DLP Homework 4

電控碩 0860077 王國倫

1. Introduction

In this lab, we will implement the ResNet18, ResNet50 architecture and load parameters from a pretrained model to analysis diabetic retinopathy, create own dataloader to provide model train, compare and visualize the accuracy between the pretrained model and without pretrained, plot their confusion matrix.

2. Experiment setups

A. The details of your model (ResNet)

The ResNet18 and ResNet50 are available from torchvision models, so I use these to train diabe-tic retinopathy data and redesigned these, add fully connected and activation function on the last layers, below is my ResNet18 and ResNet50.

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

import torch.nn as nn
from torchvision import models

class ResNet18(nn.Module):
    def __init__(self, num_class, feature_extract, use_pretrained):
        super(ResNet18,self).__init__()

        self.model = models.resnet18(pretrained=use_pretrained)
        set_parameter_requires_grad(self.model, feature_extract)
        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Linear(num_ftrs, 256)
        self.activate = nn.LeakyReLU()
        self.fc = nn.Linear(256, num_class)

    def forward(self, x):
        x = self.model(x)
        x = self.activate(x)
        out = self.fc(x)
        return out

class ResNet50(nn.Module):
    def __init__(self, num_class, feature_extract, use_pretrained):
        super(ResNet50,self).__init__()
```

```
        self.model = models.resnet50(pretrained=use_pretrained)
        set_parameter_requires_grad(self.model, feature_extract)
        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Linear(num_ftrs, 512)
        self.activate = nn.LeakyReLU()
        self.fc = nn.Linear(512, 256)
        self.fc2 = nn.Linear(256, num_class)

    def forward(self, x):
        x = self.model(x)
        x = self.activate(x)
        x = self.fc(x)
        x = self.activate(x)
        out = self.fc2(x)
        return out

def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

B. The details of your Dataloader

I create RetinopathyDataset by myself, and write some useful function, like calculate weight loss, transform and download data. The purpose of weight loss is handle imbalanced data, because this lab provided data is very imbalanced data, the "0" class is accounted for 70%. The transform make image to be processed, including rotate, normalize, convert tensor and so on. You can use download data function, download data to dataset folder, and get the dataset path.

```
#!/usr/bin/env python3
import pandas as pd
import numpy as np
from PIL import Image
import os
from torch.utils.data.dataset import Dataset
import torchvision.transforms as transforms
import gdown
from zipfile import ZipFile
import shutil

class RetinopathyDataset(Dataset):
    def __init__(self, mode):
        """
        Args:
            mode : Indicate procedure status(training or testing)
            self.img_name (string list): String list that store all image names.
            self.label (int or float list): Numerical list that store all
            ground truth label values.
        """
```

```
        self.__downloaddata()
        self.img_name, self.label = self.__getData(mode)
        self.trans = self.__trans(mode)
        self.weight_loss = self.__cal_we_loss()
        print("> Found %d images..." % (len(self.img_name)))

    def __len__(self):
        """'return the size of dataset"""

        return len(self.img_name)

    def __getitem__(self, index):

        # read rgb images
        rgb_image = Image.open(os.path.join(self.root, self.img_name[index] +
        ".jpeg")).convert('RGB')

        # convert images
        img = self.trans(rgb_image)
        label = self.label[index]

        return img, label

    def __cal_we_loss(self):

        weight = [0 for x in range(len(set(self.label)))]
        for i in range(len(weight)):
            weight[i] = self.label[self.label == i].size

        return np.true_divide(np.max(weight), weight)


    def __trans(self, mode):

        if mode == "train":
            transform = transforms.Compose([
            transforms.RandomVerticalFlip(),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]),
        ])
        else:
            transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]),
        ])

        return transform


    def __downloaddata(self):
```

```
        # download dataset and extract zip to current path
        model_url = 'https://drive.google.com/u/1/uc?id=1RTmrk7Qu9IBjQYLczaYKOvXaHWBS0
        model_name = 'RetinopathyDataset'

        if not os.path.isdir(os.path.join(os.getcwd(),"dataset",model_name)):
            gdown.download(model_url, output=model_name + '.zip', quiet=False)
            zip1 = ZipFile(model_name + '.zip')
            zip1.extractall(model_name)
            zip1.close()

            # move folder
            shutil.move(os.path.join(os.getcwd(),model_name),
            os.path.join(os.getcwd(),"dataset",model_name))

            # delete zip file
            os.remove(os.path.join(os.getcwd(),model_name + '.zip'))

        self.root = os.path.join(os.getcwd(), "dataset", model_name, "data")

    def __getData(self, mode):

        if mode == 'train':
            img = pd.read_csv('./dataset/train_img.csv', header = None)
            label = pd.read_csv('./dataset/train_label.csv', header = None)
            return np.squeeze(img.values), np.squeeze(label.values)
        elif mode == "test":
            img = pd.read_csv('./dataset/test_img.csv', header = None)
            label = pd.read_csv('./dataset/test_label.csv', header = None)
            return np.squeeze(img.values), np.squeeze(label.values)
        else:
            raise Excption("Error! Please input train or test")
```

