Lab 2 EEG classification

林穎志 411551007

1. Introduction

This is a binary classification task using deep learning approach on EEG Data. We implemented the classic EEG model: EEGNet and DeepConvNet. Furthermore, we tried different settings to seek for the best performance and plotted the figures containing training/testing accuracy and loss.

2. Experiment set up

A. The detail of your model

a. EEGNet

EEGNet is a compact convolutional neural network for EEG-based BCIs that can generalize across different BCI paradigms in the presence of limited data and can produce interpretable features. EEGNet introduced the use of **depthwise and separable convolutions** for EEG signal classification task. The autors cliamed that they use a **depthwise convolution** to learn frequency-specific spatial filters and the **separable convolution** is a combination of a depthwise convolution, which learns a temporal summary for each feature map individually, followed by a pointwise convolution, which learns how to optimally mix the feature maps together.

EEGNet is composed of 3 blocks:

In the block 1, the author performed 2 convolutional steps including a 2D convolution and a depthwise convolution. Applied batch normalization along the feature map dimension before applying exponential linear unit(ELU) nonlinearity.

In the block 2, they performed a seperable convolution, which is a depthwise convolution followed by a pointwise convolutions.

And the last block contains of a dense layer and an activation function. The model architecture is showed as below.

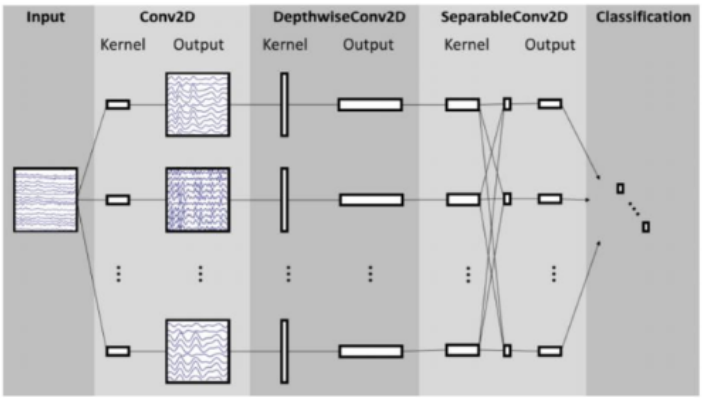
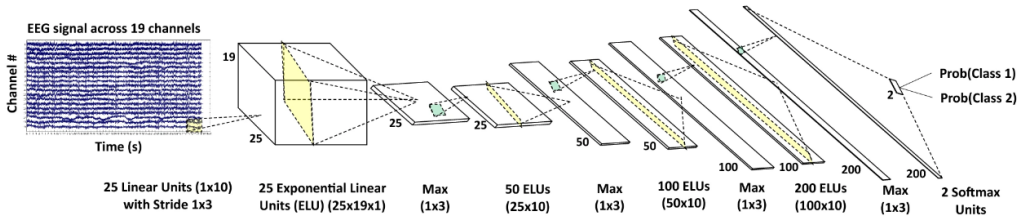


