ELU(),
        nn.AvgPool2d(kernel_size=(1,8), stride=(1,8), padding=0),
        nn.Dropout(p=0.25)
    )
    self.classify = torch.nn.Sequential(
        nn.Linear(736, 2, bias=True)
    )

  def forward(self, x):
    x = self.firstconv(x)
    x = self.depthwiseConv(x)
    x = self.separableConv(x)
    x = x.flatten(1)
    x = self.classify(x)
    return x
```

圖二、EEG 實作細節。

EEG 網路主要可以看成四層，第一層只有簡單的卷積加上 batchnorm，第二、三層則是多加上激活函數、pooling、Dropout，最後則接上 linear fully connected。最後定義整個網路的 forward，整個網路就算是定義好了。在這邊直得注意的是，kernel size 也就是所謂的 filter 不是正方形的而是 1 by 51 或是 2 by 1，這種設計即是對問題也就是 EEG 腦波圖有更深的了解，這樣的觀察認為有助於模型能夠提取有用的特徵。

DeepConvNet:

```
DeepConvNet_ELU(
  (firstConv): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=(0, 25), bias=False)
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), padding=(0, 25), bias=False)
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ELU(alpha=1.0)
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (secondConv): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (thirdConv): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (fourthConv): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1))
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=9800, out_features=2, bias=True)