C. Describing your evaluation through the confusion matrix

The confusion matrix can help me understand real result between predicted data and ground truth data, we can see Fig.1 and Fig.2, the ideal result is the higher the diagonal value, the model is better, so the Fig.2 is better than Fig.1. The plot confusion matrix code will show below.

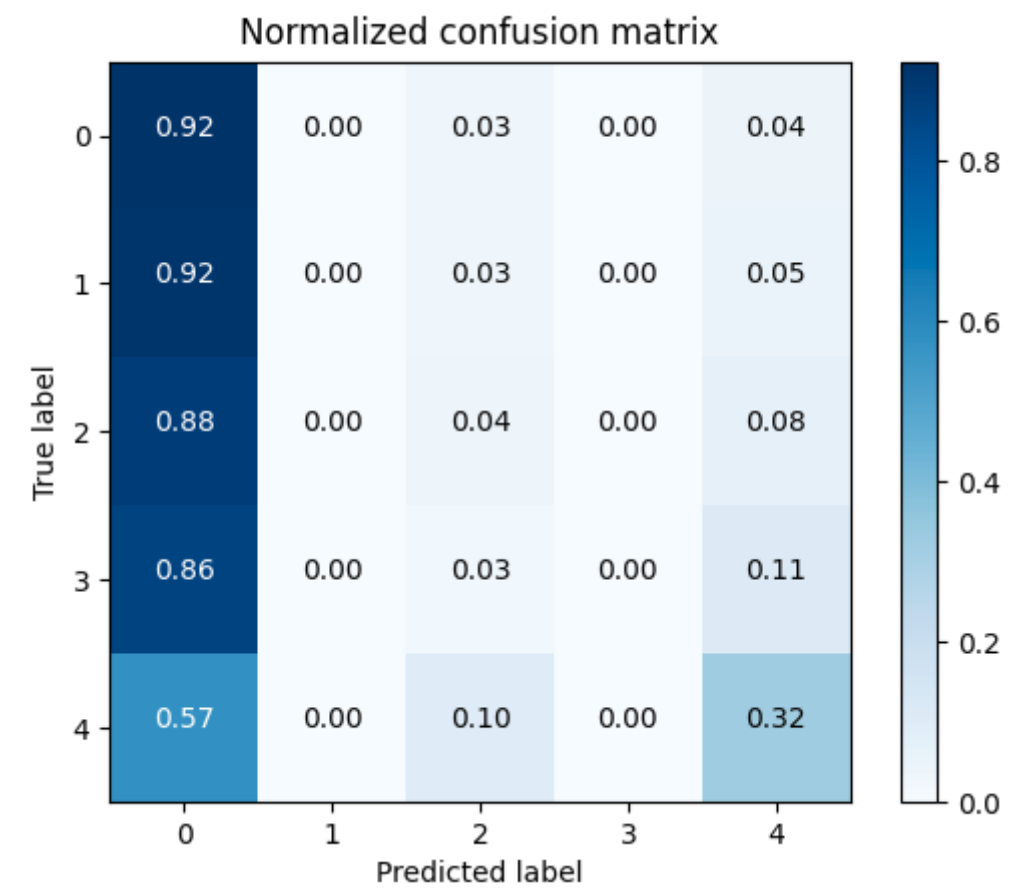

Fig.1 ResNet50 without pretrained and weight loss

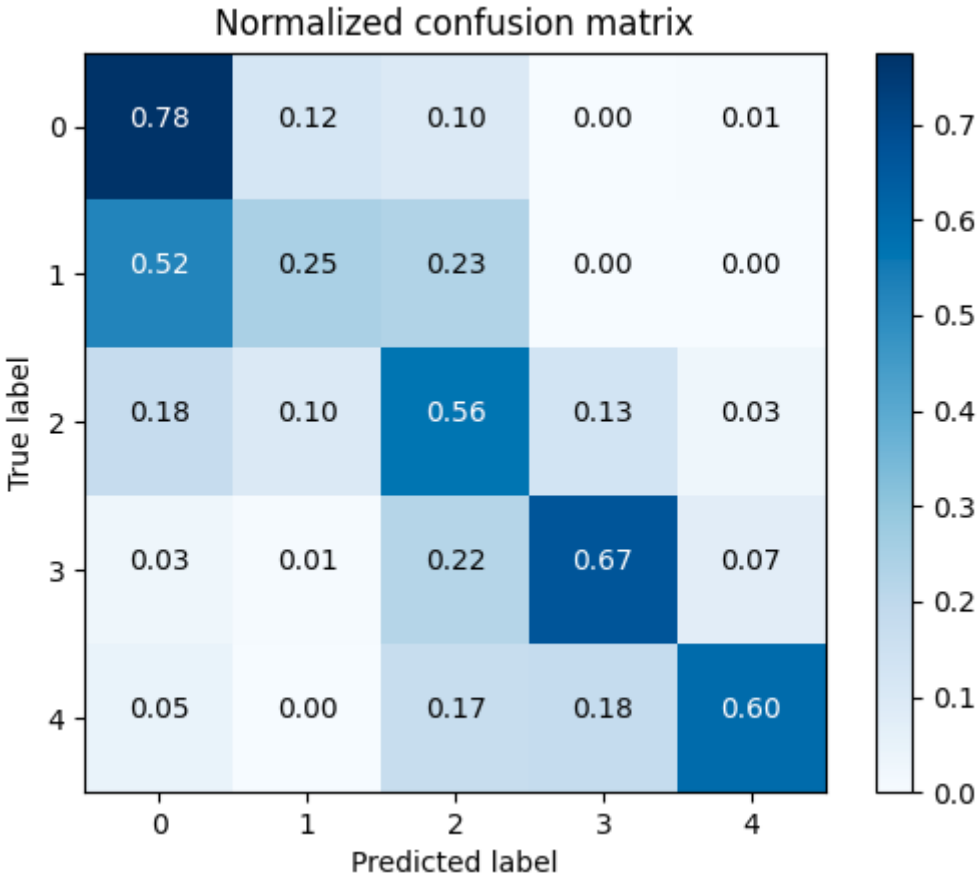

Fig.2 ResNet50 with pretrained and weight loss

```
def plot_confusion_matrix(self, num_class):

        matrix = np.zeros((num_class,num_class))
```

```
        self.model.eval()
        for i, (data, label) in enumerate(self.testloader):
            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]

                ground_truth = pred.cpu().numpy().flatten()
                actual = label.cpu().numpy().astype('int')

                for j in range(len(ground_truth)):
                    matrix[actual[j]][ground_truth[j]] += 1

        for i in range(num_class):
            matrix[i,:] /=  sum(matrix[i,:])

        plt.figure(1)
        plt.imshow(matrix, interpolation="nearest", cmap=plt.cm.Blues)
        plt.title("Normalized confusion matrix")
        plt.colorbar()

        thresh = np.max(matrix) / 2.
        for i, j in itertools.product(range(matrix.shape[0]), range(matrix.shape[1])):
            plt.text(j, i, format(matrix[i, j], '.2f'),
                 horizontalalignment="center",
                 color="white" if matrix[i, j] > thresh else "black")

        tick_marks = np.arange(num_class)
        plt.xticks(tick_marks)
        plt.yticks(tick_marks)

        plt.ylabel('True label')
        plt.xlabel('Predicted label')

        plt.savefig(os.path.join(self.picture_path,
        self.file_name + '_confusion_matrix.png'))
```

3. Experimental results

A. The highest testing accuracy

Screenshot

Fig.3 ResNet50 with pretrained and no weight loss

Fig.4 ResNet18 with pretrained and no weight loss

Weight loss

The provided dataset is imbalanced data, so I add weight loss to cross entropy, try to get the best result. Although the accuracy is not much change, the confusion matrix can present is good(you can see Fig.5 and Fig.6), these model has theirs advantages and disadvantages, depend on the case which you want to use.

