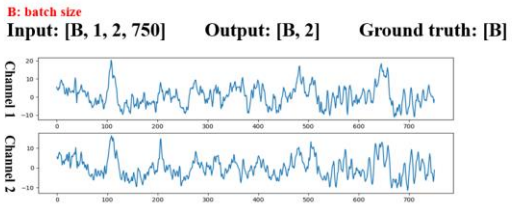
1. Introduction

在這次的 lab 中，利用 pytorch 實作了兩種 CNN 架構的模型: EEGNet 和 DeepConvNet，並將他們應用在運動想像的腦波訊號分類。同時，這次的 lab 中也嘗試了不同的 activation function：ReLU、Leaky ReLU 和 ELU，並觀察對分類準確率造成的影響。

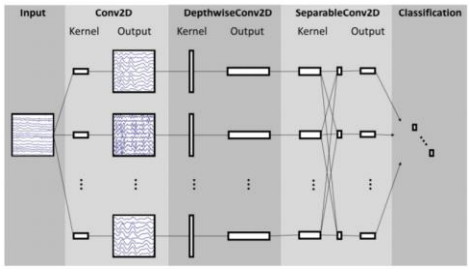
Training 和 testing dataset 來自 BCI Competition III – IIIb，是一個有兩個分類、兩個 channel 的電極訊號的運動想像腦波訊號。在經過助教提供的 data loader.py 做前處理並經 pytorch 的 dataloader 根據 batch size 分成一坨坨的訓練資料後，如下圖 input data 的 size: [B, 1, 2, 750]代表就代表[batch size, 固定為 1, 2 個電極 channel, 750 個腦波訊號的時間序列紀錄值]，這裡的 2 電極 channel*750 腦波訊號的時間序列紀錄值就像是一張 height=2, width=750 的單色 channel 的影像。(雖然圖片和電極都有 channel 這個詞，但它們代表的意思是不同的)



2. Experiment set up

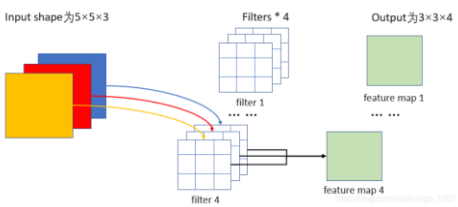
A. The detail of your model

◆ EEGNet

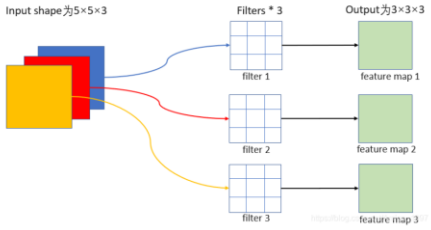



```
EEGNet(
  (firstconv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False)
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (depthwiseConv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 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)
  )
)
```

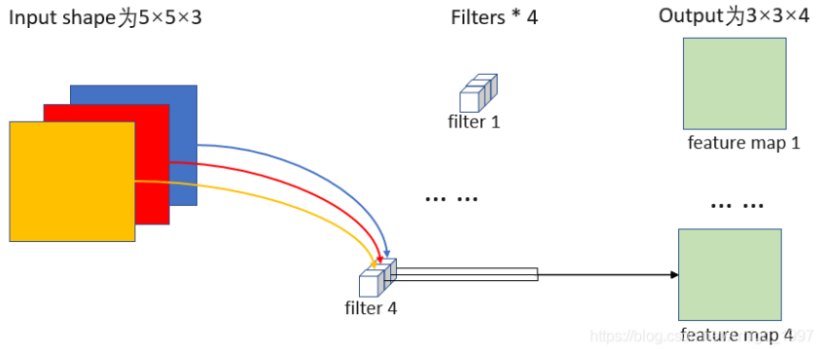
EEGNet 主要可以分成三個部分：傳統的 convolution layer、depthwise convolution layer 和 separable convolution layer。一般的傳統 convolution layer 如下圖所示，每一個 kernel(圖中指 filter)在做 convolution 時是一次對 input 的所有 channel 做，然後 output 出一張集合了 input image 的空間資訊和各個 channel 資訊的 feature map。(在 EEGNet 中，他做為 temporal convolution，學習不同的 frequency filter)