  )
)
```

圖三、DeepConvNet 模型架構。

```
class DeepConvNet_Relu(nn.Module):
  def __init__(self):
    super( DeepConvNet_Relu, self).__init__()
    self.firstConv = nn.Sequential(
        nn.Conv2d(1, 25, kernel_size=(1,5), stride=1, padding=(0,25) , bias=False),
        nn.Conv2d(25, 25, kernel_size=(2,1), stride=1, padding=(0,25) , bias=False),
        nn.BatchNorm2d(25),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(1,2)),
        nn.Dropout(p=0.5)
    )
    self.secondConv = nn.Sequential(
        nn.Conv2d(25, 50, kernel_size=(1,5)),
        nn.BatchNorm2d(50),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(1,2)),
        nn.Dropout(p=0.5)
    )
    self.thirdConv = nn.Sequential(
        nn.Conv2d(50, 100, kernel_size=(1,5)),
        nn.BatchNorm2d(100),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(1,2)),
        nn.Dropout(p=0.5)
    )
    self.fourthConv = nn.Sequential(
        nn.Conv2d(100, 200, kernel_size=(1,5)),
        nn.BatchNorm2d(200),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(1,2)),
        nn.Dropout(p=0.5)
    )
    self.classify = nn.Sequential(
        nn.Linear(9800, 2, bias=True)
    )

  def forward(self, x):