> ResNet18

- learning rate: 0.001
- batch size: 16
- epochs: 20
- with pretrained

weight loss	with	without
Accuracy	76.74%	**82.14%**

> ResNet18

- learning rate: 0.001
- batch size: 16
- epochs: 20
- without pretrained

weight loss	with	without
Accuracy	65.81%	75.23%

> ResNet50

- learning rate: 0.001
- batch size: 8
- epochs: 20
- with pretrained

weight loss	with	without

weight loss	with	without
Accuracy	78.61%	**82.21%**

> ResNet50

- learning rate: 0.001
- batch size: 8
- epochs: 20
- without pretrained

weight loss	with	without
Accuracy	73.36%	73.36%

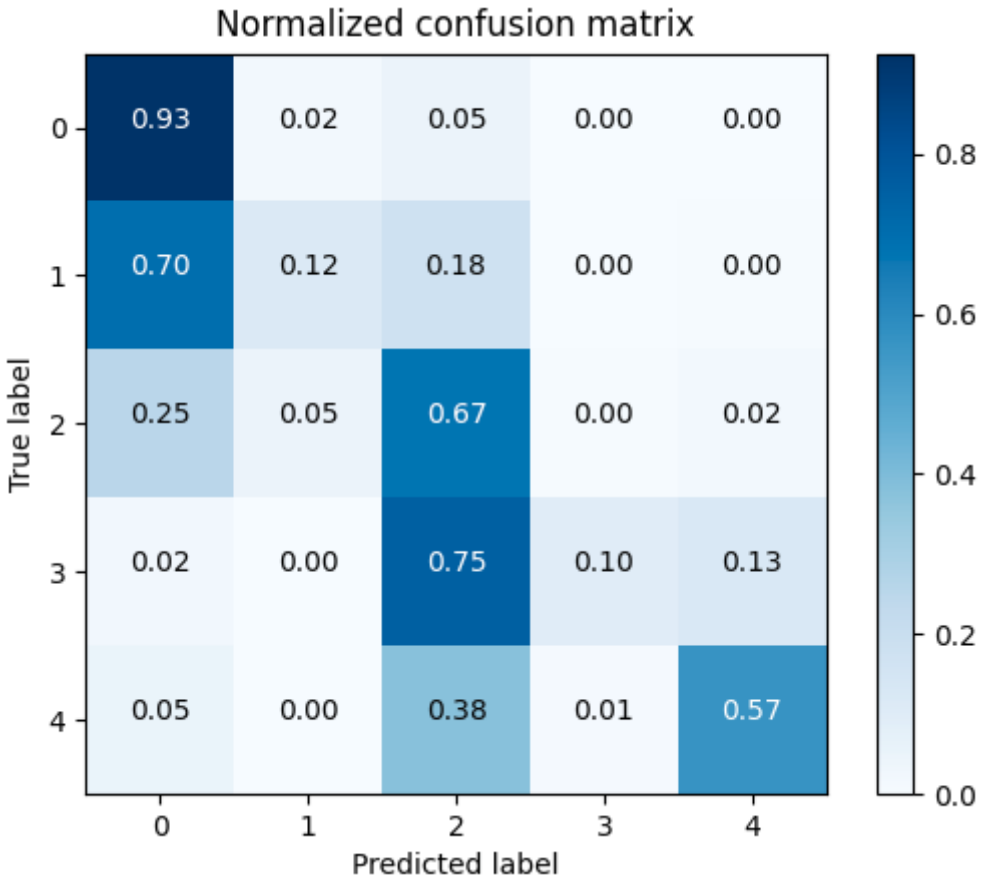

Fig.5 ResNet50 with pretrained and without weight loss

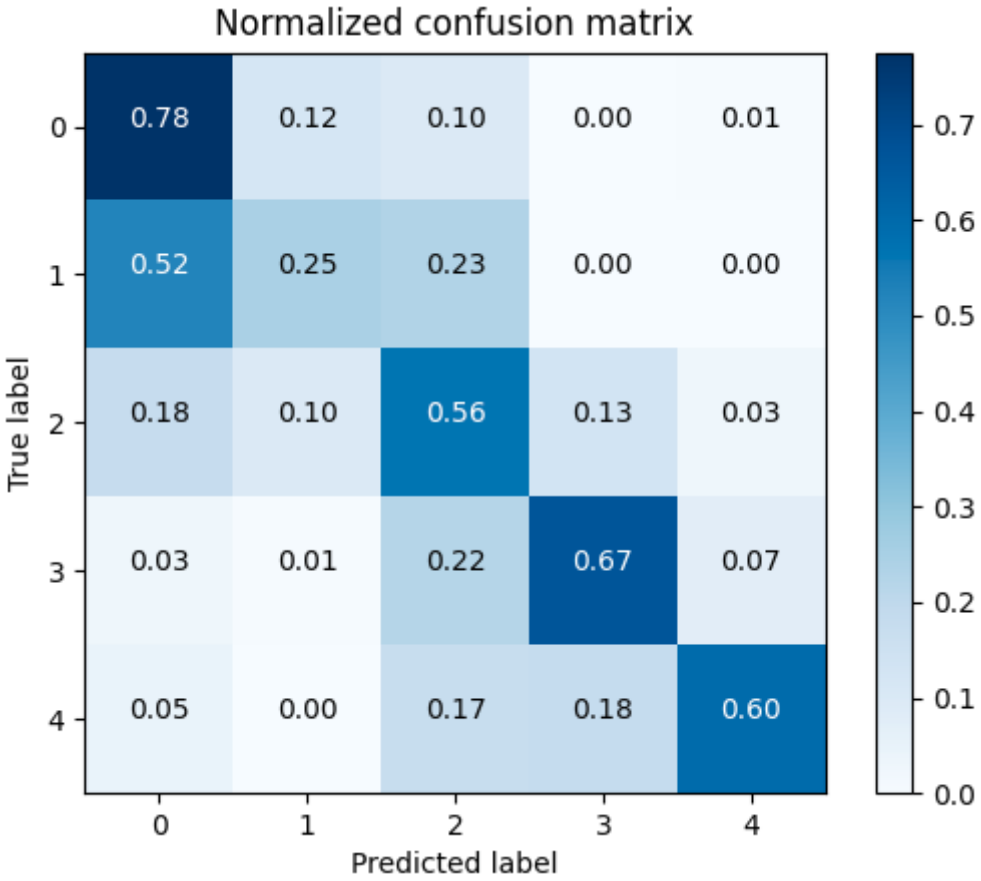

Fig.6 ResNet50 with pretrained and with weight loss

finetune and feature extract

First I adjust feature extract and finetune to see result, I use ResNet18 with pretrained and without weight loss, the other parameters are same, the result show below, use feature extract is easy to overfitting so that the other experiment will adopt finetune mode to train.

