DLP Homework 3

電控碩 0860077 王國倫

1. Introduction

In this lab, we need to implement simple EEG classification models which are EEGNet(Fig.1), Deep-ConvNet with BCI competition dataset(Fig.2), and also use different kinds of activation function including ReLU, Leaky ReLU, ELU. Try to adjust model hyper-parameters to get the highest accuracy of two architectures with three kinds of activation functions, and plot each epoch accuracy during training and testing.

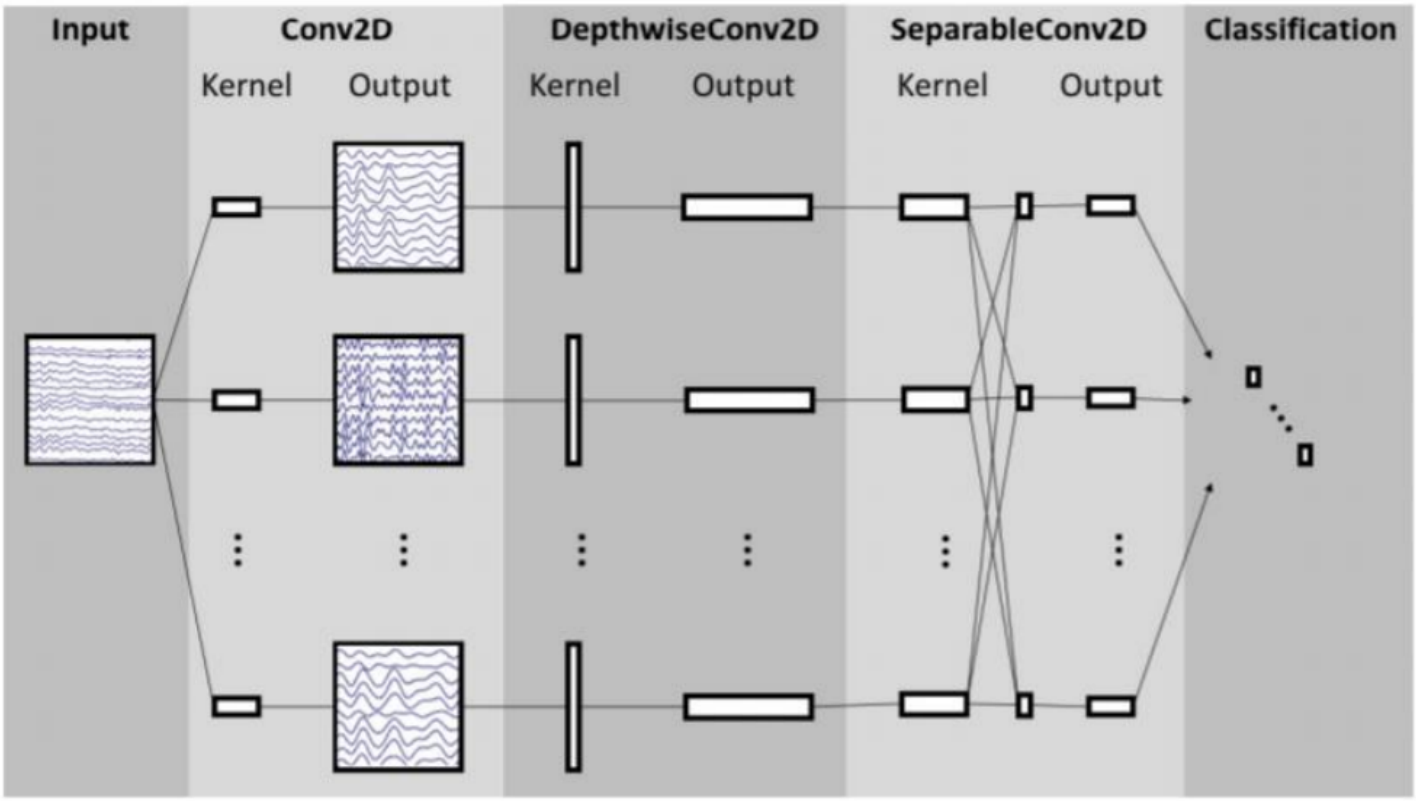

Fig.1 EEGNet architecture

The EEG data have been preprocessed, so we can directly use "dataloader.py" to load dataset.