    x = self.firstConv(x)
    x = self.secondConv(x)
    x = self.thirdConv(x)
    x = self.fourthConv(x)
    x = x.flatten(1)
    x = self.classify(x)
    return x
```

圖四、DeepConvNet 實作細節。

而在 DeepConvNet 中，前四層基本上架構相同，而且也沒有什麼特別的地方，沒有 domain knowledge 的觀察，看起來就是利用 Deep CNN 單純的解決問題。

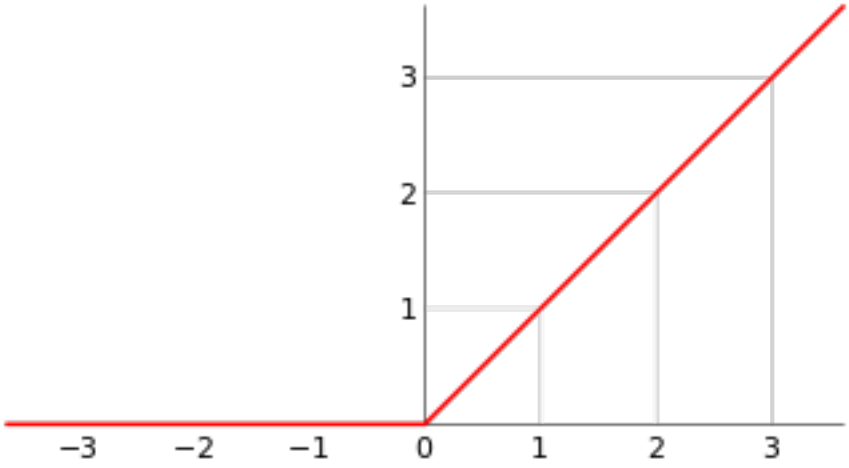
Explain the activation function (ReLU , Leaky ReLU , ELU) :

ReLU :

ReLU 作為 DL 中最基本的也做常見的激活函數其公式及圖如下分別為圖五及圖六。

$$f(x) = \max(0, x)$$

圖五、ReLU 公式。



圖六。

在輸入大於零時，及輸出完整的輸出。小於零時，輸出為零。這樣最大的好處是其斜率為 1 很好計算。其函數相較 sigmoid 也更符合生物神經網路的表現，有一定的生物學原理。而通常也有較好的表現。而且也有較好的 gradient back propagation 表現，較少發生 vanishing gradient （梯度消失）的情況。

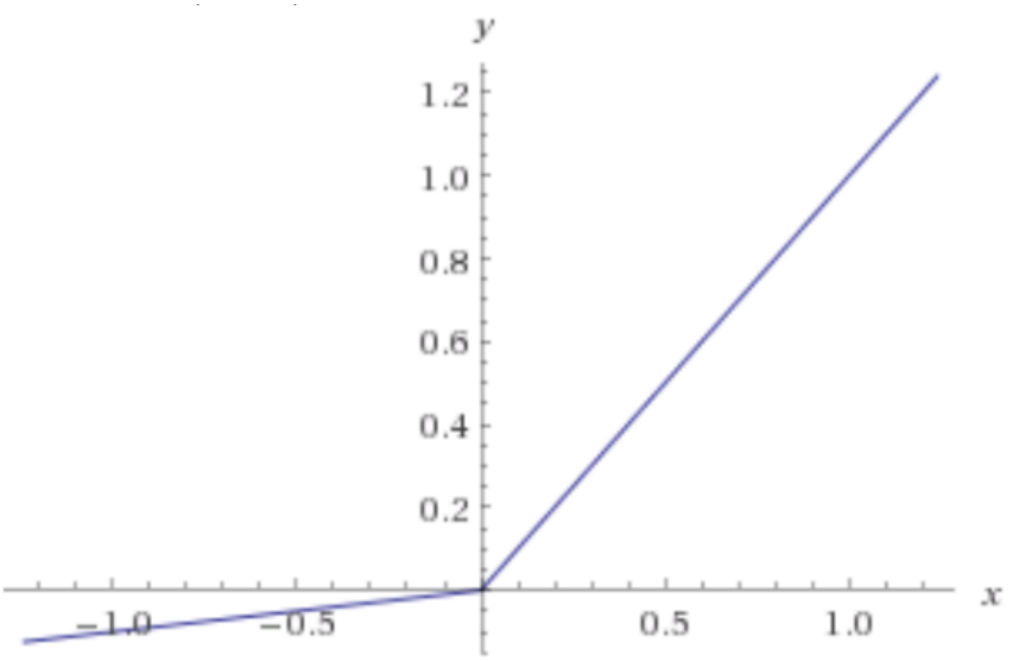
而 ReLU 也有眾多的變形，其主要變化大多在於零之前的，試圖解決當輸入小於零時，輸出為零梯度為零，不易於反向更新網路參數。例如 Leaky ReLU、Parametric ReLU、ELU 。

Leaky ReLU :

Leaky ReLU 作為 ReLU 的變形，其公式及圖分別為圖七、圖八。

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

圖七，Leaky ReLU 公式。



圖八。

Leaky ReLU 作為 ReLU 的改良，將小於零時輸出的定義，改為輸出相對於輸入較小的比例，這樣小小的修改就可以對左側的情況也就是小於零時，有較小的 gradient 而不為零。對於神經網路節點來說，此節點不再是沒有反應，而是也有少許的輸出。

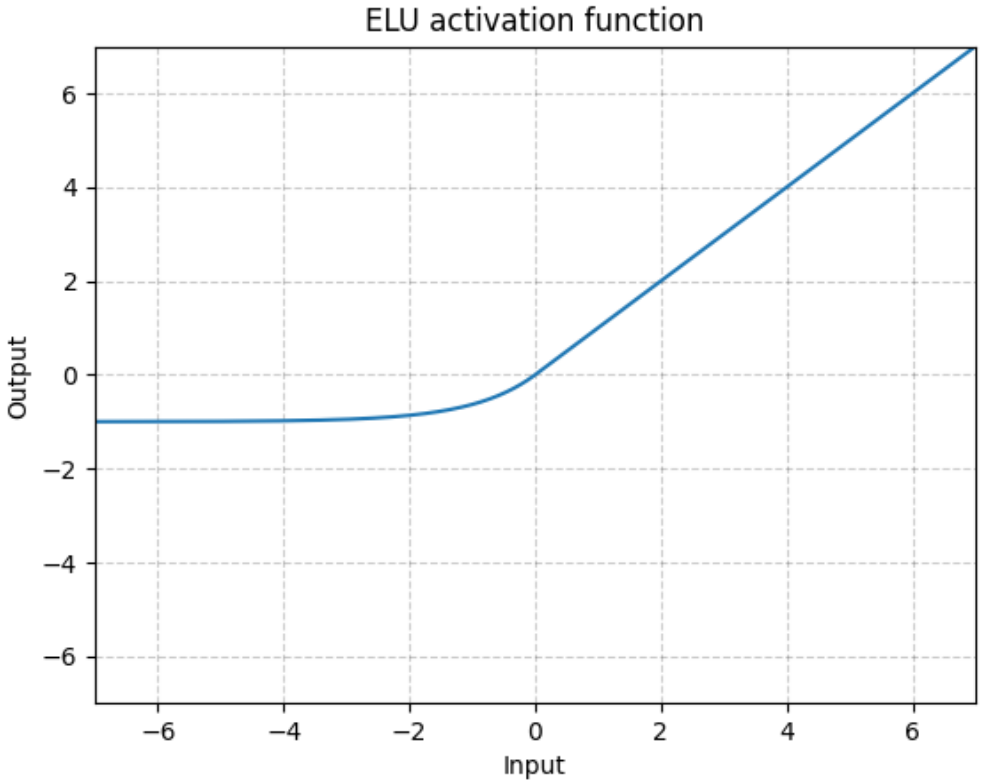
常用於可能會遭遇到 sparse gradients 的模型上，例如生成對抗網路 (Generative Adversarial Networks, GAN)。

ELU:

ELU 作為 ReLU 的變形，其公式及圖分別為圖九、圖十。

