Lab2 : EEG classification

311581005 智能碩一 吳佳豪

1 INTRODUCTION

EEG 是在無刺激的狀況下收集腦波，它所測量到的大腦電生理活動，是以毫秒為單位，具有很高的時間分辨率，並且為非侵入性的研究工具。希望藉由腦波和大腦意識的相對應關係，透過測量腦波來推測大腦的狀態。

這次作業主要是透過 EEGNet 與 DeepConvNet 兩個模型對 EEG 的資料集做 classification，並且比較使用三個不同的 activation function，分別是 ReLU、LeakyReLU、ELU 來看對模型分類的影響。

2 EXPERIMENT SET UP

2.1 THE DETAIL OF YOUR MODEL

2.1.1 EEGNet

EEGNet 主要是由一個基本的 convolution + 一個 Depthwise Convolution + 一個 Separable Convolution(Depthwise conv + Pointwise conv)組成，所以可以有效的降低參數量。

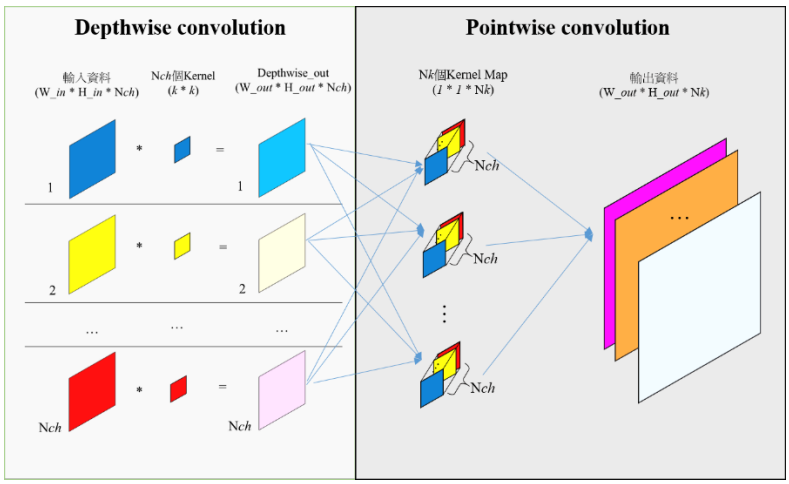


Figure 1. Separable Convolution

2.1.1.1 Depthwise Convolution

對輸入資料的 channel 都建立一個 k*k 的 filter，channel 各自對自己的 filter 做 convolution，完成後的 feature map 數量會與輸入的資料 channel 數相同，這樣雖然可以減少參數量，但是會降低不同通道與空間上的 feature 資訊。

2.1.1.2 Pointwise convolution

對輸入資料建立 1*1*input channel 大小的 filter，如果 Depthwise conv 後面加上 Pointwise conv 進行運算則可以取得不同通道間的空間資訊，補足 Depthwise conv 的缺點，並且也有可升降維的優點。

```
class EEGNet(nn.Module):
    def __init__(self, activationFunction):
        super(EEGNet, self).__init__()

        self.firstconv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False),
            nn.BatchNorm2d(16, eps=1e-05, momentum=0.1,affine=True, track_running_stats=True)
        )
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1),stride=(1, 1), groups=16, bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1,affine=True, track_running_stats=True),
            activationFunction(),
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(p=0.25)
        )
        self.separableConv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1,affine=True, track_running_stats=True),
            activationFunction(),
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p=0.25)
        )
        self.classify = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=736, out_features=2, bias=True)
        )

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

Figure 2. EEGNet model implement

2.1.2 DeepConvNet

主要是由多層的 convolution layer、Pooling layer 與 full connection layer 組成，model 結構皆與提供參考的架構相同。

