DL_LAB2_Report _EEG classification

學號：312554004　姓名：林垣志

(1) Introduction:

In this lab, I will implement simple EEG classification models which are EEGNet, DeepConvNet with BCI competition dataset. The datasets are provided by TA, include [S4b_train.npz, X11b_train.npz] and [S4b_test.npz, X11b_test.npz], and I need to try different kinds of activation function including ReLU, Leaky ReLU, ELU.

There are 1080 data in both training data and testing data shown below.

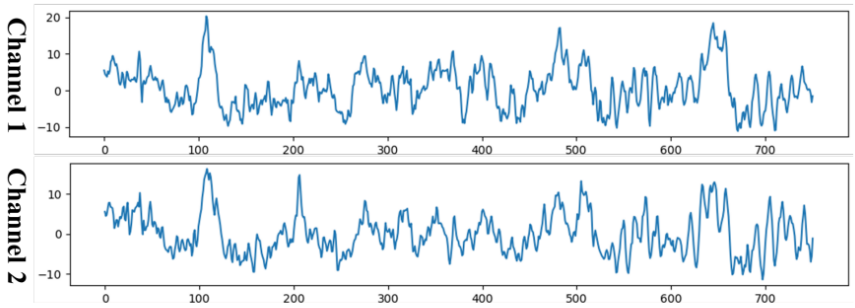


Figure 1: BCI Dataset

(2) Experiment setups:

(a) The detail of your model

- EEGNet:

The figure (code) below shows the EEGNet model. EEGNet use depthwise separable convolution to replace conventional convolution. This way can reduce the computational complexity. It separates conventional convolution into depthwise and separable convolution. Depthwise convolution will learn the correlation between different signal channel. Separable convolution will learn how to combine each feature map.

```
class EEGNet(nn.Module):
    def __init__(self, activation = nn.ReLU()):
        super(EEGNet, self).__init__()

        # Layer 1
        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-5, momentum = 0.1, affine = True, track_running_stats = True),
        )

        # Depthwise Layer
        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size = (2, 1), stride = (2, 1), groups = 16, bias = False),
            nn.BatchNorm2d(32, eps = 1e-5, momentum = 0.1, affine = True, track_running_stats = True),
            activation,
            nn.AvgPool2d(kernel_size = (1, 4), stride = (1, 4), padding = 0),
            nn.Dropout(p = 0.25)
        )

        # Separable Layer
        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-5, momentum = 0.1, affine = True, track_running_stats = True),
            activation,
            nn.AvgPool2d(kernel_size = (1, 8), stride = (1, 8), padding = 0),
            nn.Dropout(p = 0.25)
        )

        # in_features = 736, out_features = 2
        self.classify = nn.Sequential(nn.Linear(736, 2, bias = True))
```

```
    def forward(self, x):
        firstResults = self.firstConv(x)
        dwResults = self.depthwiseConv(firstResults)
        separableResults = self.separableConv(dwResults)

        """ view results """
        separableResults = separableResults.view(separableResults.shape[0],-1)

        out = self.classify(separableResults)
        return out
```

- DeepConvNet:

  The figure (code) below shows DeepConvNet model, it's a traditional CNN architecture.

```
class DeepConvNet(nn.Module):
    def __init__(self, activation = nn.ReLU()):
        super(DeepConvNet, self).__init__()

        """
         conv0, input = [1, 1, C=2, T=750], filter=25, kernel_size=(1,5)
         conv1, filter=25, kernel_size=(2,1)
         conv2, filter=50, kernel_size=(1,5)
         conv3, filter=100, kernel_size=(1,5)
         conv4, filter=200, kernel_size=(1,5)
        """

        self.convnet1 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5)),
            nn.Conv2d(25, 25, kernel_size=(2, 1)),
            nn.BatchNorm2d(25, eps = 1e-5, momentum = 0.1),
            activation,
            nn.MaxPool2d(kernel_size = (1, 2)),
            nn.Dropout(p = 0.5)
        )

        Filters = [25,50,100,200]

        for i in range(1,len(Filters)):
            setattr(self, f'convnet{i+1}', nn.Sequential(
                nn.Conv2d(Filters[i-1], Filters[i], kernel_size = (1,5)),
                nn.BatchNorm2d(Filters[i], eps = 1e-5, momentum = 0.1),
                activation,
                nn.MaxPool2d(kernel_size = (1,2)),
                nn.Dropout(p = 0.5)
            ))

        self.classify = nn.Linear(8600, 2)

    def forward(self, x):
        results1 = self.convnet1(x)
        results2 = self.convnet2(results1)
        results3 = self.convnet3(results2)
        results4 = self.convnet4(results3)

        """ view results """
        results4 = results4.view(results4.shape[0], -1)

        out = self.classify(results4)
        return out
```

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

1. **ReLU**:

ReLU (Rectified Linear Unit) is an activation function that transforms all negative values to zero while keeping positive values unchanged, ensuring that the output is always greater than or equal to zero. However, it has some drawbacks. One potential issue is that positive values may become unbounded, and converting negative values to zero can cause neurons to become unresponsive to errors since the gradient becomes zero.

