# Report Spec

## 1. Introduction

在 LAB2 中要將初步學掉的 CNN 用法透過 pytorch 學以致用，譬如 model 每層 Layer 的架設與初值設定，另外必須要充分了解到 convolution 的作用，是利用 Parameter sharing、Local connectivity、Equivalent representation ...等特性來達到 CNN 最大的效果。

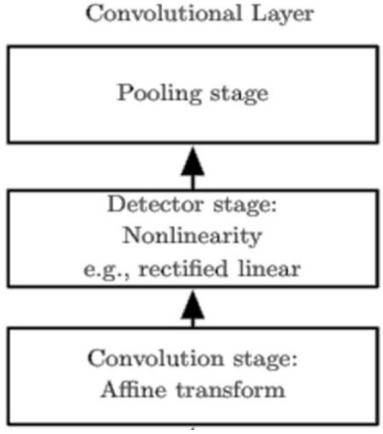
另外於本次實作中，也是更加熟悉了 pytorch 之使用，並且了解到 cuda 平行運算的強大，順帶一提一些基本的 debug 還是需要化解的。另外由於本次作業有根據 acc 的準確度來判定分數，所以從 accuracy 與 loss 中找到最佳的 hyperparmeter 設定，也是一大挑戰。

## 2. 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=0.5, 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=0.5, inplace=False)
    (5): Flatten()
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

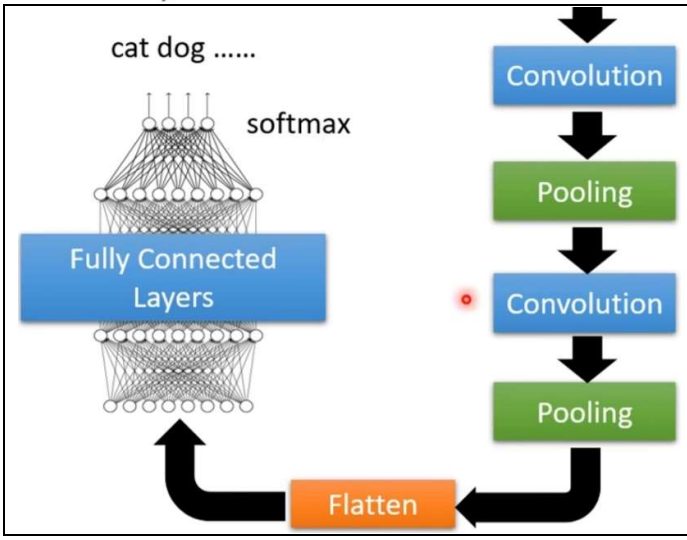



其實 Convolution 的架構不外乎就是這些

1. 利用 Convolution 的特性，將原本的 input 依照 kernel size 分成對應的 feature map。
2. 根據 filter(neurons)數量切割出不同的 channel，再 by channel 做 Normalization 為的是要使 gradient descent 更容易
3. 加上 activation function 為的是突破 linear 的限制
4. 丟入 Pooling 根據 kernel size 取最佳特徵值

- DeepConvNet

```
DeepConvNet(
  (conv1): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1))
  )
  (conv2): Sequential(
    (0): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (1): BatchNorm2d(25, 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)
  )
  (conv3): 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)
  )
  (conv4): 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)
  )
  (conv5): 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)
    (5): Flatten()
  )
  (classify): Sequential(
    (0): Linear(in_features=8600, out_features=2, bias=True)
  )
)
```




5. 最後利用 flatten() 打平 channel、與 softmax 等之類的 activation function，來總結一個推測的結果出來。
6. 補充 1： 另外 stride, padding 等應用在 conv 上的作用這邊簡短提個大概，就是 kernel size 移動的距離、與空位補 0 的意思。
7. 還有 dropout 能降低 overfit，另外有趣的是 pooling 其實不是必要，在某些講求精度的 model 其實是做 pooling 會降低 performance 的。(full convonlution)

兩個 CNN 經過每層 layer 後的 shape

EEGNet

```
torch.Size([128, 1, 2, 750])
torch.Size([128, 16, 2, 750])
torch.Size([128, 32, 1, 187])
torch.Size([128, 736])
torch.Size([128, 2])
torch.Size([128, 2])
```