```python
class DeepConvNet(nn.Module):
    def __init__(self, activationFunction):
        super(DeepConvNet, self).__init__()

        self.model = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5), bias=True),
            nn.Conv2d(25, 25, kernel_size=(2, 1), bias=True),
            nn.BatchNorm2d(25, eps=1e-5, momentum=0.1),
            activationFunction(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),

            nn.Conv2d(25, 50, kernel_size=(1, 5), bias=True),
            nn.BatchNorm2d(50, eps=1e-5, momentum=0.1),
            activationFunction(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),

            nn.Conv2d(50, 100, kernel_size=(1, 5), bias=True),
            nn.BatchNorm2d(100, eps=1e-5, momentum=0.1),
            activationFunction(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),

            nn.Conv2d(100, 200, kernel_size=(1, 5), bias=True),
            nn.BatchNorm2d(200, eps=1e-5, momentum=0.1),
            activationFunction(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),
        )
        self.classify = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=8600, out_features=2),
        )

    def forward(self, x):
        x = self.model(x)
        x = self.classify(x)
        return x
```

Figure 3. DeepConvNet model implement

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

2.2.1 ReLU

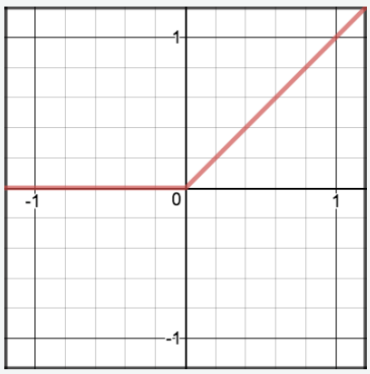
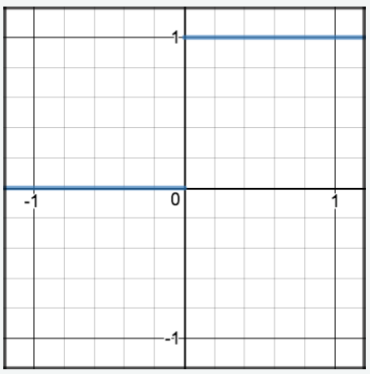
Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z <= 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$
	

Figure 4. ReLU function and derivative

ReLU 在輸入值為正時，不存在飽和問題，可以避免 sigmoid 會發生的梯度消失問題，因此可以訓練更深層的網路，而當輸入值為負時，直接將值當 0，因此計算速度與收斂速度都較 sigmoid 與 tanh 快。但缺點是反向傳播時，當輸入為負數時梯度將完全為零，則該神經元的梯度永遠都會是 0，發生所謂的 Dead ReLU 問題。

2.2.2 Leaky ReLU

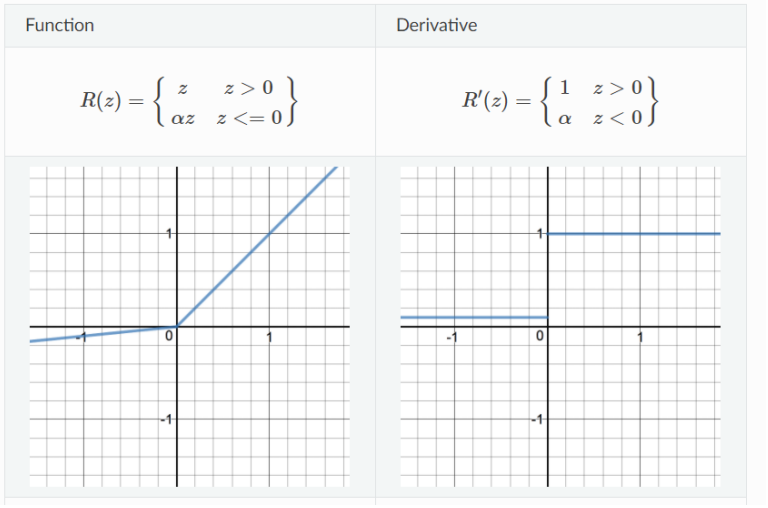


Figure 5. Leaky ReLU function and derivative

Leaky ReLU 在負值時會給予一個微小的斜率，使得為負值時不會直接當 0，因此解決 ReLU 的 Dead ReLU 問題。但理論上雖然應該都會比 ReLU 好，但經過大量驗證後，由於實際效果不穩定，所以不一定永遠比 ReLU 好。

2.2.3 ELU

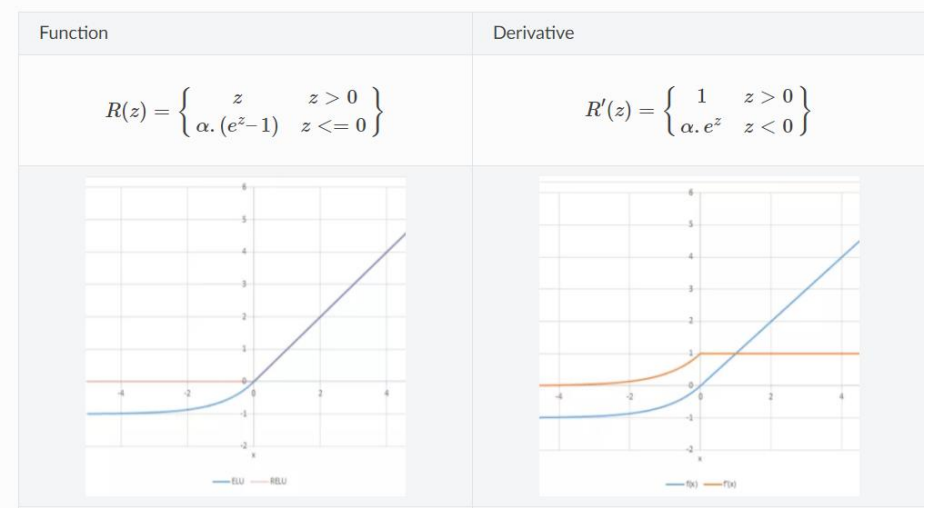


Figure 6. ELU function and derivative

ELU 使用 exponential 的函數因此在較小的輸入下會飽和至負值，可以讓梯度逐漸趨緩，使均值向 0 加速學習，並且有 ReLU 大部分的優點，也解決 Dead ReLU 問題，但計算量大且也無法驗證效果比 ReLU 好。

