NCTU_Deep Learning Lab2

309833023 簡廷羽

1. Introduction

   In this lab, we need to implement two different network ” EEGNet” and ” DeepConvNet” and feed BCI dataset into those architectures to train and test the data. Besides, we are required to use three different kind of activation function “ReLU”、”LeakyRelu”、”ELU” to adjust the model. Then, we also have to plot images include train and test dataset, display accuracy with different activation functions.

2. Experiment set up (30%)

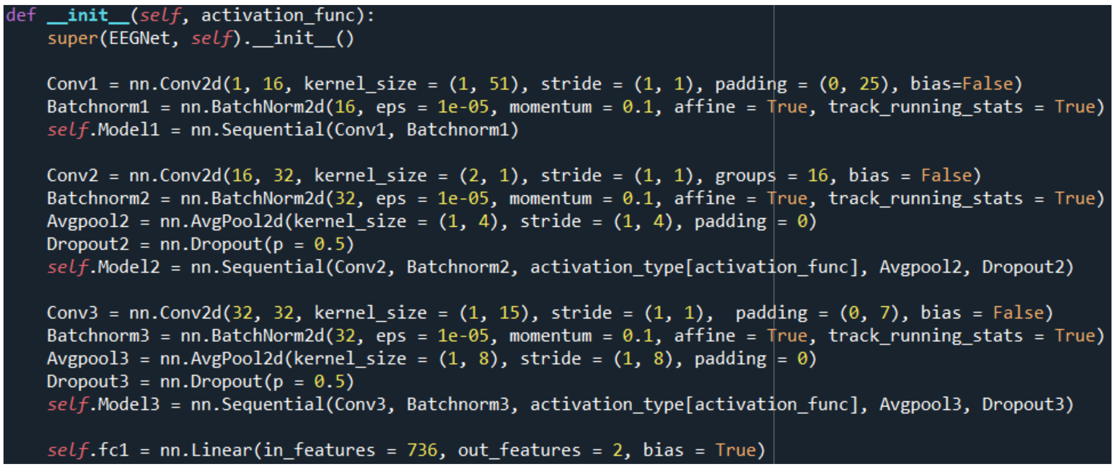
   A. The detail of your model

   During implementing these two models, I just follow the hints of the documents which TA supply. By Filling in the parameters and calling torch.nn library, it’s easy to complete building models.

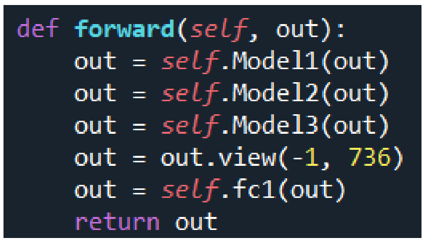
   For EEGNet, three layers are needed and a fc-layer putting at last layer due to the use of cross entropy. For DCNet, five layers are implement and also a linear function added.

   Then accomplish output flow of “forward”, and send it layer by layer, whole model had been finished.