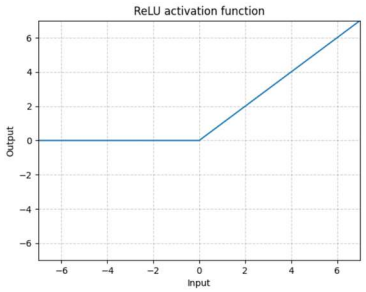
DeepConvNet

```
torch.Size([128, 1, 2, 750])
torch.Size([128, 25, 2, 746])
torch.Size([128, 25, 1, 373])
torch.Size([128, 50, 1, 184])
torch.Size([128, 100, 1, 90])
torch.Size([128, 8600])
torch.Size([128, 2])
torch.Size([128, 2])
```

## B. Explain the activation function (ReLU, Leaky ReLU, ELU)

- **ReLU**

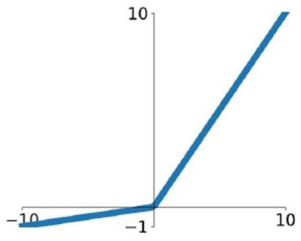
$$f(x) = \max(0, x)$$




Relu 為 nonlinear function，他非常簡單去設定，不會梯度消失問題。
但也有個重大的缺點；非常脆弱，當 x < 0 時 梯度為 0。導致不對任何值有反應。(非梯度消失問題)