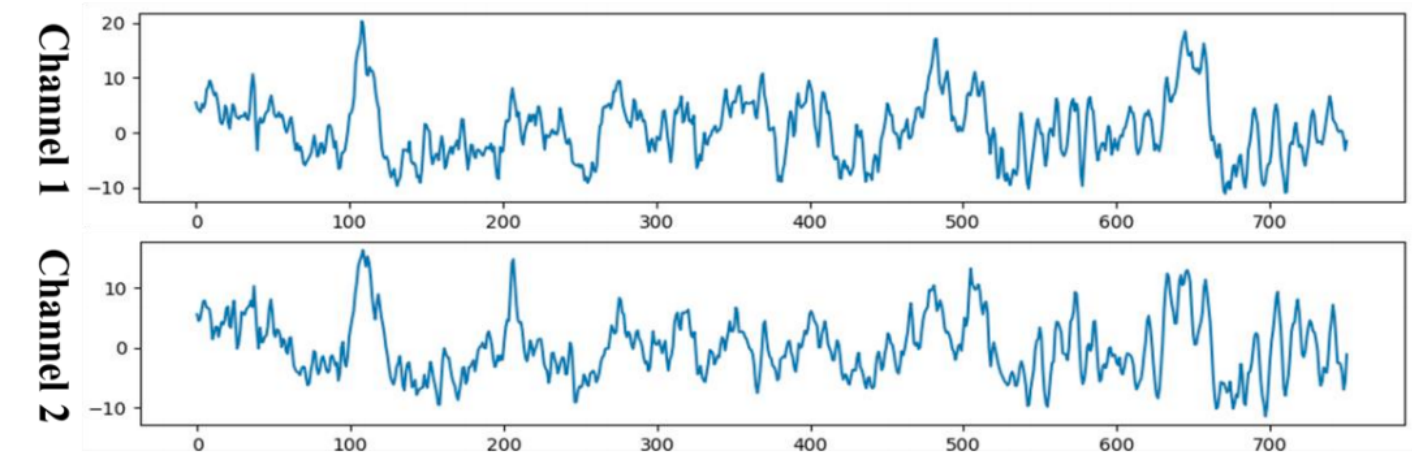

Fig.2 EEG dataset

2. Experiment setups

A. The detail of your model

Activation function select

In order to select different activation function, use dic to control so that I can easily to get acti-vation function. For example, if I want to use nn.ReLU(), only type activate[ReLU], then output nn.ReLU() function.

```
activate = {
    'ReLU' : nn.ReLU(),
    'LeakyReLU' : nn.LeakyReLU(),
    'ELU' : nn.ELU()
}
```

EEGNet

According to Fig.3, I create a EEGNet with pytorch, inherit nn.Model so that I can use it to calcualte gradient easily, the detail code will show below.

```
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=0.25)
  )
  (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=0.25)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

Fig.3 EEGNet details

```
#!/usr/bin/env python3

import torch.nn as nn

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

        activate = {
            'ReLU' : nn.ReLU(),
            'LeakyReLU' : nn.LeakyReLU(),
            'ELU' : nn.ELU()
        }

        # network parameter
        self.firstconv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size = (1,51), stride = (1,1),
            padding = (0, 25), bias = False),
            nn.BatchNorm2d(16, affine = True)
        )

        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size = (2,1), stride = (1,1),
            groups = 16, bias = False),
            nn.BatchNorm2d(32, affine = True),
            activate[activate_function],
            nn.AvgPool2d((1, 4), stride=(1, 4), padding = 0),
            nn.Dropout(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, affine = True),
            activate[activate_function],
            nn.AvgPool2d((1, 8), stride=(1, 8), padding = 0),
            nn.Dropout(0.25)
        )

        self.classify = nn.Sequential(
            nn.Linear(736, 2, bias = True)
        )

    def forward(self,x):

        x = self.firstconv(x)
        x = self.depthwiseConv(x)
        x = self.separableConv(x)
        output = x.view(x.size(0), -1)
        return self.classify(output)
```

DeepConvNet

The DeepConvNet architecture can refer to Fig.4, where C = 2, T = 750 and N = 2. The max norm term is ignorable. Additionally, the stride and padding design a reasonable value make the number of the classify neural is appropriate, the code show below.

Layer	# filters	size	# params	Activation	Options
Input		(C, T)			
Reshape		(1, C, T)			
Conv2D	25	(1, 5)	150	Linear	mode = valid, max norm = 2
Conv2D	25	(C, 1)	25 * 25 * C + 25	Linear	mode = valid, max norm = 2
BatchNorm			2 * 25		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	50	(1, 5)	25 * 50 * C + 50	Linear	mode = valid, max norm = 2
BatchNorm			2 * 50		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	100	(1, 5)	50 * 100 * C + 100	Linear	mode = valid, max norm = 2
BatchNorm			2 * 100		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	200	(1, 5)	100 * 200 * C + 200	Linear	mode = valid, max norm = 2
BatchNorm			2 * 200		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Flatten					
Dense	N			softmax	max norm = 0.5

Fig.4 DeepConvNet architecture

```
#!/usr/bin/env python3

import torch.nn as nn

class DeepConvNet(nn.Module):
    def __init__(self, activate_function):
        super(DeepConvNet, self).__init__()

        activate = {
            'ReLU' : nn.ReLU(),
            'LeakyReLU' : nn.LeakyReLU(),
            'ELU' : nn.ELU()
        }

        # network parameter
        self.block1 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size = (1,5), stride = (1,2), bias = True),
            nn.Conv2d(25, 25, kernel_size = (2,1), bias = True),
            nn.BatchNorm2d(25, affine = True),
```

```
            activate[activate_function],
            nn.MaxPool2d((1,2)),
            nn.Dropout(0.5)
        )

        self.block2 = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size = (1,5), stride = (1,2), bias = True),
            nn.BatchNorm2d(50, affine = True),
            activate[activate_function],
            nn.MaxPool2d((1,2)),
            nn.Dropout(0.5)
        )

        self.block3 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size = (1,5), bias = True),
            nn.BatchNorm2d(100, affine = True),
            activate[activate_function],
            nn.MaxPool2d((1,2)),
            nn.Dropout(0.5)
        )

        self.block4 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size = (1,5), bias = True),
            nn.BatchNorm2d(200, affine = True),
            activate[activate_function],
            nn.MaxPool2d((1,2)),
            nn.Dropout(0.5)
        )

        self.classify = nn.Sequential(
            nn.Linear(1600, 2, bias = True)
        )

    def forward(self,x):