   1. EEGNet

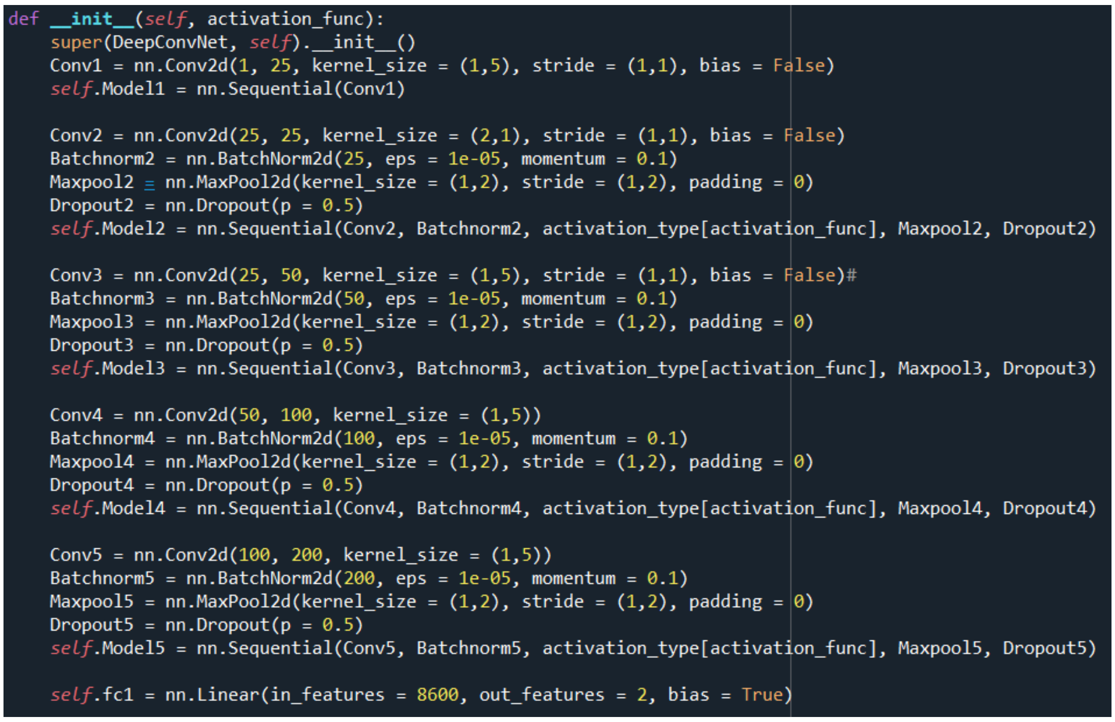


```
def __init__(self, activation_func):
    super(EEGNet, self).__init__()

    Conv1 = nn.Conv2d(1, 16, kernel_size = (1, 51), stride = (1, 1), padding = (0, 25), bias=False)
    Batchnorm1 = nn.BatchNorm2d(16, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True)
    self.Model1 = nn.Sequential(Conv1, Batchnorm1)

    Conv2 = nn.Conv2d(16, 32, kernel_size = (2, 1), stride = (1, 1), groups = 16, bias = False)
    Batchnorm2 = nn.BatchNorm2d(32, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True)
    Avgpool2 = nn.AvgPool2d(kernel_size = (1, 4), stride = (1, 4), padding = 0)
    Dropout2 = nn.Dropout(p = 0.5)
    self.Model2 = nn.Sequential(Conv2, Batchnorm2, activation_type[activation_func], Avgpool2, Dropout2)

    Conv3 = nn.Conv2d(32, 32, kernel_size = (1, 15), stride = (1, 1),  padding = (0, 7), bias = False)
    Batchnorm3 = nn.BatchNorm2d(32, eps = 1e-05, momentum = 0.1, affine = True, track_running_stats = True)
    Avgpool3 = nn.AvgPool2d(kernel_size = (1, 8), stride = (1, 8), padding = 0)
    Dropout3 = nn.Dropout(p = 0.5)
    self.Model3 = nn.Sequential(Conv3, Batchnorm3, activation_type[activation_func], Avgpool3, Dropout3)

    self.fc1 = nn.Linear(in_features = 736, out_features = 2, bias = True)
```

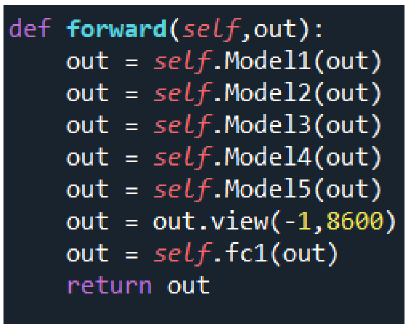


```
def forward(self, out):
    out = self.Model1(out)
    out = self.Model2(out)
    out = self.Model3(out)
    out = out.view(-1, 736)
    out = self.fc1(out)
    return out
```