- **Leaky ReLU**

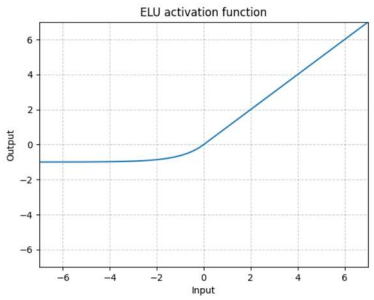
$$f(x) = \max(0.01x, x)$$




為了解決 dead relu 現象 使用了一種固定斜率的方法使得 x<0 時梯度也不會變為 0

- **ELU**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

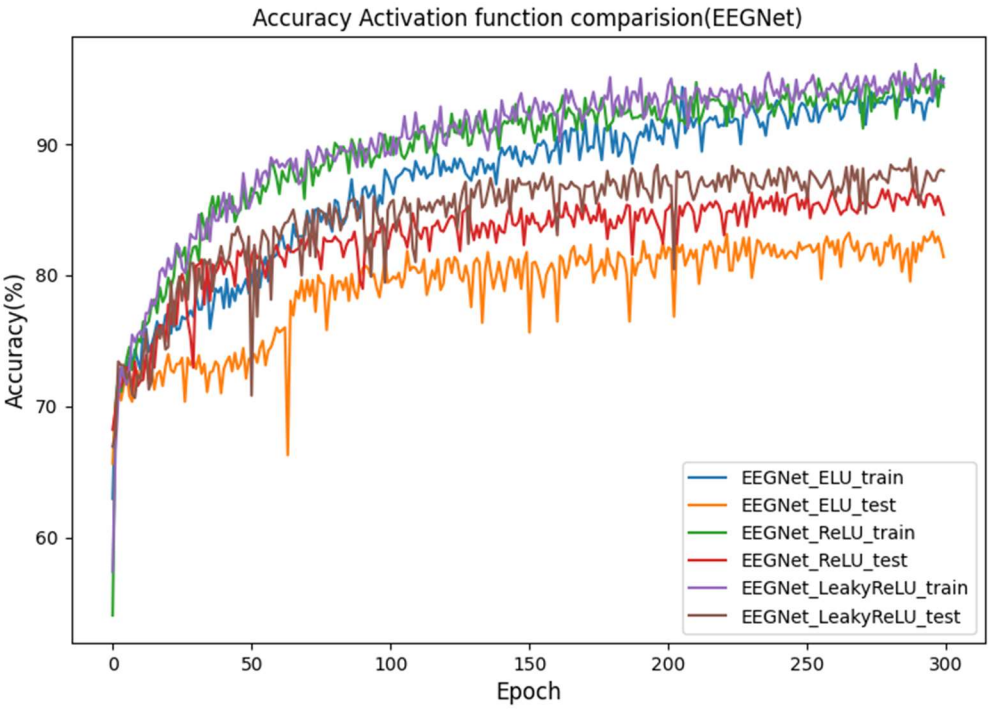


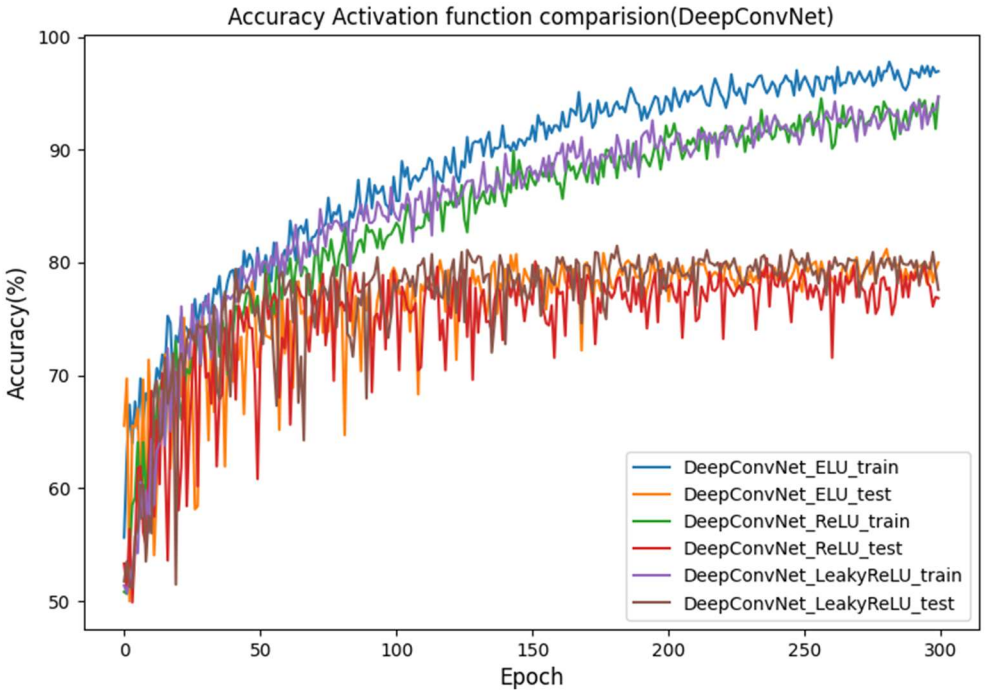

也解決的 dead relu 的現象，利用**指數**來當基準計算