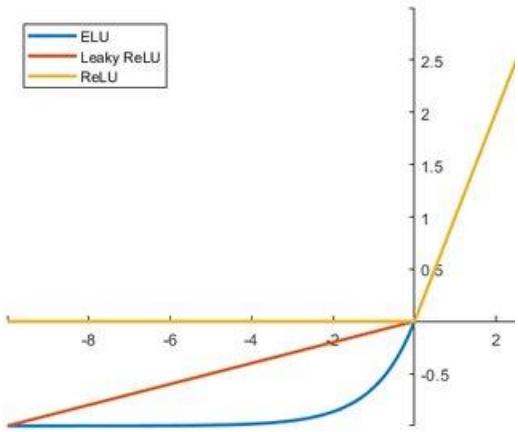


Figure 7. ReLU、Leaky ReLU、ELU function comparison

3 EXPERIMENTAL RESULTS

超參數設定：

Table 1. Model hyperparameter settings

Hyperparameters	
Epochs	1000
Batch_size	128
Learning_rate	1e-3
Optimizer	Adam
Weight_deacy	1e-4
Early_stop	400
Random seed	123456
Data augmentation	Horizontal Shift Add Noise

3.1 THE HIGHEST TESTING ACCURACY

3.1.1 Show Results

下方實驗結果皆依照 Table1.參數進行設定。

發現在使用 EEGNet 架構並搭配 ReLU 激活函數時，可以達到 **88.80%**的最高 accuracy。

Table 2. Accuracy of all models

Results	ReLU	Leaky ReLU	ELU
EEGNet	**88.80%**	**88.33%**	87.13%
DeepConvNet	85.83%	86.48%	**87.41%**

由下面兩張圖可以發現 EEDNet 因為模型大小相較 DeepConvNet 還要來的小，所以收斂速度都較快，但也相對地比較容易 Overfitting，所以訓練到後面 test loss 都有上升一點。

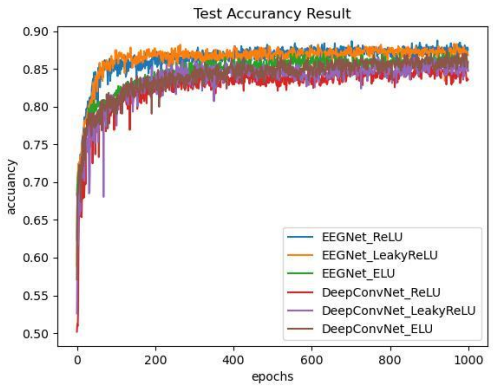


Figure 8. Test accuracy results

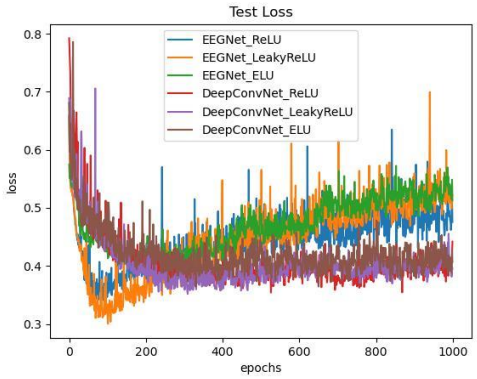


Figure 9. Test loss results

3.1.2 Anything you want to present (Change Optimizer)

以下實驗結果參數都依照 Table1.參數進行設定，除了 Optimizer 以及 SGD 與 RMSprop 皆添加 momentum=0.9 進行訓練網路。可以發現使用 Adagrad 在每個項目都獲取最低的 accuracy，可能原因是因為 learning rate 過小且沒有添加 momentum，所以收斂速度過慢所導致。並依照下方表格可以發現 Adam 因為擁有 Adaptive + Momentum 的特色，所以在所有項目中皆拿到高的精確度，因此其他實驗皆使用 Adam 作為本次作業的最佳 Optimizer。