        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        output = x.view(x.size(0), -1)
        return self.classify(output)
```

B. Explain the activation function

The activation function can improve model non-linear property, and increase model generaliza-tion, get more high accuracy. This lab will use ReLU, LeakyReLU and ELU, and I will introduce advantage and disadvantage of these.

ReLU

- advantage

    - The calculation is faster than tanh and Sigmoid, because Relu's mathematical operations are simpler.
    - Avoid and correct the problem of vanishing gradients.
- disadvantage
    - It can only be used in the hidden layer of the neural network model.
    - It may cause the weight to not be updated, because when the input X<0, the gradient will be 0 (Relu characteristic), which will make the weight unable to be adjusted. This is called dying ReLu problem.
    - When the input is greater than 0, the excitation function may blow up.

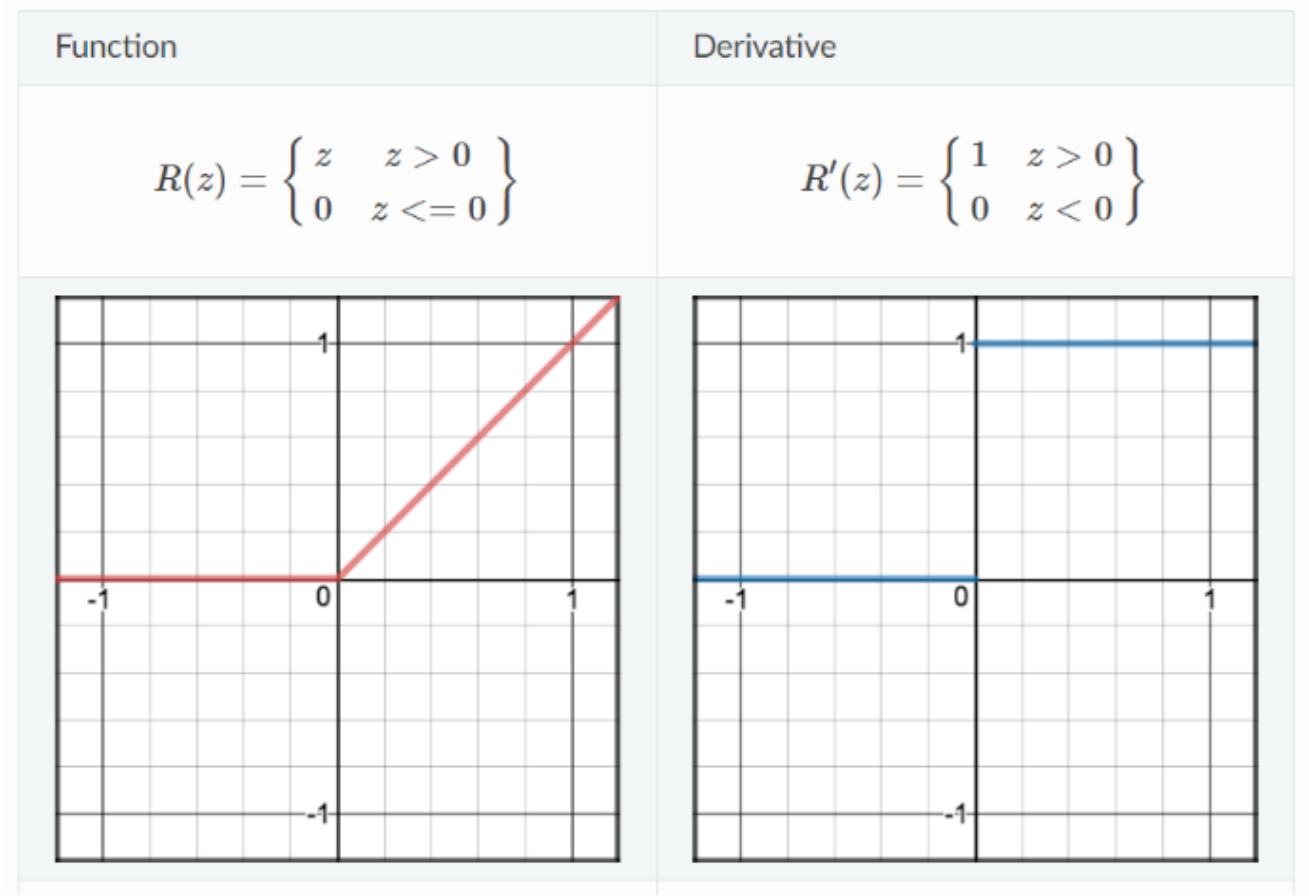

Fig.5 ReLU function and derivative

LeakyReLU

- advantage
    - If x<0 is not 0, the dying ReLU problem can be solved.
- disadvantage
    - When x<0, it is linear, so it cannot be used in complex classification.

- The performance may be lower than Sigmoid and Tanh.

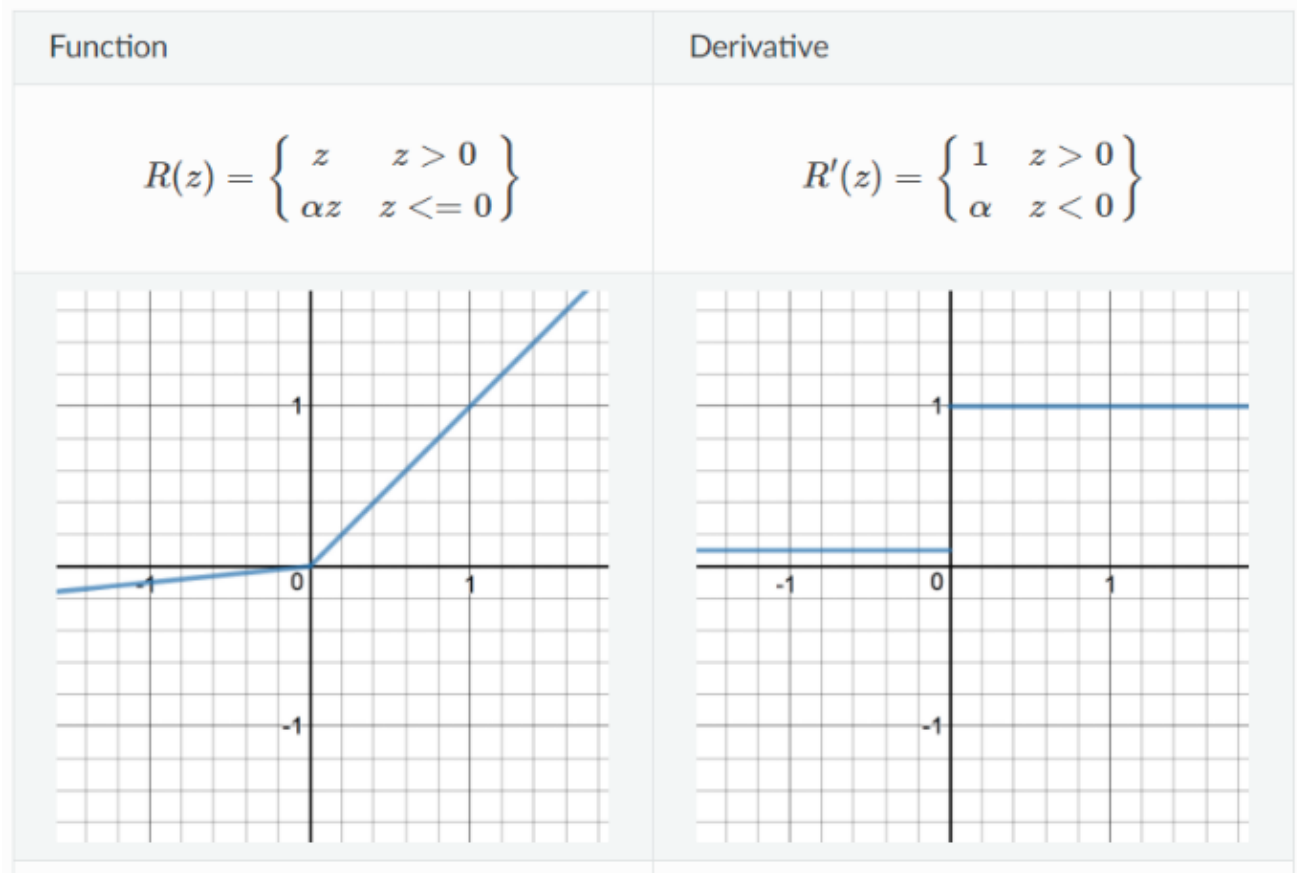

Fig.6 LeakyReLU function and derivative

ELU

- advantage
  - Smooths out more slowly.
  - ELU is a substitute for ReLU and can have negative output.
- disadvantage

- When the input is greater than 0, the excitation function may blow up.

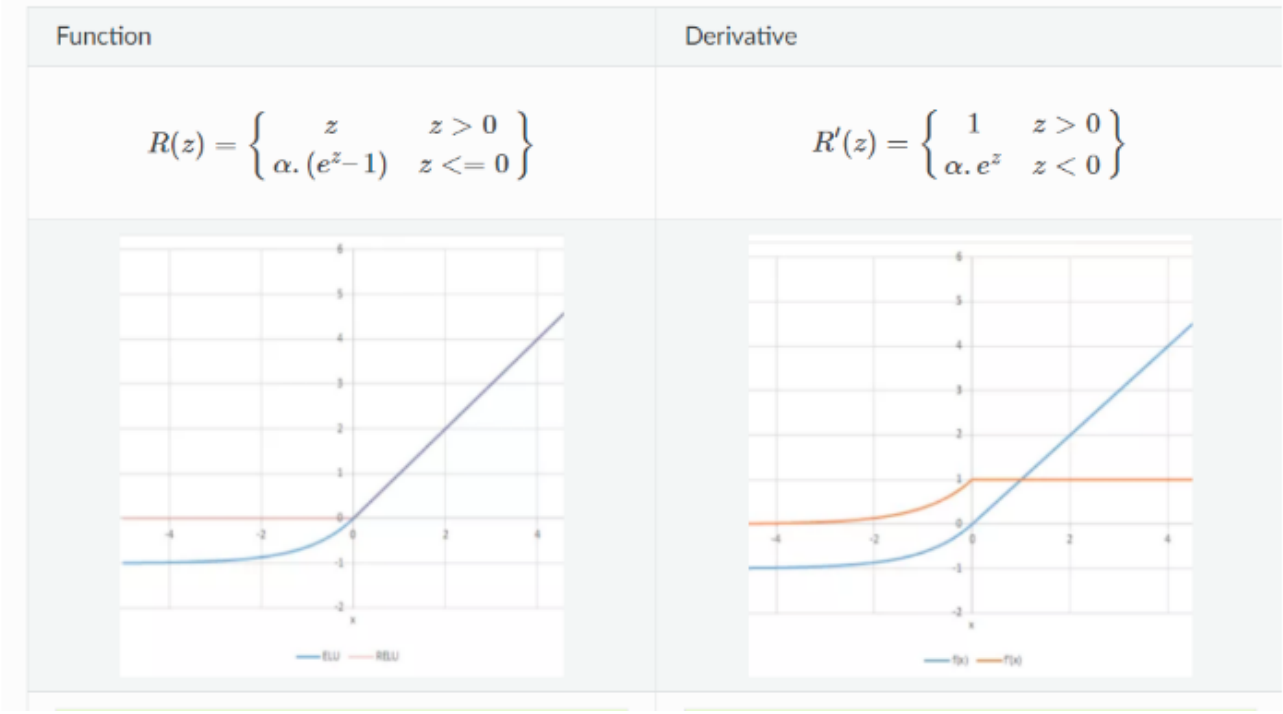

Fig.7 ELU function and derivative

3. Experimental results

A. The highest testing accuracy

Screenshot with two models