Fig. 1 EEGNet

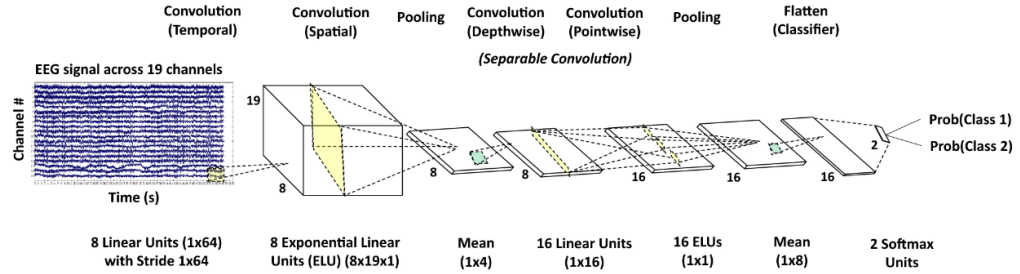
```
class EEGNet(nn.Module):
    def __init__(self, args, activation):

        super(EEGNet, self).__init__()
        # self.nb_classes = nb_classes
        self.dropoutRate = args.dropout_rate
        self.activation = activation
        self.alpha = args.elu_alpha

        if self.activation == 'elu':
            self.act_fun = nn.ELU(alpha=self.alpha)
        elif self.activation == 'leakyrelu':
            self.act_fun = nn.LeakyReLU()
        else:
            self.act_fun = nn.ReLU()

        # Layer 1
        self.l1 = nn.Sequential(
            nn.Conv2d(1, 16, (1, 51), padding = (0, 25), bias = False),  #(1, 64),
            nn.BatchNorm2d(16, False)
        )

        #DepthwiseConv2D D * F1 (C, 1)
        self.Depthwise = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, stride = (1,1), kernel_size=(2, 1), groups=16, bias=False),
            nn.BatchNorm2d(32, False),
            self.act_fun,
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout2d(0.25)
        )

        # Layer 2
        self.Separable = nn.Sequential(
            nn.Conv2d(in_channels = 32, out_channels = 32, kernel_size=(1, 15), stride = (1,1), bias=False, padding=(0,7)),
            nn.BatchNorm2d(32, False),
            self.act_fun,
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout2d(0.25)
        )

        # FC Layer
        self.fc1 = nn.Linear(in_features=736, out_features=2, bias=True)

    def forward(self, x):
```

b. DeepConvNet

Compares to EEGNet, DeepConvNet mainly uses max pooling in its architecture, and its model architecture compares to EEGNet is shown as below:



(c) Deep convolutional neural network architecture. Convolutions are shown in yellow and all have stride 1×3, across the temporal dimension. Max pool layers in green.



(d) EEGNet architecture. Convolutions are shown in yellow and all have stride across the temporal dimension. Mean pool layers in green.

Fig. 2 Comparison of EEGNet and DeepConvNet (source: https://www.nature.com/articles/s41598-020-70569-y/figures/3)

```
class DeepConvNet(nn.Module):
    def __init__(self, args):

        super(DeepConvNet, self).__init__()
        # self.nb_classes = nb_classes
        self.dropoutRate = args.dropout_rate
        self.activation = args.activation
        self.alpha = args.elu_alpha

        if self.activation == 'elu':
            self.act_fun = nn.ELU(alpha=self.alpha)
        elif self.activation == 'leakyrelu':
            self.act_fun = nn.LeakyReLU()
        else:
            self.act_fun = nn.ReLU()


        # Layer 1
        self.b1 = nn.Sequential(
            nn.Conv2d(1, 25, (1, 5)),
            nn.Conv2d(25, 25, (2, 1)),
            nn.BatchNorm2d(25, False),
            self.act_fun,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(self.dropoutRate)
        )

        self.b2 = nn.Sequential(
            nn.Conv2d(25, 50, (1, 5)),
            nn.BatchNorm2d(50, False),
            self.act_fun,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(self.dropoutRate)
        )

        self.b3 = nn.Sequential(
            nn.Conv2d(50, 100, (1, 5)),
            nn.BatchNorm2d(100, False),
            self.act_fun,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(self.dropoutRate)
        )

        self.b4 = nn.Sequential(
            nn.Conv2d(100, 200, (1, 5)),
            nn.BatchNorm2d(200, False),
            self.act_fun,
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(self.dropoutRate)
        )

        # FC Layer
        # NOTE: This dimension will depend on the number of timestamps per sample in your data.
```