2.DeepConvNet



```
def __init__(self, activation_func):
    super(DeepConvNet, self).__init__()
    Conv1 = nn.Conv2d(1, 25, kernel_size = (1,5), stride = (1,1), bias = False)
    self.Model1 = nn.Sequential(Conv1)

    Conv2 = nn.Conv2d(25, 25, kernel_size = (2,1), stride = (1,1), bias = False)
    Batchnorm2 = nn.BatchNorm2d(25, eps = 1e-05, momentum = 0.1)
    Maxpool2 = nn.MaxPool2d(kernel_size = (1,2), stride = (1,2), padding = 0)
    Dropout2 = nn.Dropout(p = 0.5)
    self.Model2 = nn.Sequential(Conv2, Batchnorm2, activation_type[activation_func], Maxpool2, Dropout2)

    Conv3 = nn.Conv2d(25, 50, kernel_size = (1,5), stride = (1,1), bias = False)#
    Batchnorm3 = nn.BatchNorm2d(50, eps = 1e-05, momentum = 0.1)
    Maxpool3 = nn.MaxPool2d(kernel_size = (1,2), stride = (1,2), padding = 0)
    Dropout3 = nn.Dropout(p = 0.5)
    self.Model3 = nn.Sequential(Conv3, Batchnorm3, activation_type[activation_func], Maxpool3, Dropout3)

    Conv4 = nn.Conv2d(50, 100, kernel_size = (1,5))
    Batchnorm4 = nn.BatchNorm2d(100, eps = 1e-05, momentum = 0.1)
    Maxpool4 = nn.MaxPool2d(kernel_size = (1,2), stride = (1,2), padding = 0)
    Dropout4 = nn.Dropout(p = 0.5)
    self.Model4 = nn.Sequential(Conv4, Batchnorm4, activation_type[activation_func], Maxpool4, Dropout4)

    Conv5 = nn.Conv2d(100, 200, kernel_size = (1,5))
    Batchnorm5 = nn.BatchNorm2d(200, eps = 1e-05, momentum = 0.1)
    Maxpool5 = nn.MaxPool2d(kernel_size = (1,2), stride = (1,2), padding = 0)
    Dropout5 = nn.Dropout(p = 0.5)
    self.Model5 = nn.Sequential(Conv5, Batchnorm5, activation_type[activation_func], Maxpool5, Dropout5)

    self.fc1 = nn.Linear(in_features = 8600, out_features = 2, bias = True)
```



```
def forward(self,out):
    out = self.Model1(out)
    out = self.Model2(out)
    out = self.Model3(out)
    out = self.Model4(out)
    out = self.Model5(out)
    out = out.view(-1,8600)
    out = self.fc1(out)
    return out
```

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

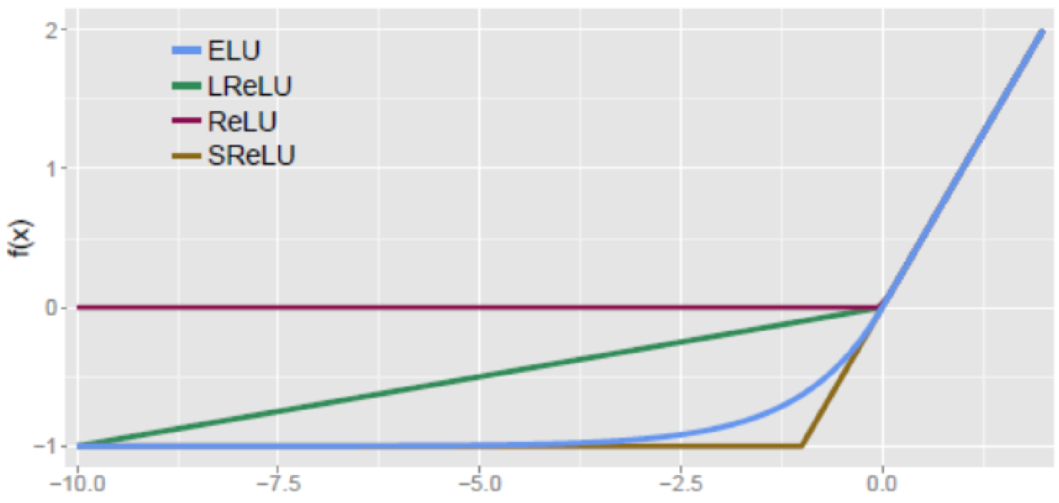
Relu set negative value to zero, LeakyRelu set a non-zero slope to it instead,ELU with exponential is smoother than above two functions.

Below two graphs are formulas and image comparison.