> ResNet18

- learning rate: 0.001
- batch size: 16
- epochs: 20

	finetune	**feature extract**
Accuracy	82.14%	75.58%

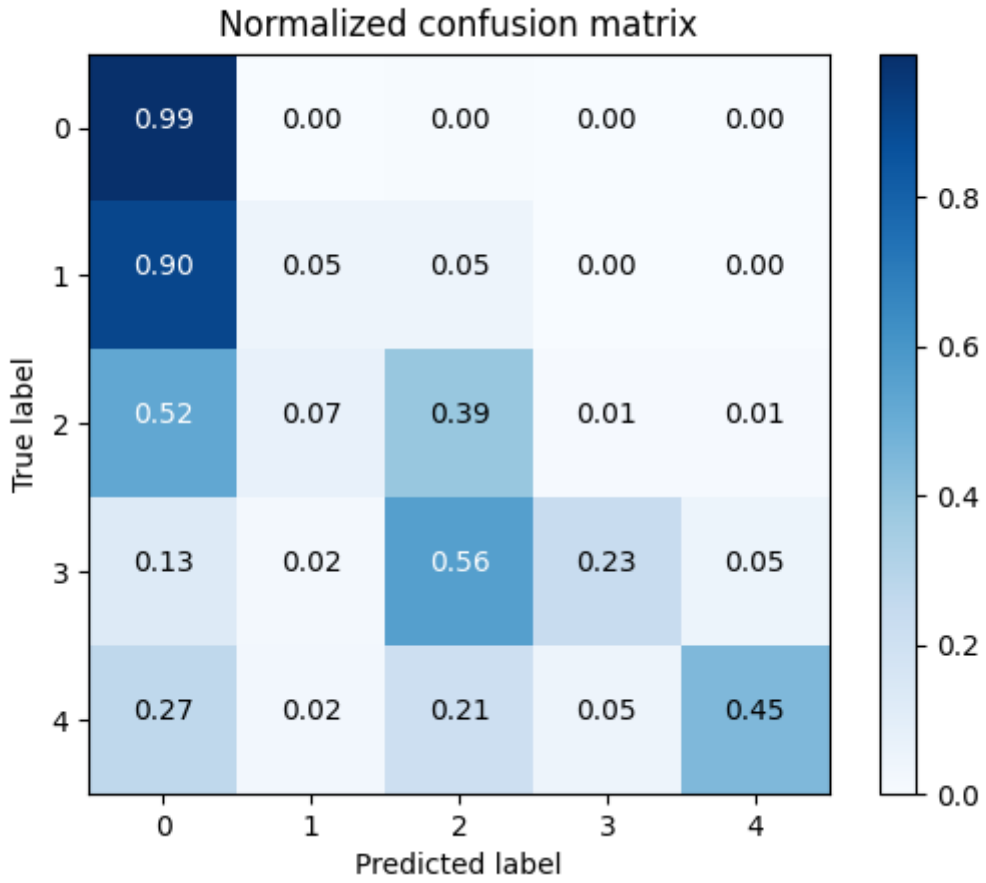

Fig.7 ResNet18 finetune

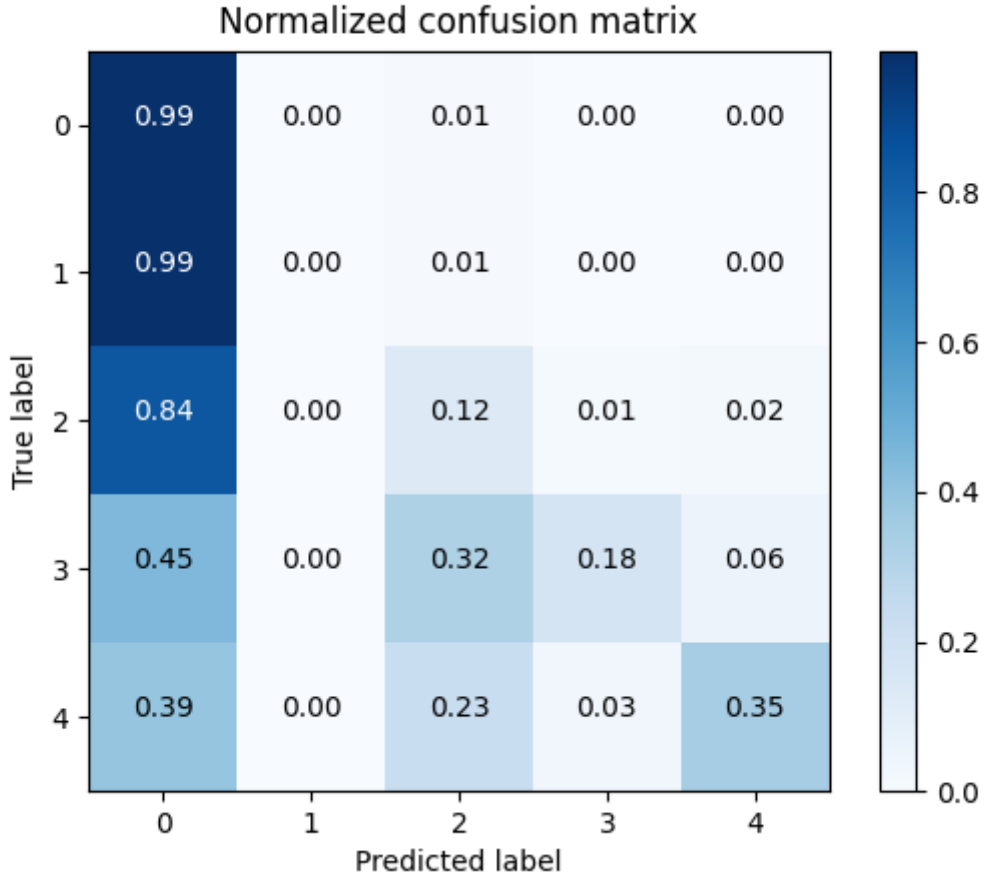

Fig.8 ResNet18 feature extract

Save the best model

In order to save the best model, I will store the best parameters in the training, on the other hand, the test accuracy is flow, so the final result maybe not the best, I add it to ensure each training will store the best model so that I can easily keep training next time or show test accuracy.

```
if(te_ac > best_accuracy):
   best_accuracy = te_ac
   best_model_weight = copy.deepcopy(self.model.state_dict())

   # save the best model
   torch.save(best_model_weight, os.path.join(self.weight_path,
   self.file_name + '.pkl'))
```

B. Comparison figures

Plotting the comparison figures

This lab will comparison use pretrained or not, I also add weight loss to comparison, so there will four pictures below, and code show below the picture.