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

    def forward(self, x):
```

B. Activation function

a. ReLU

ReLU is an activation function defined as:

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

In other words, it returns the input value if it is positive or zero, and for negative inputs, it outputs zero.

b. Leaky ReLU

Leaky ReLU is a modified version of ReLU that addresses the dying ReLU problem. It is defined as:

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

where α is a small constant typically set to a small positive value like 0.01. The main difference between Leaky ReLU and ReLU is that Leaky ReLU introduces a small slope αx for negative inputs instead of setting them to zero.

c. ELU

ELU was first proposed in the paper: *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. ELU is defined as:

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

ELU is a function that tend to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has a extra alpha constant which should be positive number. ELU has the advantages of

(1)ELU becomes smooth slowly until its output equal to −α whereas ReLU sharply smoothes

(2)ELU is a strong alternative to ReLU

(3)Unlike to ReLU, ELU can produce negative outputs. On the contrast, ELU has disadvantages that for all x larger than 0, it can blow up the activation with the output range of [0, inf].

3. Experiment results

A. The highest testing accuracy(Screenshot)

Table 1

Model	**Accuracy**	Activation function	Learning rate	Optimizer	Epoch	Loss	Dropout
EEGNet	**88.15%**	ReLu	0.001	Adam	300	Cross entropy	0.25
DeepConVNet	**84.54%**	ReLu	0.001	Adam	300	Cross entropy	0.2

Model	ReLu	Leaky Relu	ELU
EEGNet	**88.15%**	84.72%	82.5%
DeepConVNet	**84.54%**	83.61%	82.59%

```
 65%|                                                                | 195/300 [00:32<00:20,  5.24it/s]
Epoch 00196: reducing learning rate of group 0 to 6.2500e-05.
 75%|                                                                | 226/300 [00:38<00:12,  5.89it/s]
Epoch 00227: reducing learning rate of group 0 to 3.1250e-05.
 86%|                                                                | 257/300 [00:43<00:07,  5.70it/s]
Epoch 00258: reducing learning rate of group 0 to 1.5625e-05.
 96%|                                                                | 288/300 [00:48<00:02,  5.58it/s]
Epoch 00289: reducing learning rate of group 0 to 7.8125e-06.
100%|                                                               | 300/300 [00:50<00:00,  5.92it/s]
 29%|                                                                 | 88/300 [00:14<00:37,  5.69it/s]
Epoch 00089: reducing learning rate of group 0 to 5.0000e-04.
 40%|                                                                | 119/300 [00:19<00:25,  6.97it/s]
Epoch 00120: reducing learning rate of group 0 to 2.5000e-04.
 56%|                                                                | 168/300 [00:31<00:53,  2.46it/s]
Epoch 00169: reducing learning rate of group 0 to 1.2500e-04.
 68%|                                                                | 203/300 [00:45<00:40,  2.41it/s]
Epoch 00204: reducing learning rate of group 0 to 6.2500e-05.
 87%|                                                                | 262/300 [01:10<00:16,  2.26it/s]
Epoch 00263: reducing learning rate of group 0 to 3.1250e-05.
 98%|                                                                | 293/300 [01:23<00:02,  2.45it/s]
Epoch 00294: reducing learning rate of group 0 to 1.5625e-05.
100%|                                                               | 300/300 [01:26<00:00,  3.49it/s]
Test acc of ReLu: 88.14814814814815, Test acc of ELu: 82.5, Test acc of LeakyReLu: 84.72222222222221
```

Fig.3 Screenshot of accuracy (EEGNet)