depthwise convolution layer 則如下圖所示，每一個 kernel(圖中指 filter)做 convolution 時只負責對 input 的單一個 channel 做，並產生出一張單 channel 的 feature map。比起普通的 convolution，這樣可以減少 kernel 需要學習的參數，並學習 frequency-specific spatial filter。



最後的 separable convolution layer 則首先依然是經過一個 depthwise convolution layer(可看 EEGNet 架構中的第一張圖，他學習的是每個 feature map 的 temporal summary)，接著再通過一個如下圖所示的 pointwise convolution，他會對影像的各個 channel 的對應位置做加權。這樣的方法可以使 output 的 feature map 數變多，並將從前面層輸入的各個 feature map 做混和(這兩點也就是 depthwise convolution 中沒做到的)。



◆ DeepConvNet

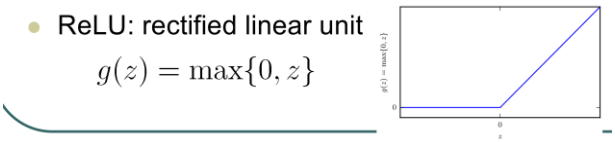
DeepConvNet 則如下圖所示，是由多個傳統 convolution layer 串起來的模型。

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 (ReLU, Leaky ReLU, ELU)

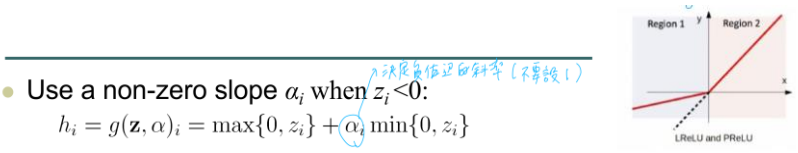
◆ ReLU

ReLU 的全名是 Rectified Linear Unit，通過這個 function 之後，所有小於 0(負值)的輸入都會變為 0，大於 0(正值)的輸入則保持不變輸出。缺點是，正數輸入的輸出可能無限大，負數改為 0 相當於產生負數的 neuron 計算出的 gradient 也等於 0。



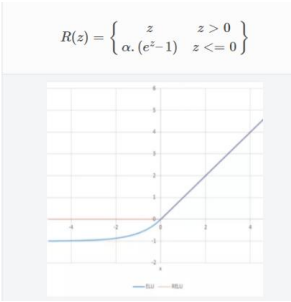
◆ Leaky ReLU

Leaky ReLU 則是不希望放棄負數，因此在負數的地方留下小小的斜率(alpha)，讓輸出仍接近 0 但不為 0。



◆ ELU

ReLU 和 Leaky ReLU 在 input=0 的點皆不可微分，ELU 則是整個函數都連續可微，並且在輸入為負時，也會產生小小的負值輸出。



3. Experimental results

A. The highest testing accuracy

◆ EEGNet

當時的 hyperparameter 設定為：

Batch size: 64

Learning rate: 0.001

Optimizer: Adam

Total training epochs: 300(但最高 testing accuracy 出現的時候並不一定是在最後一個 epoch，如同以下比較圖中顯示的，testing accuracy 是會有波動的)

Loss function: cross entropy

```
relu: 87.77777777777777
leaky_relu: 87.68518518518519
elu: 83.88888888888889
```

◆ DeepConvNet

Batch size: 64

Learning rate: 0.01

Optimizer: Adam

Total training epochs: 300(但最高 testing accuracy 出現的時候並不一定是在最後一個 epoch)