## 3. Experimental results

### A. The highest testing accuracy

- Screenshot with two models



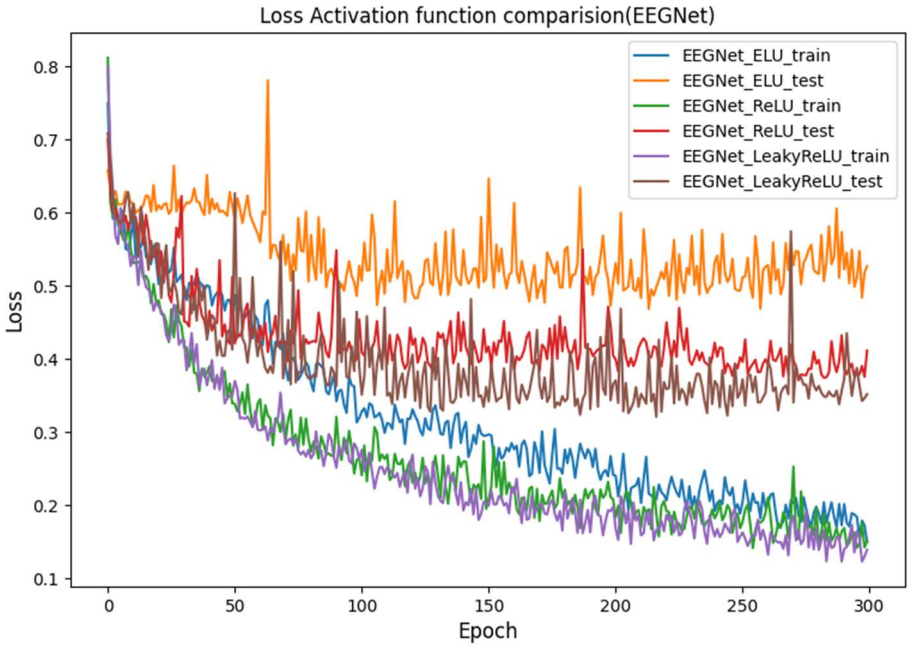



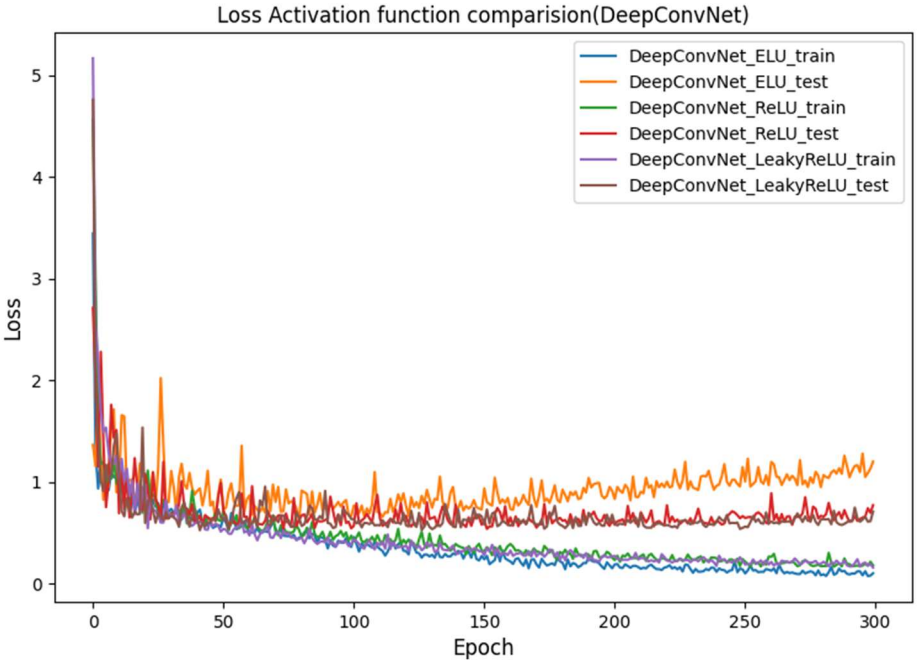

**Best accurancy**

| | ELU | ReLU | LeakyReLU |
|---|---|---|---|
| EEGNet | 83.333333 | 86.759259 | 88.888889 |
| DeepConvNet | 81.203704 | 80.370370 | 81.481481 |