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

圖九、ELU 公式。



圖十。

Exponential linear units (ELU)，一樣作為 ReLU 的改型，也是對左側，也就是小於零時作修改。而其頃向將輸出收斂於零來產生更好的結果。而 ELU 有以下幾點優點

1. 當輸出為負時，輸出較平滑，而不是 ReLU 那麼尖銳。
2. 強於 ReLU
3. 相對於 ReLU，ELU 可以產生負值。

而 ELdU 也已被證明相較於 ReLU 來說 ELU 對分類問題（classification）有更好的表現。

Experimental results:

The highest testing accuracy:

DeepConvNet_Relu: test_set acc = 0.86175
epochs=300 optimizer=Adam lr=1e-3 batch_size=1024

DeepConvNet_LeakyReLU: test_set acc = 0.87116
epochs=300 optimizer=Adam lr=1e-3 batch_size=1024

DeepConvNet_ELU: test_set acc = 0.84535
epochs=300 optimizer=Adam lr=1e-3 batch_size=1024

EGGNet_ReLU: test_set acc = 0.89265
epochs=300 optimizer=Adam lr=1e-3 batch_size=1024

EGGNet_LeakyReLU: test_set acc = 0.87626
epochs=300 optimizer=Adam lr=1e-3 batch_size=1024

EGGNet_ELU: test_set acc = 0.87395
epochs=300 optimizer=Adam lr=1e-3 batch_size=1024

上述的結果皆為無數次嘗試的結果，其中有嘗試不同的 optimizer 分別為 Adam、SGDM、RSMprop，表現如同上述順序。

發現不同的 batch_size 所造成的影響：

Batch 較小時，更新較積極，且一整個 data set 中一次可以更新的次數較多。

Batch 較大時，更新較穩重，且一整個 data set 中一次可以更新的次數較少。

最後選擇較大的 Batch 作為訓練策略。

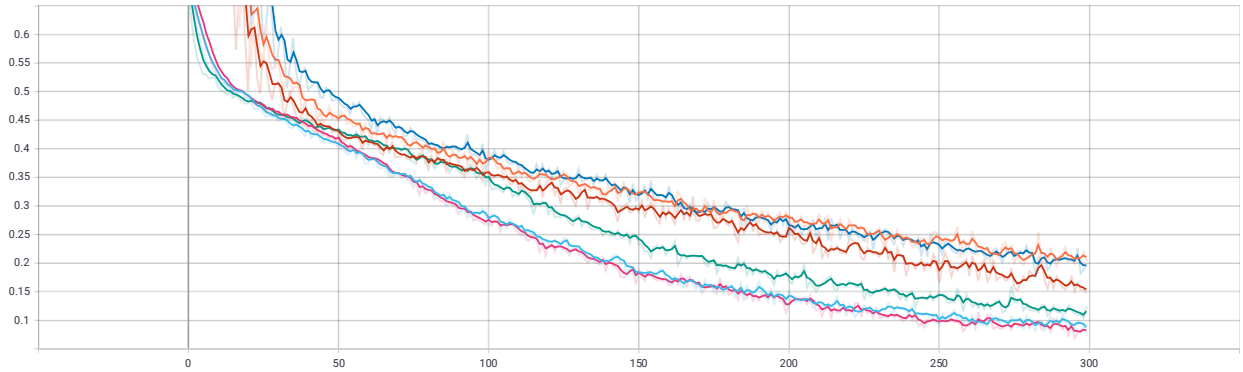
此問題在眾多嘗試中，任然無法將 test_acc 拉到接近 95%，因為我認為一方面為對此問題的了解不夠透徹，沒有關於 EEG 的 domain knowledge 。