```
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): ReLU()
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.25, 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): ReLU()
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.25, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

Fig.8 EEGNet model

```
DeepConvNet(
  (block1): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 2))
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1))
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU()
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.5, inplace=False)
  )
  (block2): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 2))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block3): 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): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (block4): 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): ReLU()
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.5, inplace=False)
  )
  (classify): Sequential(
    (0): Linear(in_features=1600, out_features=2, bias=True)
  )
)
```

Fig.9 DeepConvNet model

EEGNet

- learning rate: 0.0003
- batch size: 16
- epochs: 300

Activation function	ReLU	ELU	LeakyReLU
Accuracy	84.54%	83.52%	**87.22%**

DeepConvNet

- learning rate: 0.0003
- batch size: 16
- epochs: 300

Activation function	ReLU	ELU	LeakyReLU
Accuracy	82.22%	81.67%	82.13%

Different learning rate and batch size

I also adjust different batch size and learning rate to find the highest accuracy, the result and it parameters will show below.

- network: EEGNet
- epochs: 300
- learning rate: 0.0003
- activation function: LeakyReLU

batch size	8	16	32	64	128	256
Accuracy	85.93%	**87.22%**	86.85%	86.20%	85.83%	83.61%

- network: EEGNet
- epochs: 300
- batch size: 16
- activation function: LeakyReLU

learning rate	0.01	0.001	0.0001	0.0003
Accuracy	79.44%	85.09%	83.43%	**87.22%**

Plot comparison result

In order to plot each epoch accuracy during training and testing, I write a visual code to present result. First, input three txt path, ReLU, LeakyReLU and ELU, respectively. Second, read txt file and get train accuracy and test accuracy. Finally, plot the comparison image Fig.10 and Fig.11, and the code show below.

```
#!/usr/bin/env python3

import numpy as np
import argparse
import matplotlib.pyplot as plt

def read(file):
    train_accurancy = []
    test_accurancy = []
    with open(file) as f:
        Lines = f.readlines()
        for line in Lines:
            _, _, tr, te = line.split(",")
            train_accurancy.append((float)(tr.split(":")[1]))
            test_accurancy.append((float)(te.split(":")[1]))

    return np.array(train_accurancy), np.array(test_accurancy)
```

```
def plt_result(ReLU, LeakyReLU, ELU, network_name, save_file):

    plt.title("Activation function comparision({0})".format(network_name),
    fontsize = 18)

    Re_train, Re_test = read(ReLU)
    Le_train, Le_test = read(LeakyReLU)
    ELU_train, ELU_test = read(ELU)

    e = [x for x in range(1,301)]

    plt.plot(e, Re_train, label="ReLU_train")
    plt.plot(e, Re_test, label="ReLU_test")
    plt.plot(e, Le_train, label="LeakyReLU_train")
    plt.plot(e, Le_test, label="LeakyReLU_test")
    plt.plot(e, ELU_train, label="ELU_train")
    plt.plot(e, ELU_test, label="ELU_test")

    plt.legend(loc='lower right')

    plt.xlabel("Epochs")
    plt.ylabel("Accuracy(%)")

    plt.savefig('{0}.png'.format(save_file))
    plt.show()

def create_argparse():
    parser = argparse.ArgumentParser(prog="DLP homework 3",
    description='This code will show activation function result with matpltlib')

    parser.add_argument("ReLU_file", type=str, default="none",
    help="input ReLU file path, default is none")
    parser.add_argument("LeakyReLU_file", type=str, default="none",
    help="input LeakyReLU file path, default is none")
    parser.add_argument("ELU_file", type=str, default="none",
    help="input ELU file path, default is none")
    parser.add_argument("network", type=str, default="none",
    help="input network name(EEGNet or DeepConvNet), default is none")
    parser.add_argument("save_file", type=str, default="none",
    help="save result with this save_file name, default is none")

    return parser

if __name__ == "__main__":

    parser = create_argparse()
    args = parser.parse_args()

    plt_result(args.ReLU_file, args.LeakyReLU_file, args.ELU_file,
    args.network, args.save_file)
```