- Anything you want to present

loss value




值得一提的是：

這邊在算 acc 時只要除總數 1080 即可得到 accuracy 的%

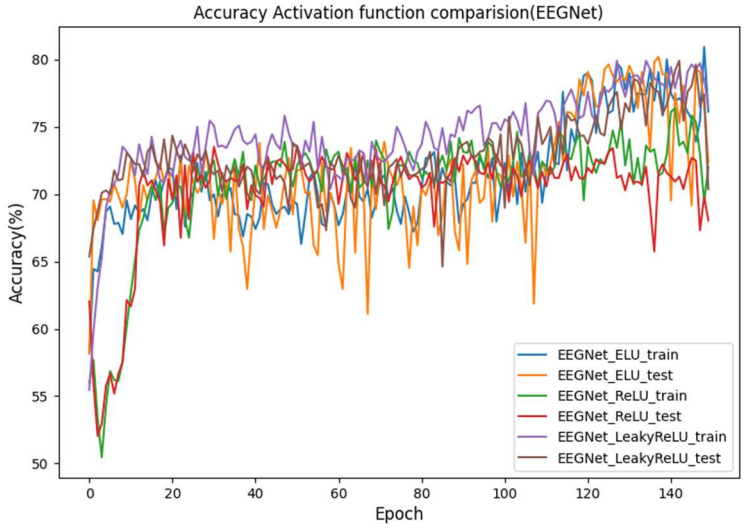
但是 loss 不一樣，會因為 batch size 的緣故，需要除不同的值。

## B. Comparison Figures

下面是我只調整 lr 所帶來不同的結果。

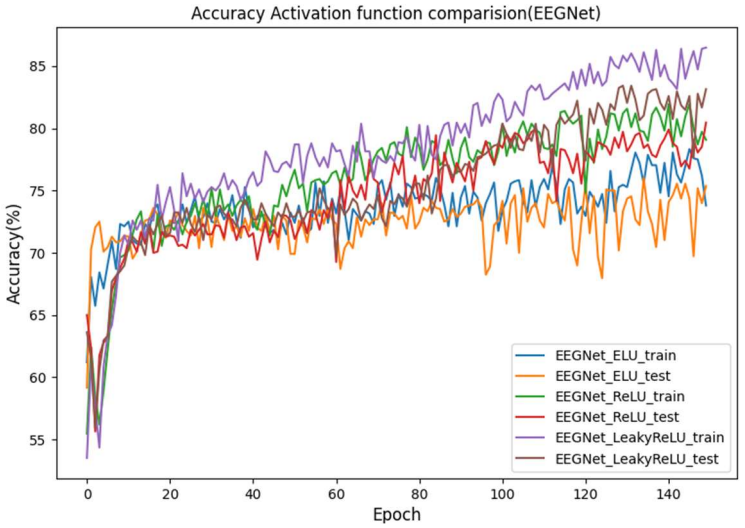
Batch_size:128,optimizer:Adam,lr:{'lr': 0.1}

| | ELU | ReLU | LeakyReLU |
|---|---|---|---|
| EEGNet | 80.185185 | 73.611111 | 79.907407 |
| DeepConvNet | 81.574074 | 60.277778 | 61.759259 |



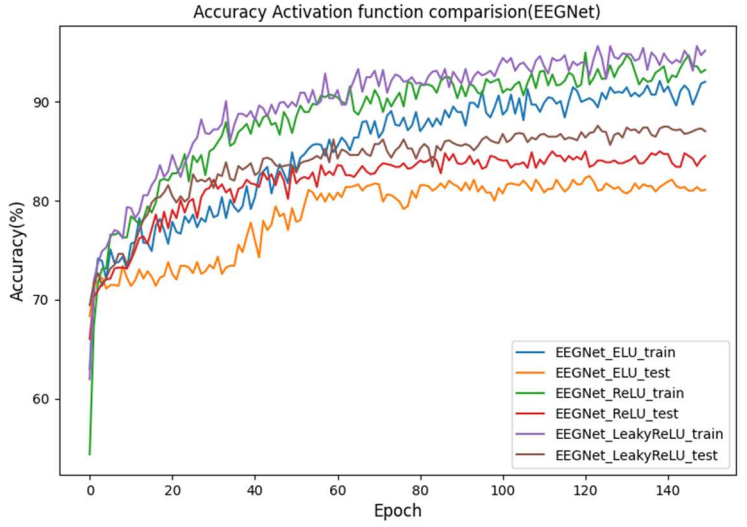

Batch_size:128,optimizer:Adam,lr:{'lr': 0.05}

| | ELU | ReLU | LeakyReLU |
|---|---|---|---|
| EEGNet | 76.018519 | 80.462963 | 83.425926 |
| DeepConvNet | 81.203704 | 74.814815 | 79.444444 |