給予用於訓練的資料也偏少再者因其 EEG 的資料特性也不適合做 data augmentation。倘若給予更多的資料甚至是沒有 labels 的也可以利用 semi supervised learning 對沒有 labels 的資料給予 pseudo labels 來增加訓練資料。

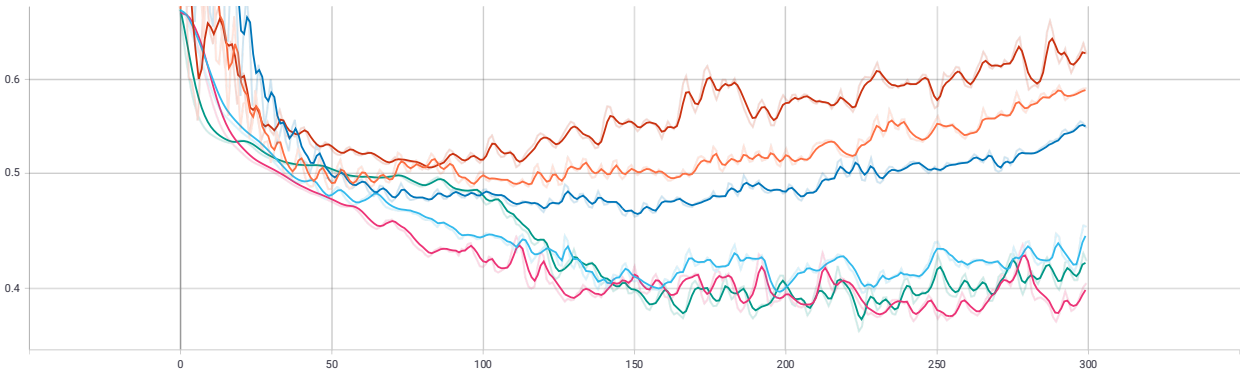
Comparison figures:

橘色為 DeepConvNet_Relu、
深藍為 DeepConvNet_LeakyReLU、
紅色為 DeepConvNet_ELU、
淡藍為 EGGNet_ReLU、
粉紅為 EGGNet_LeakyReLU、
墨綠為 EGGNet_ELU。

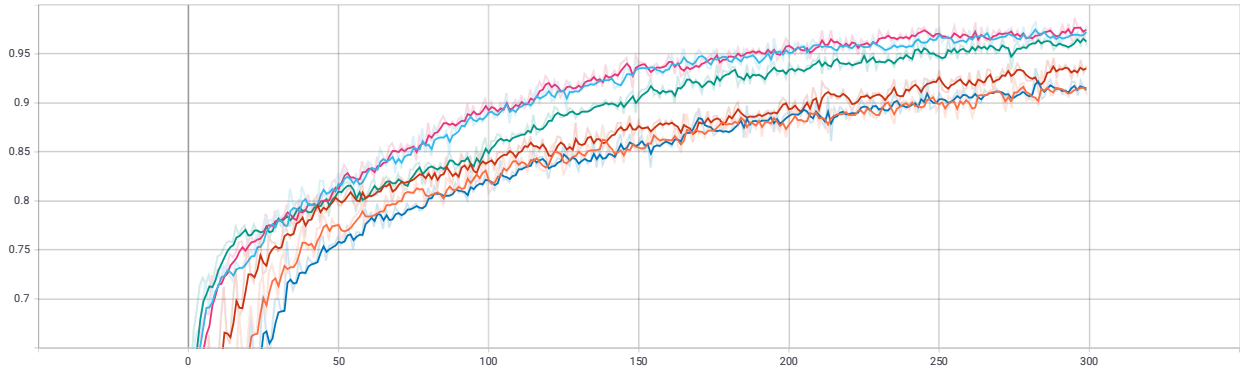
圖十一、Tensorboard 數據資料圖。



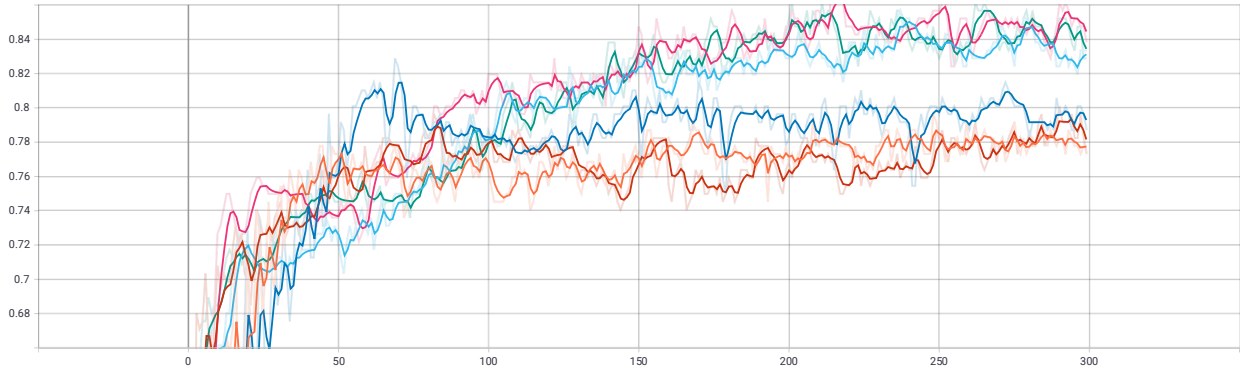
圖十二、loss_train。



圖十三、loss_vaild。



圖十四、accuracy_train。



圖十五、accuracy_valid。

由上述圖可以得知 EEGNet 的表現較 DeepConvNet 好，不管是在 loss 或是 accuracy 中都是。而且在 DeepConvNet 中越訓練 loss_valid 不降反升。猜測可能是 overfitting，以至於在 Valid 中表現不好。EEGNet 則是無法有效的收斂，loss 震盪幅度巨大。

Discussion:
Anything you want to share:

發現 tensorflow 中的資料分析工具中的 tensorboard 竟然可以用在 pytorch 上面，而且相當簡單不複雜。

Pytorch 相較於 tensorflow 真的相對簡單很多，學習成本低，Pytorch 的文件也清楚明瞭。

在定義 model 的 forward 時，如果途中 flow 有錯，pytorch 不會提出警告，而且也可以進行訓練，但內部可能真的沒有 flow，但進行 loss BP 時參數不會 update，模型表現很差，訓練不起來。

Colab 白嫖搶不到 GPU，所以我買了 Colab Pro 10.88 鎂。

可以將 Google Drive mount 在 colab 的 VM 上，之後所有資料都可以從 Google Drive 上存取。

Colab 上可以直接將 tensorboard 顯示出來，訓練 Model 時可以即時的觀看 loss_train、loss_valid 的表現。

台大李宏毅的課程 2021 Machine Learning 和其 Homeworks 對我的幫助很大，學到很多東西。