B. Comparison figures

EEGNet

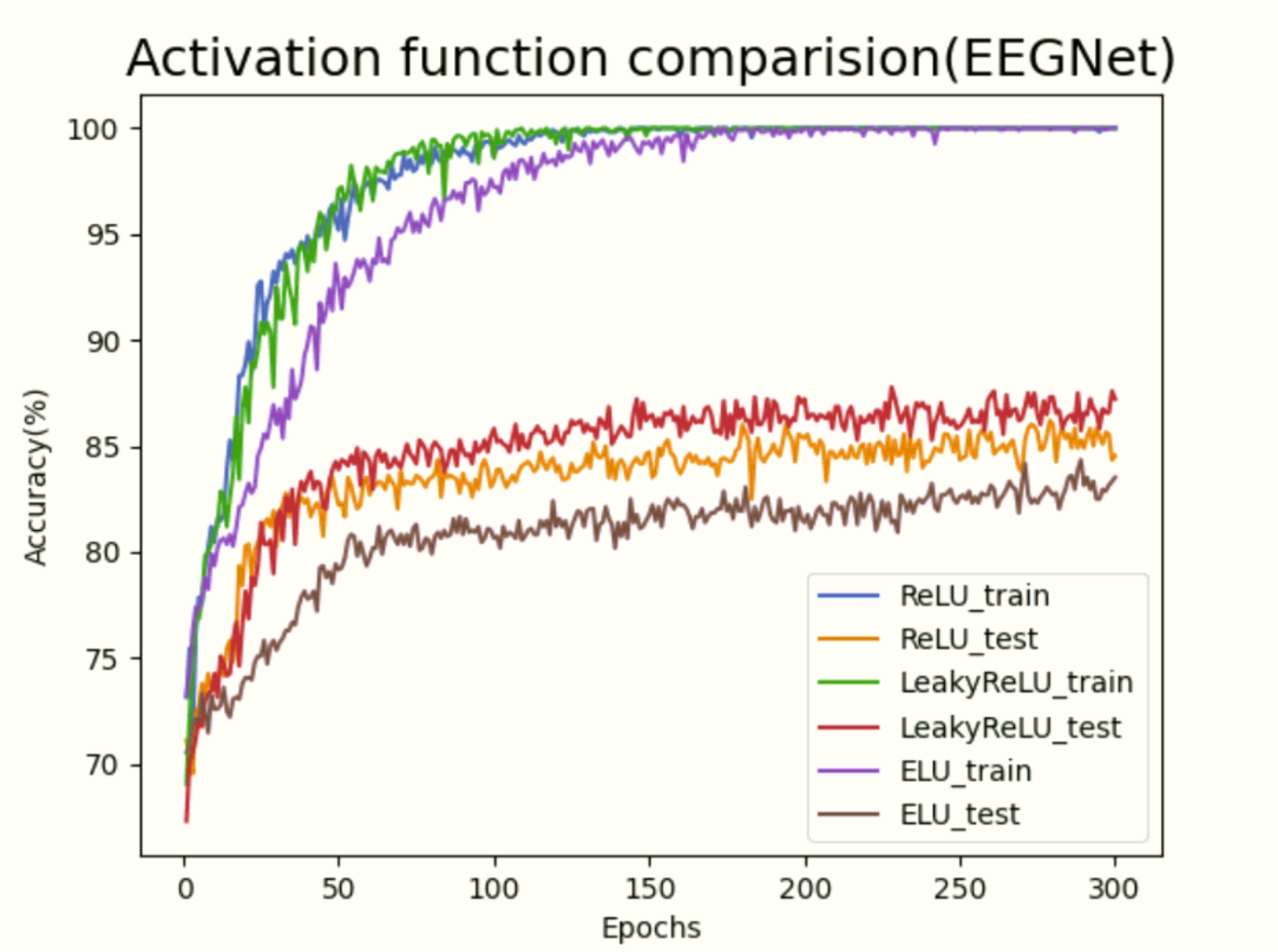

Fig.10 EEGNet activation function comparison

DeepConvNet

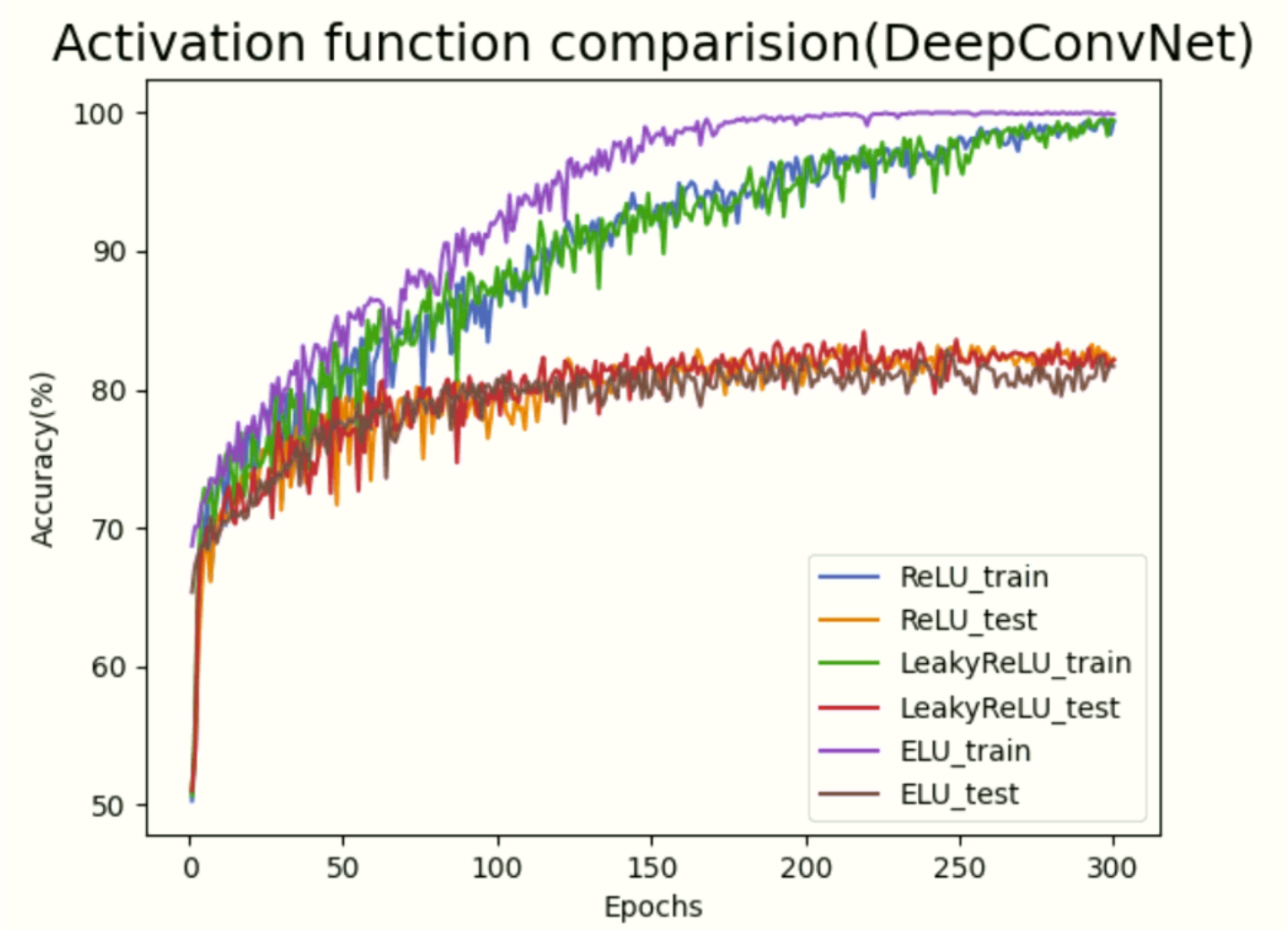

Fig.11 DeepConvNet activation function comparison

4. Discussion

A. Virtual environment

I create the virtual environment for this lab by used virtualenv, and write the command into script "install.sh", and required libraries in requirements.txt, the code show below. And then, I can use DLP_homework3 to activate this virtual environment, and use deactivate to exit this virtual environment.

```
#!/usr/bin/env bash

echo "Install required libraries"
pip3 install virtualenv

virtualenv_name="DLP_homework3"
VIRTUALENV_FOLDER=$(pwd)/${virtualenv_name}
virtualenv ${virtualenv_name}

source ${VIRTUALENV_FOLDER}/bin/activate
python3 -m pip install -r requirements.txt
```

```
deactivate
echo "alias DLP_homework3='source ${VIRTUALENV_FOLDER}/bin/activate '" >> ~/.bashrc
source ~/.bashrc
```

In the requirements.txt.

```
torch==1.8.1
torchvision==0.9.1
torchaudio==0.8.1
matplotlib==3.3.4
tqdm==4.60.0
```

B. Dataset

I rewrite the dataloader.py, and inherit torch.utils.daata.Dataset to customize our data, let I can easy use DataLoader to train or test data.

```
#!/usr/bin/env python3

import torch
import numpy as np
from torch.utils.data.dataset import Dataset

class EEGDataset(Dataset):