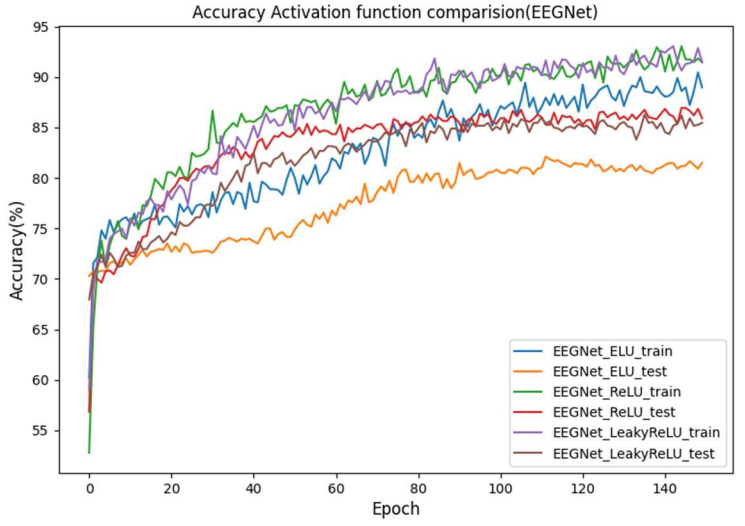

Batch_size:128,optimizer:Adam,lr:{'lr': 0.005}

| | ELU | ReLU | LeakyReLU |
|---|---|---|---|
| EEGNet | 82.500000 | 85.000000 | 87.592593 |
| DeepConvNet | 81.296296 | 79.166667 | 80.648148 |




Batch_size:128,optimizer:Adam,lr:{'lr': 0.001}

| | ELU | ReLU | LeakyReLU |
|---|---|---|---|
| EEGNet | 82.129630 | 86.944444 | 86.296296 |
| DeepConvNet | 80.555556 | 82.314815 | 81.018519 |




發現在 optimizer 為 Adam 時 ：

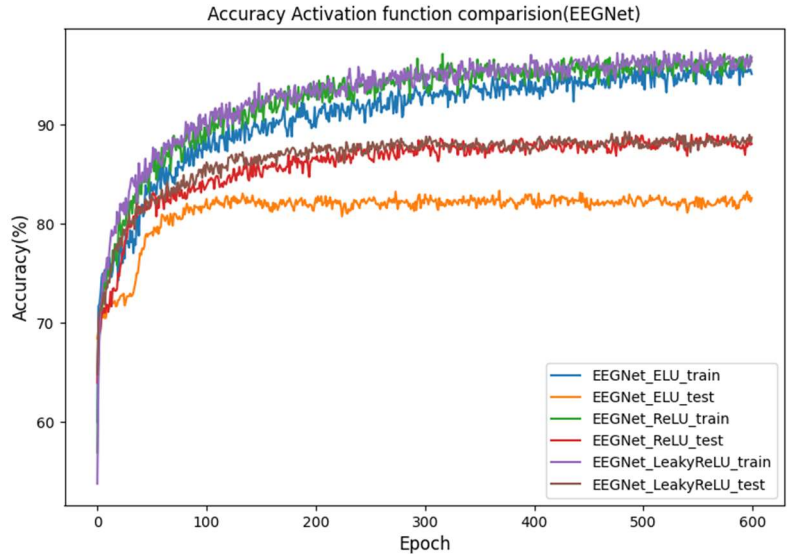
learning rate 太大並無好的結果且 acc 上下浮動劇烈。(無法進入 opt 區間)

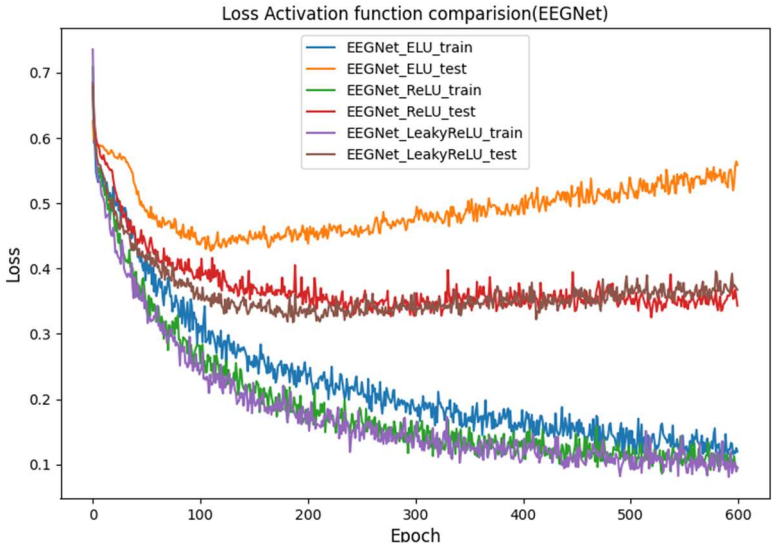
lerrning rate 調小果然浮動劇烈消失，且穩定上升

## 4. Discussion

### A. Anything you want to share

在測資時有遇到一個有趣的現象，照理來說當 loss 下降時，通常代表著 accuracy 的上升。但卻發生了 loss 上升 accuracy 也上升的情況 如圖