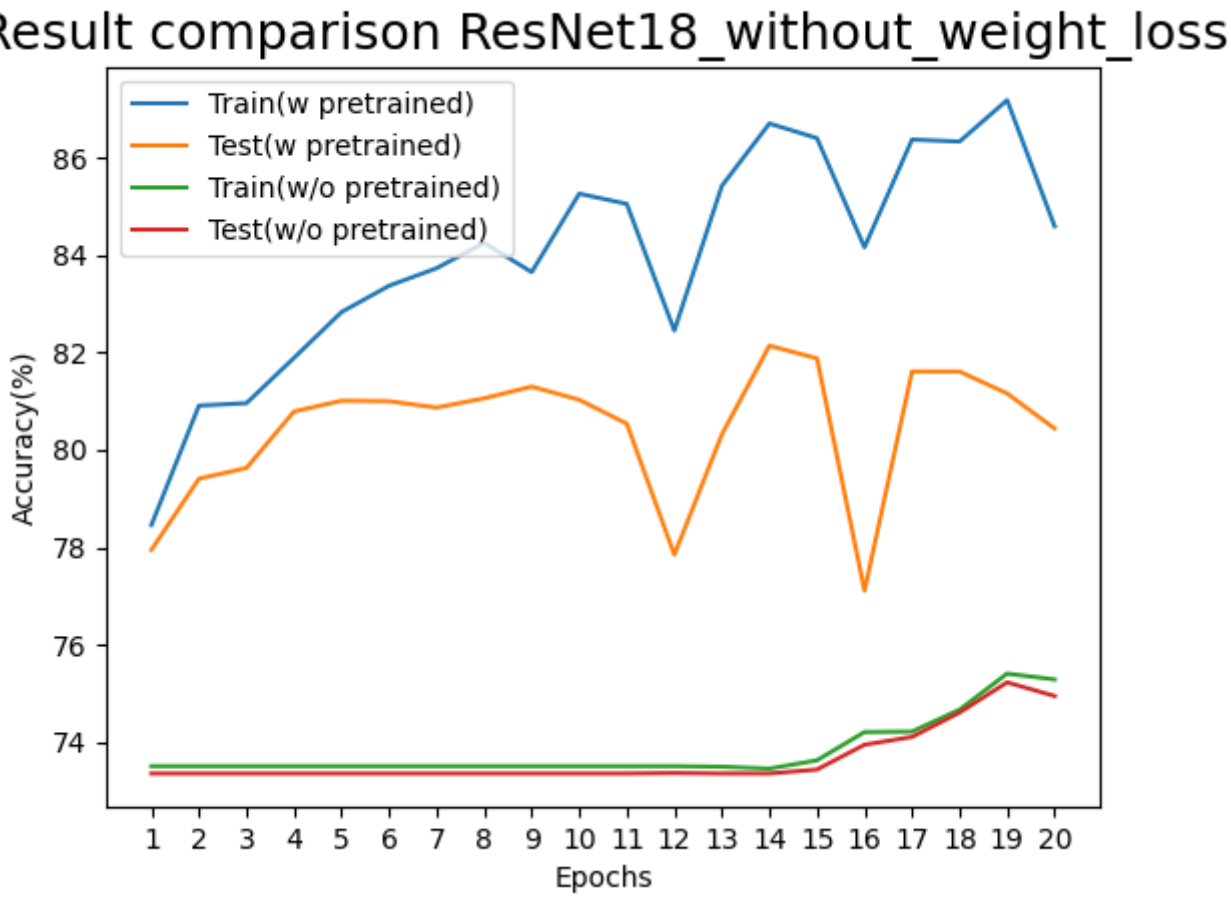

Fig.9 ResNet18 without weight loss