Table 3. Accuracy of all models with different optimizer

Optimizer	EEGNet			DeepConvNet		
	ReLU	Leaky ReLU	ELU	ReLU	Leaky ReLU	ELU
SGD	86.39%	87.13%	85.19%	82.04%	81.48%	82.87%
Adagrad	80.65%	79.26%	79.44%	79.17%	77.22%	79.81%
RMSprop	86.30%	87.78%	86.20%	**87.87%**	**87.69%**	87.31%
Adam	**88.80%**	**88.33%**	**87.13%**	85.83%	86.48%	**87.41%**

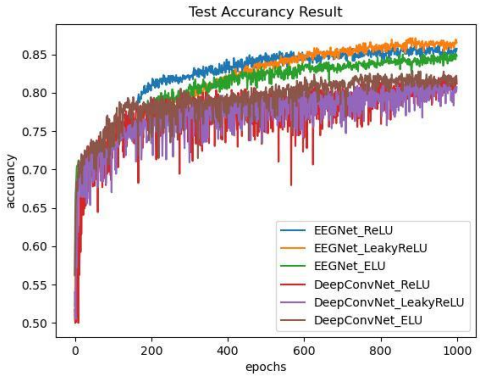


Figure 10. SGD

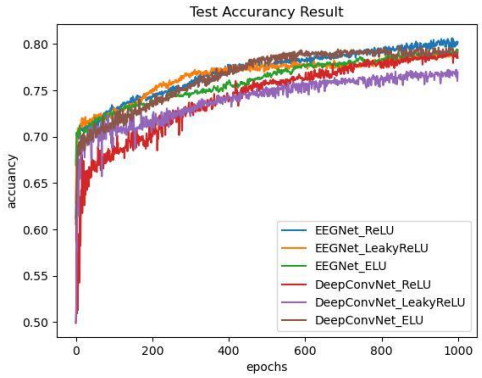


Figure 11. Adagrad

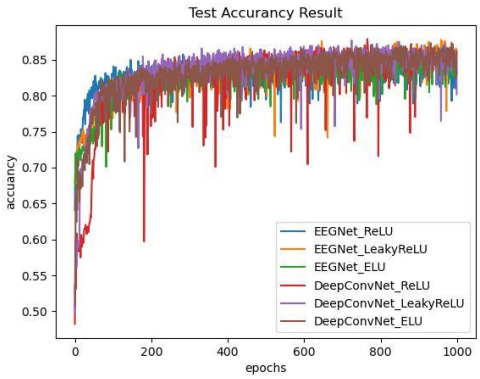


Figure 12. RMSprop

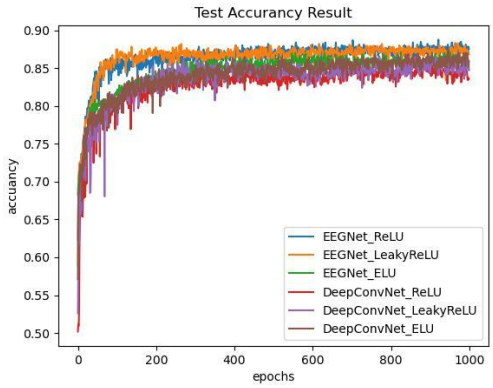


Figure 13. Adam

3.2 COMPARISON FIGURES

以下實驗結果參數都依照 Table1.參數進行設定。模型皆都使用 ReLU、Leaky ReLU、ELU 進行訓練與測試，可以發現使用 ELU 的 activation function 在 EEG 模型上面會收斂較慢且有較差的表現，但在 DeepConvNet 卻有較佳的效果。

3.2.1 EEGNet

詳細 Accuracy 參照 Table 2.。

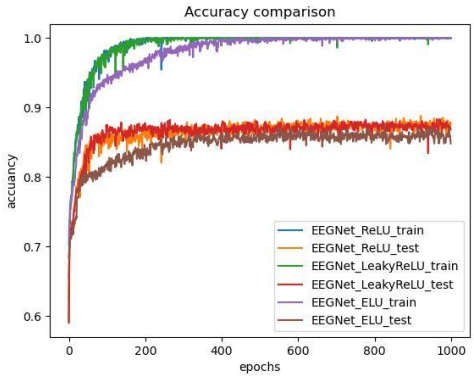


Figure 14. EEGNet Test accuracy results

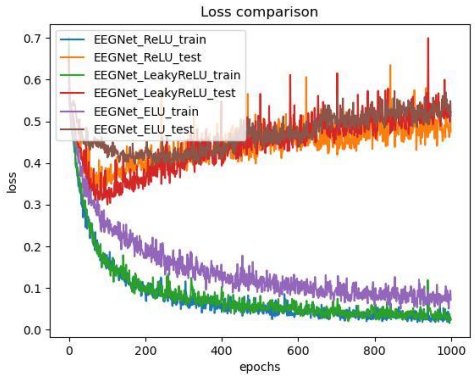


Figure 15. EEGNet Test loss results