ReLU: $R(z) = max(0, \ z)$

Leaky-Relu: $y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0 \end{cases}$

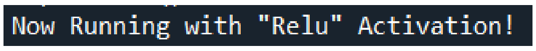
ELU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$



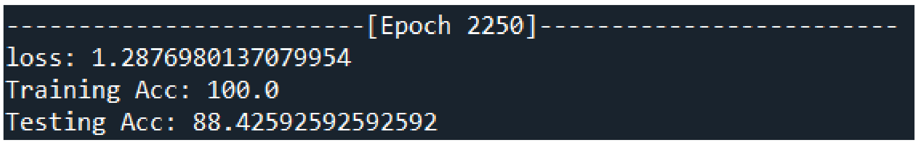
3. Experimental results (30%)

A. The highest testing accuracy

EEG_Relu:

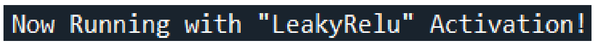


```
Now Running with "Relu" Activation!
```

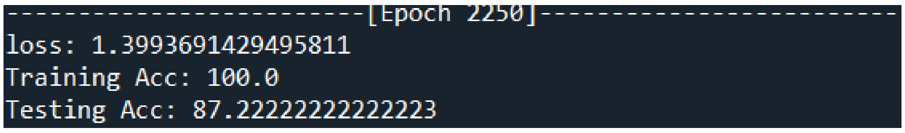


```
-------------------------[Epoch 2250]-------------------------
loss: 1.2876980137079954
Training Acc: 100.0
Testing Acc: 88.42592592592592
```

EEG_LeakyRelu:

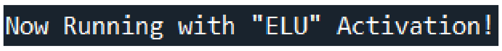


```
Now Running with "LeakyRelu" Activation!
```

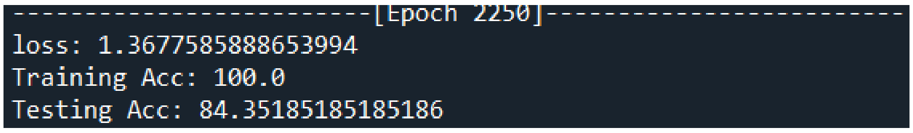


```
-------------------------[Epoch 2250]-------------------------
loss: 1.3993691429495811
Training Acc: 100.0
Testing Acc: 87.22222222222223
```

EEG_ELU:

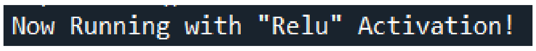


```
Now Running with "ELU" Activation!
```

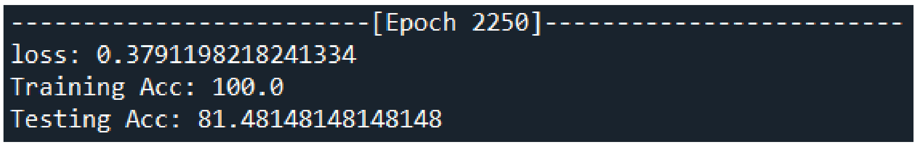


```
-------------------------[Epoch 2250]-------------------------
loss: 1.3677585888653994
Training Acc: 100.0
Testing Acc: 84.35185185185186
```

DCN_Relu:

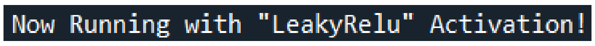


```
Now Running with "Relu" Activation!
```

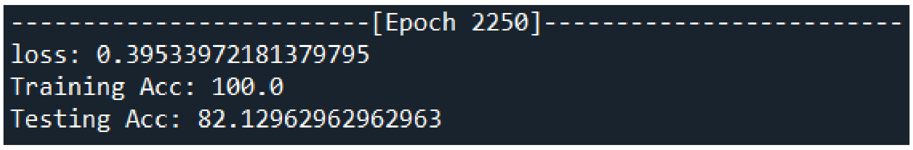


```
-------------------------[Epoch 2250]-------------------------
loss: 0.3791198218241334
Training Acc: 100.0
Testing Acc: 81.48148148148148
```

DCN_LeakyRelu:

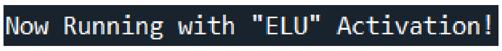


```
Now Running with "LeakyRelu" Activation!
```

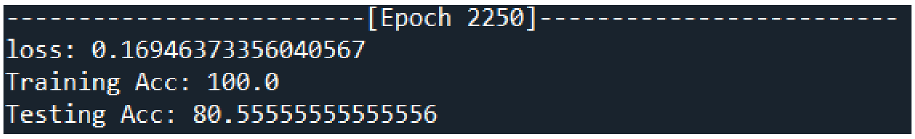


```
-------------------------[Epoch 2250]-------------------------
loss: 0.39533972181379795
Training Acc: 100.0
Testing Acc: 82.12962962962963
```

DCN_ELU:



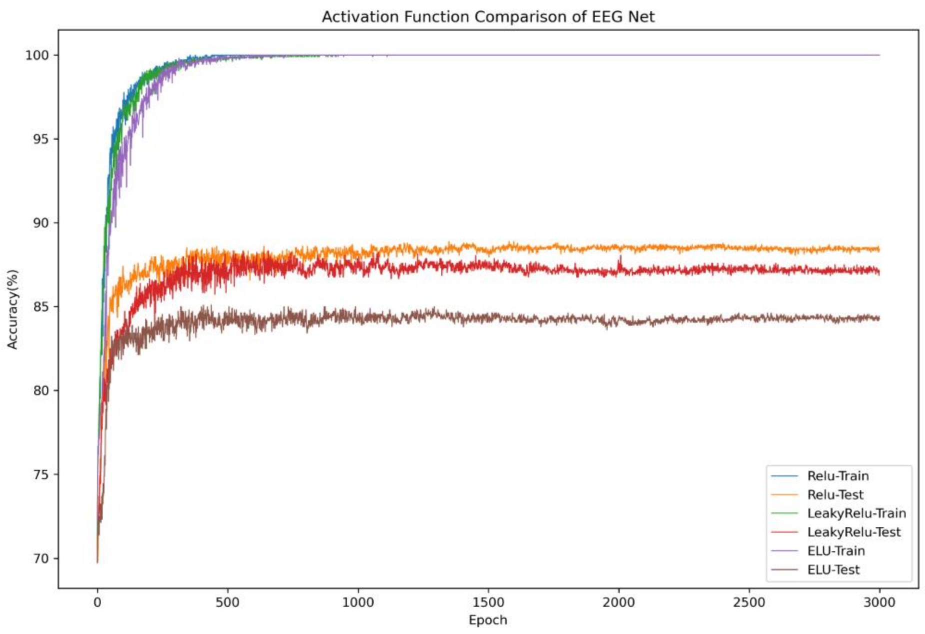
```
Now Running with "ELU" Activation!
```



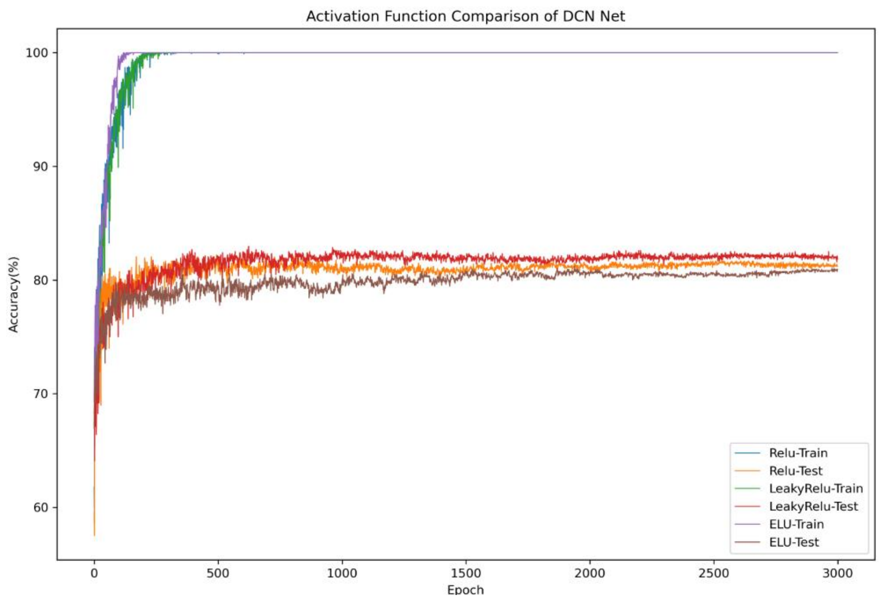
```
-------------------------[Epoch 2250]-------------------------
loss: 0.16946373356040567
Training Acc: 100.0
Testing Acc: 80.55555555555556
```

B. Comparison figures

EEGNet



DeepConvNet



4. Discussion (20%)

Thanks to TA's guide, it's not quite difficult to fill in parameters and build a model without errors. But when writing during train and test functions, there exist bugs that I cannot fixed. So I reference to another version and try to make some revision of the architecture. Finally, it can run perfectly, and code writing skill inside was awesome that I spent lots of time to understand. I think I must make great efforts to advance my coding skill, especially file and data manufacturing.