$$ReLU(x) = max(0, x)$$

2. **Leaky ReLU:**

Unlike ReLU, Leaky ReLU introduces a small, non-zero slope for negative inputs, typically in the range of 0.01 to prevent neurons from becoming completely inactive, helps prevent the "dying ReLU" problem.

$$LeakyReLU(x) = \begin{cases} x, \ if \ \ x \geq 0 \\ a * x, \ if \ \ x < 0 \end{cases}$$

3. **ELU (Exponential Linear Unit):**

Unlike ReLU, it allows for negative values and exhibits a smoother transition around zero. However, similar to ReLU, it can still suffer from the issue of unbounded positive values.

$$ELU(x) = max(0,x) + min(0,\alpha * (e^x - 1))$$

(3) Results of your testing:

(a) The highest testing accuracy

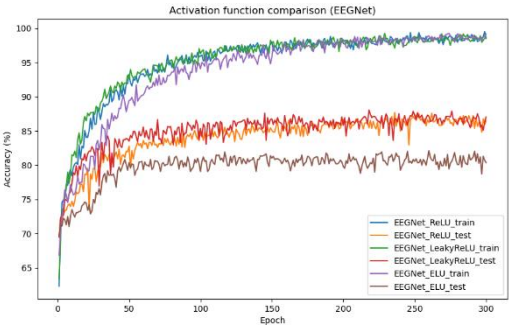
The hyper parameters I use are shown below.

Batch_size	Learning rate	Epochs	Optimizer	Loss function
64	0.001	300	Adam	CrossEntropy

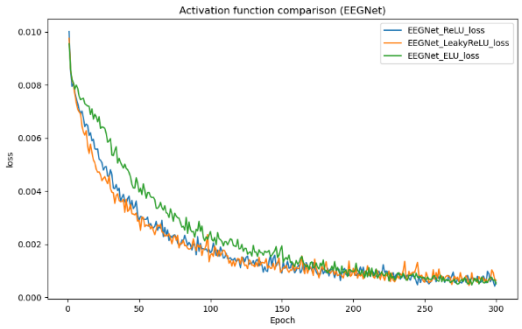
The highest testing accuracy are shown below.

	ReLU	**Leaky ReLU**	**ELU**
EGGNet	**87.685**	**88.055**	**82.129**
DeepConvNet	81.759	81.574	81.018

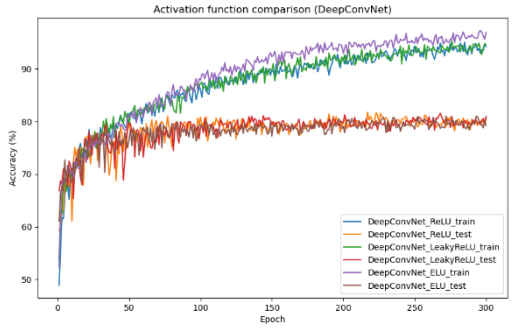
(b) Comparison figures



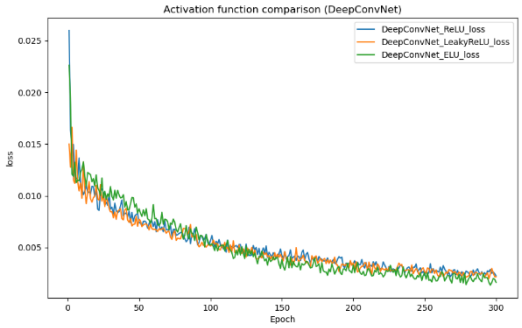
(a) EEGNet accuracy



(b) EEGNet loss



(c) DeepConvNet accuracy



(d) DeepConvNet loss

(4) Discussion:

(a) The number of parameter

As mentioned above section 2, the EEGNet model utilizes depthwise and separable convolutions, which not only reduce computational complexity but also decrease the number

of parameters. This is the main point of differentiation from the DeepConvNet model, and the EEGNet model actually has a total of 17,874 parameters, while the DeepConvNet model has 150,977 parameters, a difference of approximately 8 times.

(b) The results of different activation function

In the experiments, I observed that using ELU as activation function in both EEGNet and DeepConvNet model yielded comparatively worse results when compared to using ReLU or Leaky ReLU. We can see Leaky ReLU has 88% accuracy in EEGNet, and ReLU has 81.75% accuracy in DeepConvNet.

(c) Encountered Issue

- At first, I didn't know why the dataset couldn't be directly loaded. Later, I learned that I needed to first put the data into a "TensorDataset()" (by using torch.Tensor() or torch.from_numpy()), and then pass it to DataLoader().
- When the batch size is set to 64, overfitting issues typically arise after approximately 150 iterations of training. While increasing the batch size leads to a slight improvement in accuracy, but it can't be increased too much. One advantage of using a larger batch size is that it allows for training with more data, which helps mitigate overfitting. However, this also comes with the drawback of consuming more memory.
- The other issue is that I found the data need to be .to(device) to run on the GPU. Type of tensor can be operated on both CPU and GPU, while type of numpy can only run on the CPU.