3.2.2 DeepConvNet

詳細 Accuracy 參照 Table 2.。

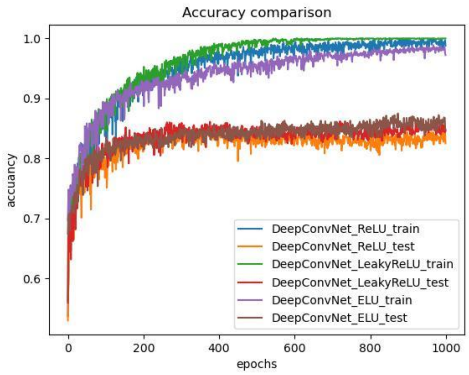


Figure 16.DeepConvNet Test accuracy results

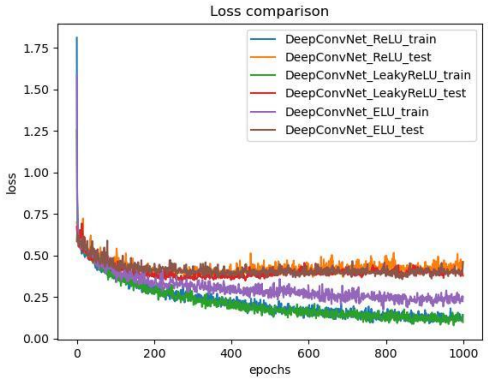


Figure 17. DeepConvNet Test loss results

4 DISCUSSION

DATA AUGMENTATION

數據增強主要將現有的人工數據資料透過技術策略來加以生成新數據，防止訓練時過度擬和與增加資料量，來解決資料不足的問題。

以下實驗結果參數都依照 Table1.參數進行設定，除了 data Augmentation。本次作業實驗主要透過增加雜訊(add Noise)和隨機位移(RandomShift)來增加精確度。

經過實驗可以發現單獨增加 noise 的效果反而會降低準確度，但是如果同時添加 RandomShift 時，可以稍微增加 accuracy。

Table 4. Accuracy of all models with different data augmentation

Data Augmentation	EEGNet			DeepConvNet		
	ReLU	Leaky ReLU	ELU	ReLU	Leaky ReLU	ELU
NoUse	87.87%	87.78%	85.00%	83.70%	84.07%	82.41%
AddNoise	87.31%	86.20%	84.63%	82.69%	82.59%	81.02%
RandomShift	88.06%	88.24%	**87.96%**	**85.83%**	**86.48%**	**87.41%**
AddAll	**88.80%**	**88.33%**	87.13%	**85.83%**	**86.48%**	**87.41%**

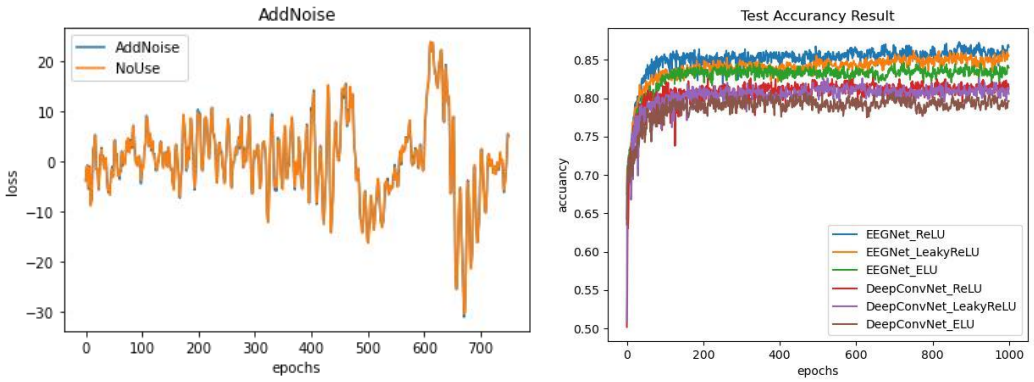


Figure18 .Only add noise

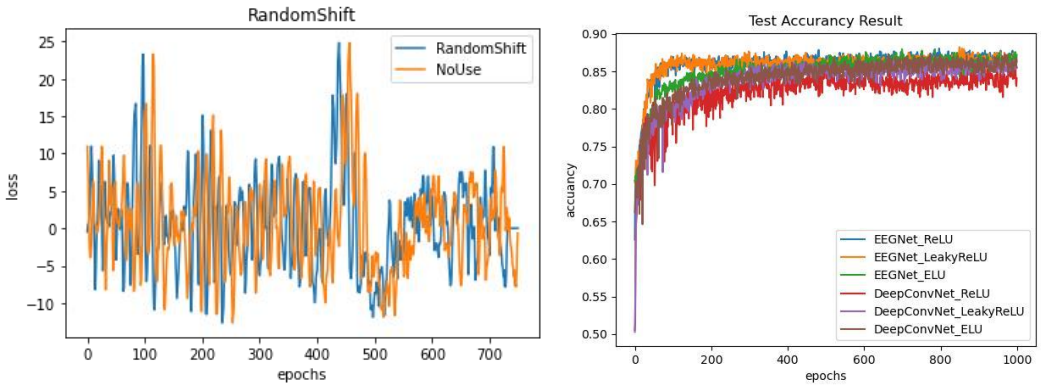


Figure 19.Only add randomshift

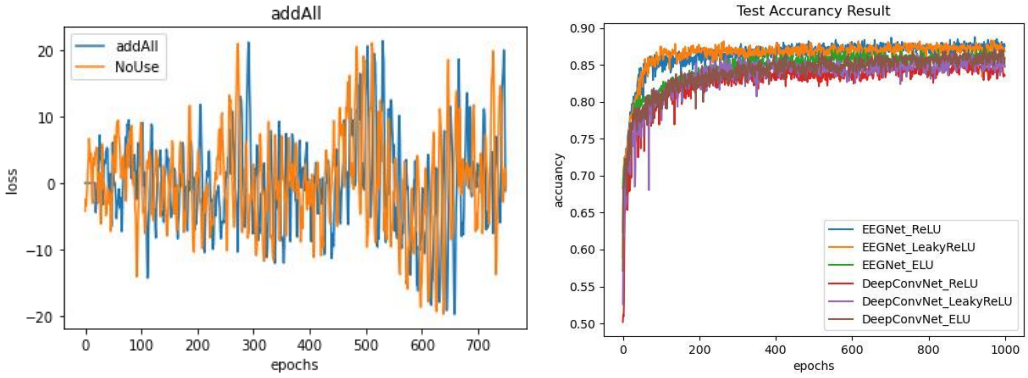


Figure 20. add noise and randomshift