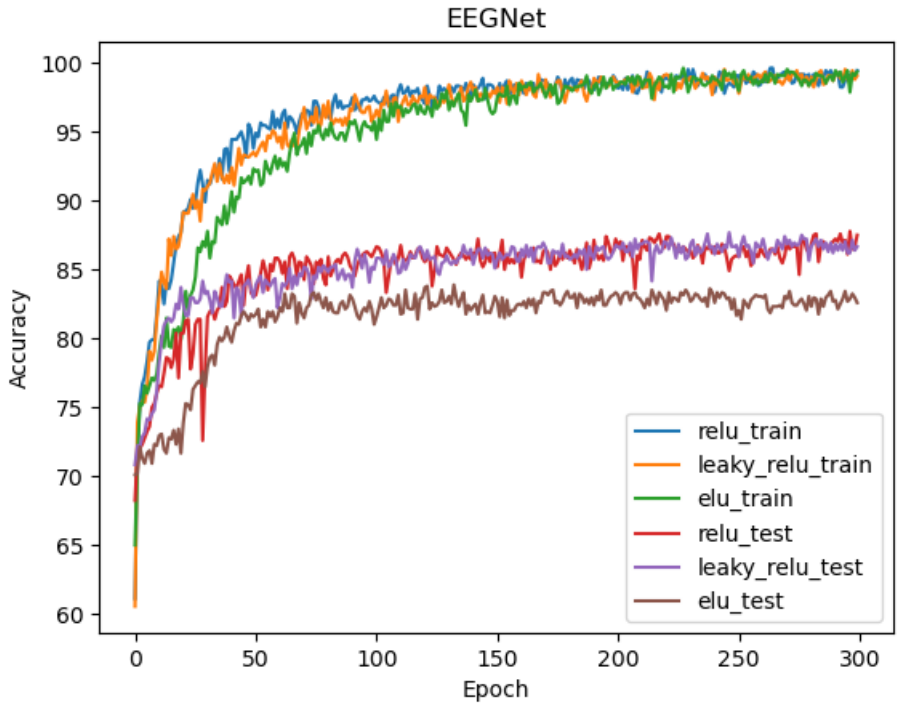
Loss function: cross entropy

```
relu: 75.27777777777777
leaky_relu: 77.87037037037037
elu: 78.70370370370371
```