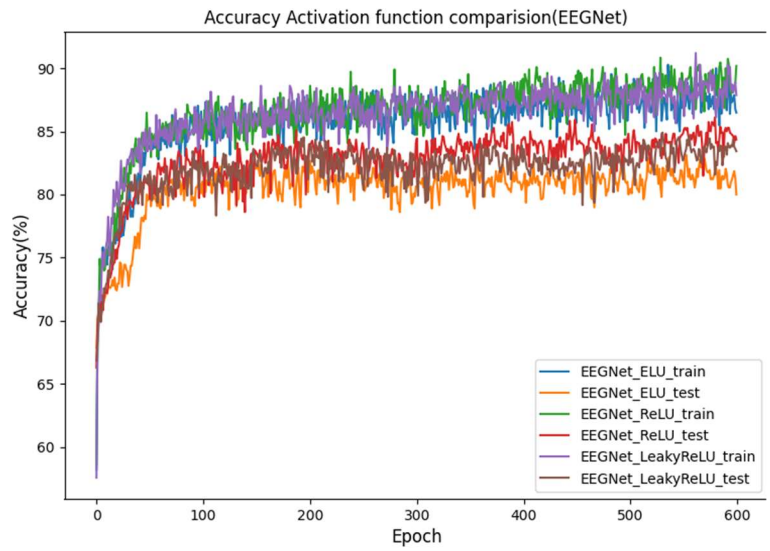

很明顯的由橘、棕色的線可以看出左圖 acc 為不變且些微上升的情況，右圖 loss 橘、棕線卻有往上的趨勢。

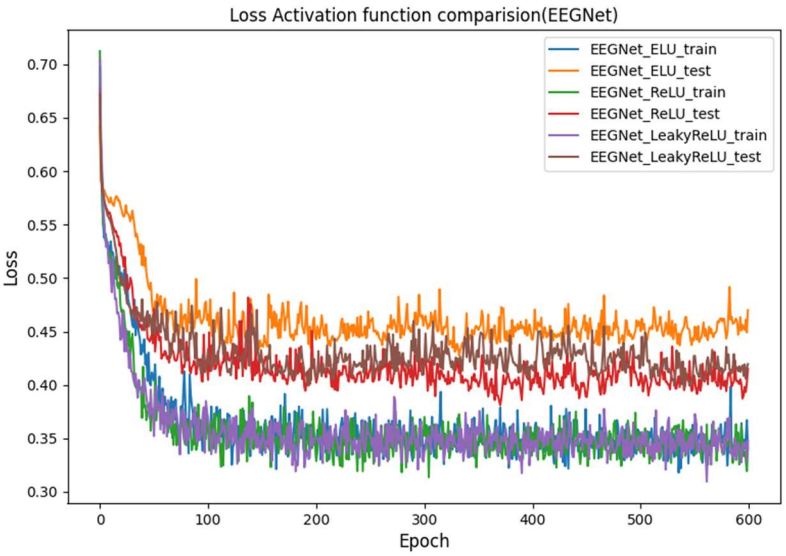
上網查證後發現可以使用一種 flooding 的正規化方法來避免此種反常理的情況發生，以下為公式。

$$\tilde{J}(\boldsymbol{\theta}) = |J(\boldsymbol{\theta}) - b| + b$$

其中 J 代表原始的目標函數，b 則是一個 hyperparmeter，為了是要設定訓練損失的下限。

下面是我套用後的成果。雖然整體浮動大但是少了 loss acc 同方向的反常理情況




另外由於我 b 值是按照上上圖抓個 0.3 附近，相信再摸索一下能得到更漂亮的數值