```
Epoch 00272: reducing learning rate of group 0 to 3.1250e-05.
100%|                                                               | 300/300 [02:21<00:00,  2.11it/s]
 35%|                                                                | 106/300 [00:47<01:26,  2.25it/s]
Epoch 00107: reducing learning rate of group 0 to 5.0000e-04.
 55%|                                                                | 164/300 [01:14<01:04,  2.12it/s]
Epoch 00165: reducing learning rate of group 0 to 2.5000e-04.
 75%|                                                                | 225/300 [01:43<00:34,  2.16it/s]
Epoch 00226: reducing learning rate of group 0 to 1.2500e-04.
 93%|                                                                | 280/300 [02:08<00:09,  2.09it/s]
Epoch 00281: reducing learning rate of group 0 to 6.2500e-05.
100%|                                                               | 300/300 [02:18<00:00,  2.17it/s]
 20%|                                                                 | 59/300 [00:21<01:02,  3.88it/s]
Epoch 00060: reducing learning rate of group 0 to 5.0000e-04.
 35%|                                                                | 105/300 [00:33<00:44,  4.41it/s]
Epoch 00106: reducing learning rate of group 0 to 2.5000e-04.
 46%|                                                                | 138/300 [00:42<00:34,  4.72it/s]
Epoch 00139: reducing learning rate of group 0 to 1.2500e-04.
 63%|                                                                | 189/300 [00:54<00:27,  4.02it/s]
Epoch 00190: reducing learning rate of group 0 to 6.2500e-05.
 97%|                                                                | 292/300 [01:20<00:02,  3.12it/s]
Epoch 00293: reducing learning rate of group 0 to 3.1250e-05.
100%|                                                               | 300/300 [01:22<00:00,  3.63it/s]
Test acc of ReLu: 84.53703703703704, Test acc of ELu: 82.5925925925926, Test acc of LeakyReLu: 83.61111111111111
```

Fig.4 Screenshot of accuracy (DeepConvNet)

B. Comparison figures

a. EEGNet

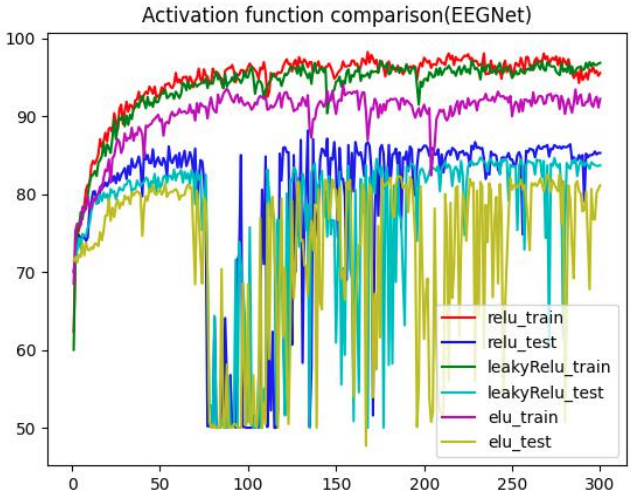


Fig.5 Comparison of different activation function using EEGNet

b. DeepConvNet

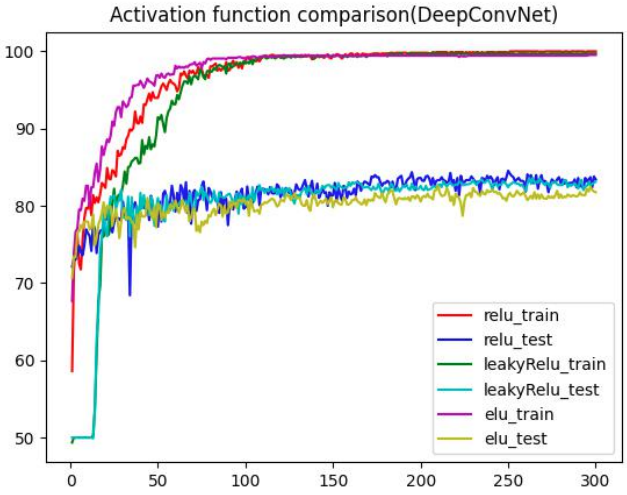


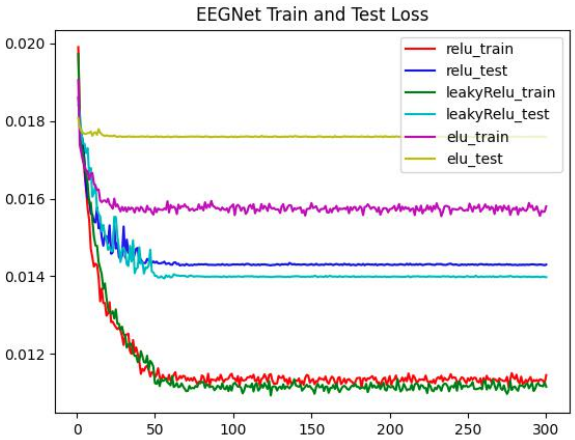