B. Comparison figures

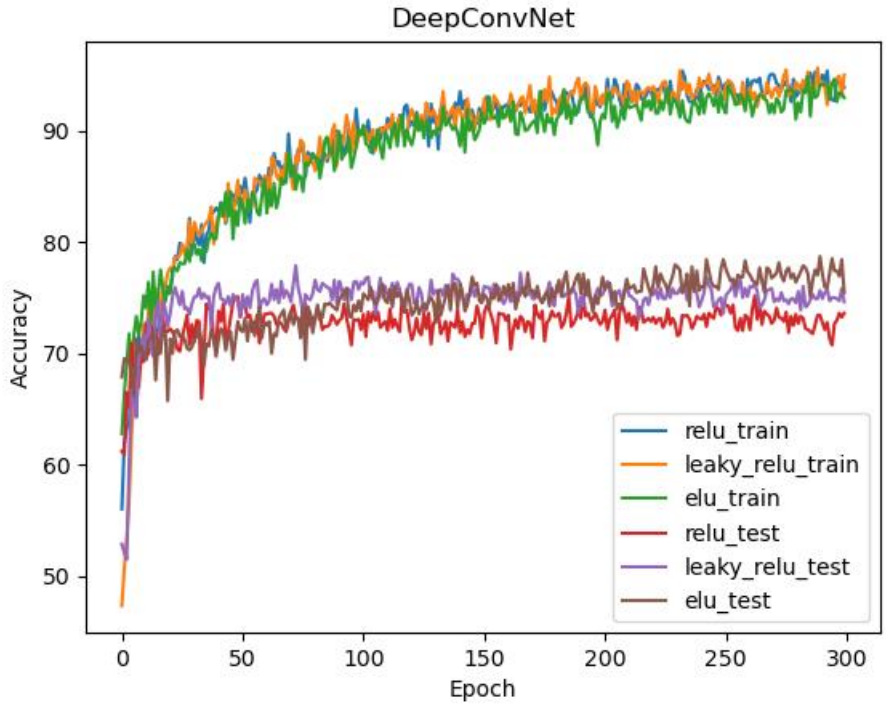
◆ EEGNet

下圖為 EEGNet 產生最高 testing accuracy 的比較圖。



◆ DeepConvNet

下圖為 DeepConvNet 產生最高 testing accuracy 的比較圖。



4. Discussion

A. Anything you want to share

我自己最一開始先試了幾種 learning rate 的組合，後來挑到了目前上面所選擇使用的 learning rate。我自己在那之後還試了幾種的 batch size，但對準確率好像都沒有什麼太大的影響，下圖是在 EEGNet 上嘗試過的幾種 batch size:

Batch size = 8:

```
relu: 84.25925925925925
leaky_relu: 82.5
elu: 80.0
```

Batch size = 32:

```
relu: 83.61111111111111
leaky_relu: 81.57407407407408
elu: 82.68518518518519
```

Batch size = 64:

```
relu: 85.18518518518519
leaky_relu: 82.87037037037037
elu: 81.29629629629629
```

Batch size = 128:

```
relu: 85.0
leaky_relu: 83.98148148148148
elu: 83.05555555555556
```

Batch size = 512:

```
relu: 84.35185185185185
leaky_relu: 84.16666666666667
elu: 84.07407407407408
```

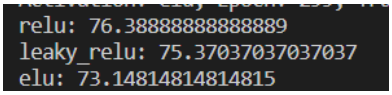
那時候準確率最高大約都是 83 多~85 出頭左右，而且用了非 spec 提供的

64 之外好像還比較低。原本有想說要調調看 optimizer 或是 model 的一些其他參數，但 google 了一下覺得 adam 其實好像感覺還不錯了，再加上覺得對 model 的一些其他參數做調整，感覺對我來說像是無頭蒼蠅一樣的在亂調，所以後來也就沒有嘗試。就決定回到 spec 上給的一些建議的參數(除了 learning rate 之外)。

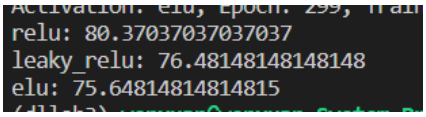
在這之後，我看到前幾年的 github 上有人用了和我幾乎一樣的參數，但卻能夠到大約 87 左右的準確率，我就很仔細的思考我們到底是差在哪裡，結果最後發現是 load dataset 進來的時候，可以選擇是不是要 shuffle data 的順序在變成 batch，我一開始也沒想那麼多，想說 spec 上也沒特別寫，乾脆就不用好了，後來讓它 shuffle 之後 EEGNet 的準確率就幾乎都可以大於 87 了。我覺得是因為因為原本不 shuffle 的話，可能就是依照收集的時間先後順序變成 batch，可能前收集和後收集的腦波又有一點差吧，所以都用前或是都用後的腦波計算出的 gradient 可能就沒有那麼可以往最好的地方走，因此影響了準確率。

在 DeepConvNet 也出現 shuffle 後準確率較高的結果。

Batch size = 64 (沒 shuffle)



Batch size = 64 (有 shuffle)



另外，比較各種 activation function 之後，其實普遍上 ReLU 的結果表現得最好的次數較多(如上圖這些我嘗試的各種 batch size 的結果)，難怪 ReLU 會是大家在訓練模型時幾乎普遍會使用的 activation function(如果沒有特殊的理由的話)。

Reference: https://blog.csdn.net/Amigo_1997/article/details/106329342