    def __init__(self, mode):

        if(mode == "train"):
            S4b = np.load('./dataset/S4b_train.npz')
            X11b = np.load('./dataset/X11b_train.npz')
        elif(mode == "test"):
            S4b = np.load('./dataset/S4b_test.npz')
            X11b = np.load('./dataset/X11b_test.npz')
        else:
            raise Excption("Error! Please input train or test")

        data = np.concatenate((S4b['signal'], X11b['signal']), axis=0)
        label = np.concatenate((S4b['label'], X11b['label']), axis=0) - 1

        data = np.transpose(np.expand_dims(data, axis=1), (0, 1, 3, 2))

        mask = np.where(np.isnan(data))
        data[mask] = np.nanmean(data)

        self.data, self.label = torch.from_numpy(data).float(),torch.from_numpy(label)

    def __getitem__(self, index):

        return self.data[index], self.label[index]

    def __len__(self):
```

```
        return self.data.shape[0]
```

C. Model select

In order to select different networkt, I use same approach(Activation function select) to access model, and write this function in "*_init_*.py" of model folder.

```
#!/usr/bin/env python3

from .DeepConvNet import DeepConvNet
from .EEGNet import EEGNet

def get_model(name, activate_function):
    models = {
        'EEGNet' : EEGNet(activate_function),
        'DeepConvNet' : DeepConvNet(activate_function)
    }

    return models[name]
```