Fig.6 Comparison of different activation function using DeepConvNet

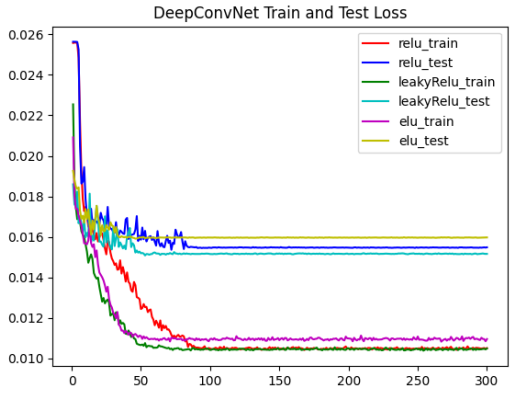
4. Discussion

A. Anything you want to share

We had done some experiments to seek for the best setting of the hyperparameters. Looking upon table 2, we can see that the right alpha value of the ELU function is very important while it may influence the performance. Nonetheless, the more the dropout rate is, the worse the performance is(table 3). However, a well rate of dropout could make a model more robust and prevent overfitting. We finally test the 3 activation functions to discuss how the different activation functions influence the loss.

a. How different activation functions influence loss





We can observe that using ELU as the activation function converge faster for both EEGNet and DeepConvNet, while its accuracy is not the best.

b. How alpha in ELU function influences accuracy

Table 2

EEGNet	**Alpha = 0.4**	**Alpha = 0.7**	**Alpha = 1.0**
Accuracy	**82.87%**	**81.85%**	**81.48%**
Activation function	ELU	ELU	ELU
Optimizer	Adam	Adam	Adam
Epoch	300	300	300
Learning rate	0.001	0.001	0.001

We can observe that alpha value in ELU do influence the final test accuracy.

c. How dropuout rate influences accuracy

Table 3

EEGNet	**Dropout rate = 0.25**	**Dropout rate = 0.5**	**Dropout rate = 0.75**
Accuracy	88.15%	86.85%	84.35%
Activation function	ReLu	ReLu	ReLu
Optimizer	Adam	Adam	Adam
Epoch	300	300	300
Learning rate	0.001	0.001	0.001

5. Extra

In this section, I modified the architecture of EEGNet by adding a channel-wise attention block which is SE Block inside of it. The architecture of SE block is shown as below. We can see that in SE block, the feature is first applied a global average pooling then put into 2 fully connect layers with the shape 1*1*C to 1*1*C/r and 1*1*C/r to 1*1*C following a sigmoid function, then the output feature is matmul with the input. We insert 2 SE block after the first layer and depthwise convolution. The result shows that the accuracy drops a little bit after the modification, we infer that due to the particularity of EEG data, though the channel-wised attention performs well in natural image but not the domain of EEG data.

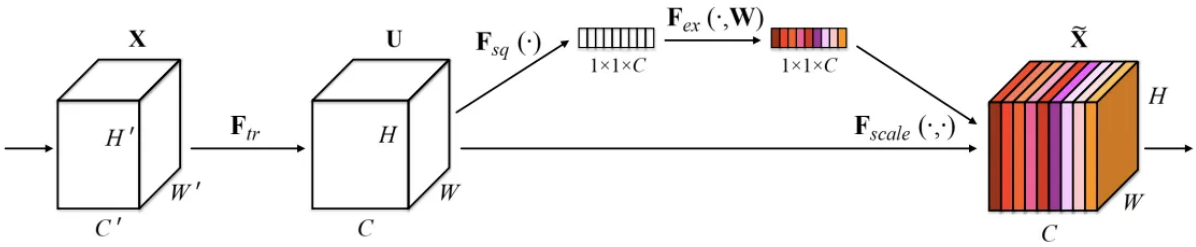


Fig. 7 SE Block