D. Train and evaluation

I write training and evaluation function in Trainer class, trainer will load dataset, setting parameter, create network and so on, I also write plt_lr_cur and recore to plot learning curve and record each epoch accuracy and loss. In order to demo, I write save model weight and load model weight, if use load model weight, it will evaluate test dataset, and show test accuracy.

```
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
import os

import torch
from torch.utils.data import DataLoader
from torch.autograd import Variable
import torch.nn as nn

from dataset.dataset import EEGDataset
from model import get_model

class Trainer():
    def __init__(self,args):
        self.args = args
```

```
        # data
        trainset = EEGDataset("train")
        testset = EEGDataset("test")

        # file name
        self.file_name ="{0}_{1}_{2}_{3}_{4}".format(args.network, args.activate_funct

        # crate folder
        self.weight_path = os.path.join(args.save_folder,"weight")
        if not os.path.exists(self.weight_path):
            os.makedirs(self.weight_path)

        self.picture_path = os.path.join(args.save_folder,"picture")
        if not os.path.exists(self.picture_path):
            os.makedirs(self.picture_path)

        self.record_path = os.path.join(args.save_folder,"record")
        if not os.path.exists(self.record_path):
            os.makedirs(self.record_path)

        # dataloader
        self.trainloader = DataLoader(dataset=trainset, batch_size=args.batch_size,
        shuffle=True, num_workers=2)
        self.testloader = DataLoader(dataset=testset, batch_size=args.batch_size,
        shuffle=True, num_workers=2)

        # model
        self.model = get_model(args.network, args.activate_function)

        # show model parameter
        print(self.model)

        # optimizer
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr = args.learning_

        # criterion
        self.criterion = nn.CrossEntropyLoss()

        # using cuda
        if args.cuda:
            self.model = self.model.cuda()
            self.criterion = self.criterion.cuda()
            print("using cuda")

        # load model if need
        if(args.load_path != None):
            if os.path.exists(args.load_path):
                self.model.load_state_dict(torch.load(args.load_path))
                accuracy = self.evaluate(self.testloader)
                print('Test Accuracy: %2.2f%%' % (accuracy))
            else:
                raise("This path is not exist!")

    # training
```

```
    def training(self):

        self.loss_record = []
        for e in range(self.args.epochs):
            train_loss = 0.0
            tbar = tqdm(self.trainloader)
            self.model.train()
            for i, (data, label) in enumerate(tbar):
                data, label = Variable(data),Variable(label)

                # using cuda
                if self.args.cuda:
                    data, label = data.cuda(), label.cuda()

                prediction = self.model(data)
                loss = self.criterion(prediction, label.long())

                loss.backward()
                self.optimizer.step()
                train_loss += loss.item()
                self.optimizer.zero_grad()

                tbar.set_description('Train loss: {0:.6f}'.format(train_loss / (i + 1)

            self.loss_record.append(train_loss / (i + 1))
            tr_ac = self.evaluate(self.trainloader)
            te_ac = self.evaluate(self.testloader)
            self.record(e+1, train_loss / (i + 1), tr_ac, te_ac)

        print('Train Accuracy: %2.2f%%' % (tr_ac))
        print('Test Accuracy: %2.2f%%' % (te_ac))

        # save model
        torch.save(self.model.state_dict(), os.path.join(self.weight_path, self.file_n

        # plot learning curve
        self.plt_lr_cur()

    # evaluation
    def evaluate(self, d):

        correct = 0.0
        total = 0.0
        self.model.eval()
        for i, (data, label) in enumerate(d):
            data, label = Variable(data),Variable(label)

            # using cuda
            if self.args.cuda:
                data, label = data.cuda(), label.cuda()

            with torch.no_grad():
                prediction = self.model(data)
                pred = prediction.data.max(1, keepdim=True)[1]
                correct += np.sum(np.squeeze(pred.eq(label.data.view_as(pred))).cpu().
```

```
                total += data.size(0)

        # print('Accuracy: %2d%% (%2d/%2d)' % (100. * correct / total, correct, total)
        return 100.0 * correct / total

    # plot learning curve
    def plt_lr_cur(self):

        plt.title("learning curve", fontsize = 18)
        ep = [x for x in range(1, self.args.epochs + 1)]
        plt.xlabel("epochs")
        plt.ylabel("loss")
        plt.plot(ep, self.loss_record)
        plt.savefig(os.path.join(self.picture_path, self.file_name + '_learning_curve.
        # plt.show()

    # record information
    def record(self, epochs, loss, tr_ac, te_ac):

        file_path = os.path.join(self.record_path, self.file_name + '.txt')
        with open(file_path, "a") as f:
            f.writelines("Epochs : {0}, train loss : {1:.6f}, train accurancy : {2:.2f
```

E. Argparse

In order to easy select different model hyper-parameters, I use argparse make user can modify own parameters, the code write in option.py.

```
#!/usr/bin/env python3

import argparse
import torch
import os

class Option():
    def __init__(self):
        parser = argparse.ArgumentParser(prog="DLP homework 3",
        description='This lab implement EEGNet and DeepConvNet with pytorch')

        # training hyper parameters
        parser.add_argument("--learning_rate", type=float, default=0.01,
        help="Learning rate for the trainer, default is 1e-2")
        parser.add_argument("--epochs", type=int, default=300,
        help="The number of the training, default is 300")
        parser.add_argument("--batch_size", type=int, default=64,
        help="Input batch size for training, default is 64")

        # save name and load model path
        parser.add_argument("--save_folder", type=str, default=os.getcwd(),
        help="save model in save folder, default is current path")
        parser.add_argument("--load_path", type=str, default=None,
```

```
            help="path to load model weight")

            # cuda
            parser.add_argument('--no-cuda', action='store_true', default=False,
            help='disables CUDA training, default is False')

            # network
            parser.add_argument("--network", type=str, default="EEGNet",
            help="select network EEGNet or DeepConvNet, default is EEGNet")

            # activation function
            parser.add_argument("--activate_function", type=str, default="ReLU",
            help="activate function (ReLU, LeakyReLU, ELU)for network, default is ReLU")
            self.parser = parser

        def create(self):

            args = self.parser.parse_args()
            args.cuda = not args.no_cuda and torch.cuda.is_available()
            print(args)
            return args
```

F. Code architecture

In this lab, I code archtitecture show below, the dataset in charge of data proccessing, load data, combine ground truth and label. The network will write on model folder, including network architecture and forward. Picture, record and weight folder will save each training result. "main.py" is main code, and excute it start training.

homework3
├── dataset
│ ├── dataset.py
│ ├── S4b_test.npz
│ ├── S4b_train.npz
│ ├── X11b_test.npz
│ └── X11b_train.npz
├── install.sh
├── main.py
├── model
│ ├── DeepConvNet.py
│ ├── EEGNet.py
│ └── *_init_*.py
├── option.py
├── picture
├── record
├── requirements.txt
├── trainer.py

```
├── visual.py
└── weight
```

```python
#!/usr/bin/env python3

from option import Option
from trainer import Trainer

if __name__ == '__main__':

    # paramters
    args = Option().create()

    # trainer
    trainer = Trainer(args)

    if(args.load_path == None):
        trainer.training()
```

tags: **DLP2021**