```
def SE_block(in_block, ch, ratio=2):
    x = GlobalAveragePooling2D()(in_block)
    x = Dense(ch//ratio, activation='relu')(x)
    x = Dense(ch, activation='sigmoid')(x)
    return multiply()([in_block, x])

class AEEGNet(nn.Module):
    def __init__(self, args):

        super(AEEGNet, self).__init__()
        # self.nb_classes = nb_classes
        self.dropoutRate = args.dropout_rate
        self.activation = args.activation
        self.alpha = args.elu_alpha

        if self.activation == 'elu':
            self.act_fun = nn.ELU(alpha=self.alpha)
        elif self.activation == 'leakyrelu':
            self.act_fun = nn.LeakyReLU()
        else:
            self.act_fun = nn.ReLU()


        # Layer 1
        self.l1 = nn.Sequential(
            nn.Conv2d(1, 16, (1, 51), padding = (0, 25), bias = False),  #(1, 64),
            nn.BatchNorm2d(16, False)
        )

        self.attention1 = SEBlock(in_channels = 16)

        #DepthwiseConv2D D * F1 (C, 1)
        '''When groups == in_channels and out_channels == K * in_channels,
        where K is a positive integer, this operation is also known as a “depthwise convolution”.
        '''
        #16, 1, 2, 1
```

```
        self.Depthwise = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=(2, 1), stride = (1,1), groups=16, bias=False),
            nn.BatchNorm2d(32, False),
            self.act_fun,
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(self.dropoutRate)
        )

        self.attention2 = SEBlock(in_channels = 32)

        # Layer 2
        self.Separable = nn.Sequential(
            nn.Conv2d(in_channels = 32, out_channels = 32, kernel_size=(1, 15), stride = (1,1), bias=False, padding=(0,7)),
            nn.BatchNorm2d(32, False),
            self.act_fun,
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(self.dropoutRate)
        )

        # FC Layer
        # NOTE: This dimension will depend on the number of timestamps per sample in your data.
        self.fc1 = nn.Linear(in_features=736, out_features=2, bias=True)



    def forward(self, x):
```

	EEGNet	**Attention_EEGNet**
Accuracy	**88.15%**	**87.96%**
Activation function	ReLu	ReLu
Optimizer	Adam	Adam
Epoch	300	300
Learning rate	0.001	0.001

```
Epoch 00134: reducing learning rate of group 0 to 2.5000e-04.
 57%|                                                  | 172/300 [00:59<00:29,  4.39it/s]
Epoch 00173: reducing learning rate of group 0 to 1.2500e-04.
 68%|                                                  | 203/300 [01:10<00:32,  3.02it/s]
Epoch 00204: reducing learning rate of group 0 to 6.2500e-05.
 85%|                                                  | 255/300 [01:26<00:15,  2.88it/s]
Epoch 00256: reducing learning rate of group 0 to 3.1250e-05.
100%|                                                  | 300/300 [01:43<00:00,  2.91it/s]
 34%|                                                  | 103/300 [00:29<00:39,  5.04it/s]
Epoch 00104: reducing learning rate of group 0 to 5.0000e-04.
 57%|                                                  | 170/300 [00:43<00:31,  4.07it/s]
Epoch 00171: reducing learning rate of group 0 to 2.5000e-04.
 67%|                                                  | 201/300 [00:50<00:20,  4.85it/s]
Epoch 00202: reducing learning rate of group 0 to 1.2500e-04.
 79%|                                                  | 236/300 [00:57<00:14,  4.44it/s]
Epoch 00237: reducing learning rate of group 0 to 6.2500e-05.
 89%|                                                  | 267/300 [01:05<00:08,  3.95it/s]
Epoch 00268: reducing learning rate of group 0 to 3.1250e-05.
 99%|                                                  | 298/300 [01:11<00:00,  5.49it/s]
Epoch 00299: reducing learning rate of group 0 to 1.5625e-05.
100%|                                                  | 300/300 [01:12<00:00,  4.14it/s]
Test acc of ReLu: 87.96296296296296, Test acc of ELu: 83.7037037037037, Test acc of LeakyReLu: 86.57407407407408
```