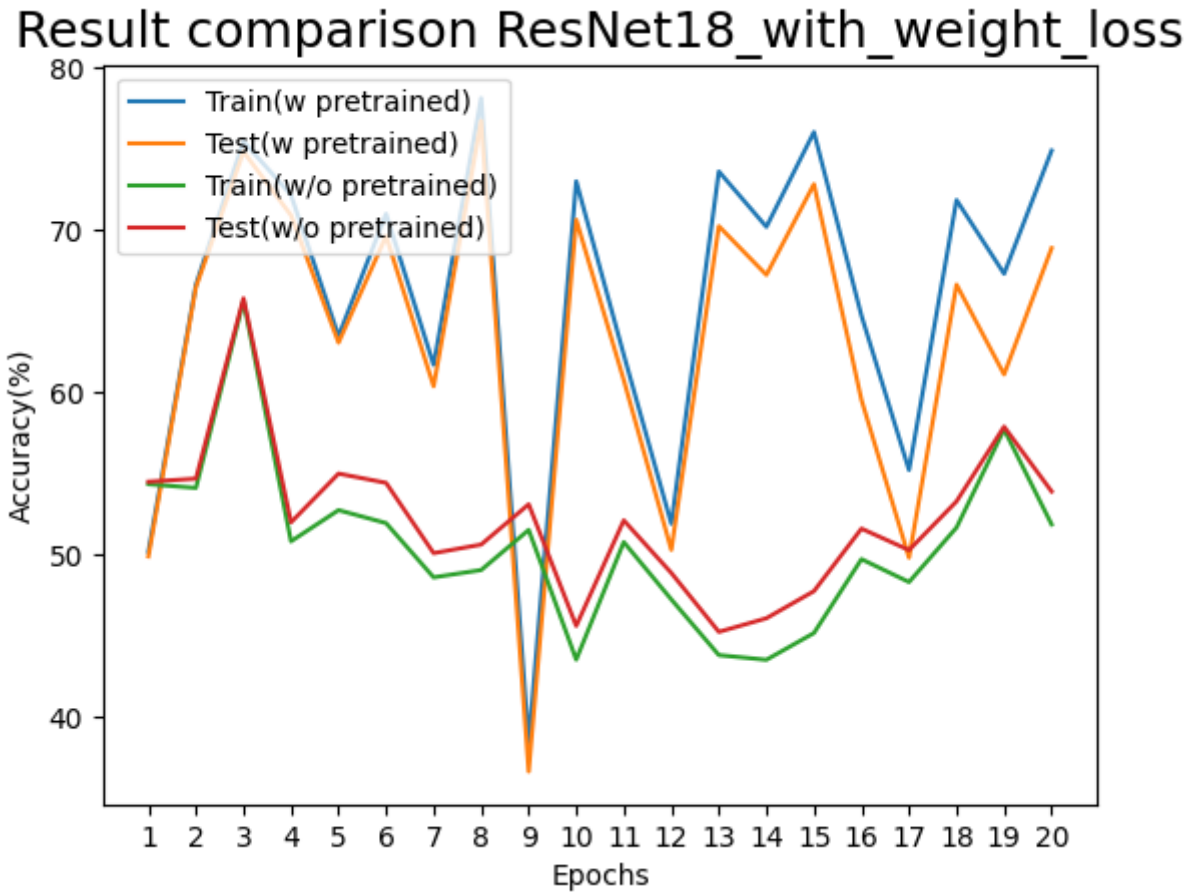

Fig.10 ResNet18 with weight loss

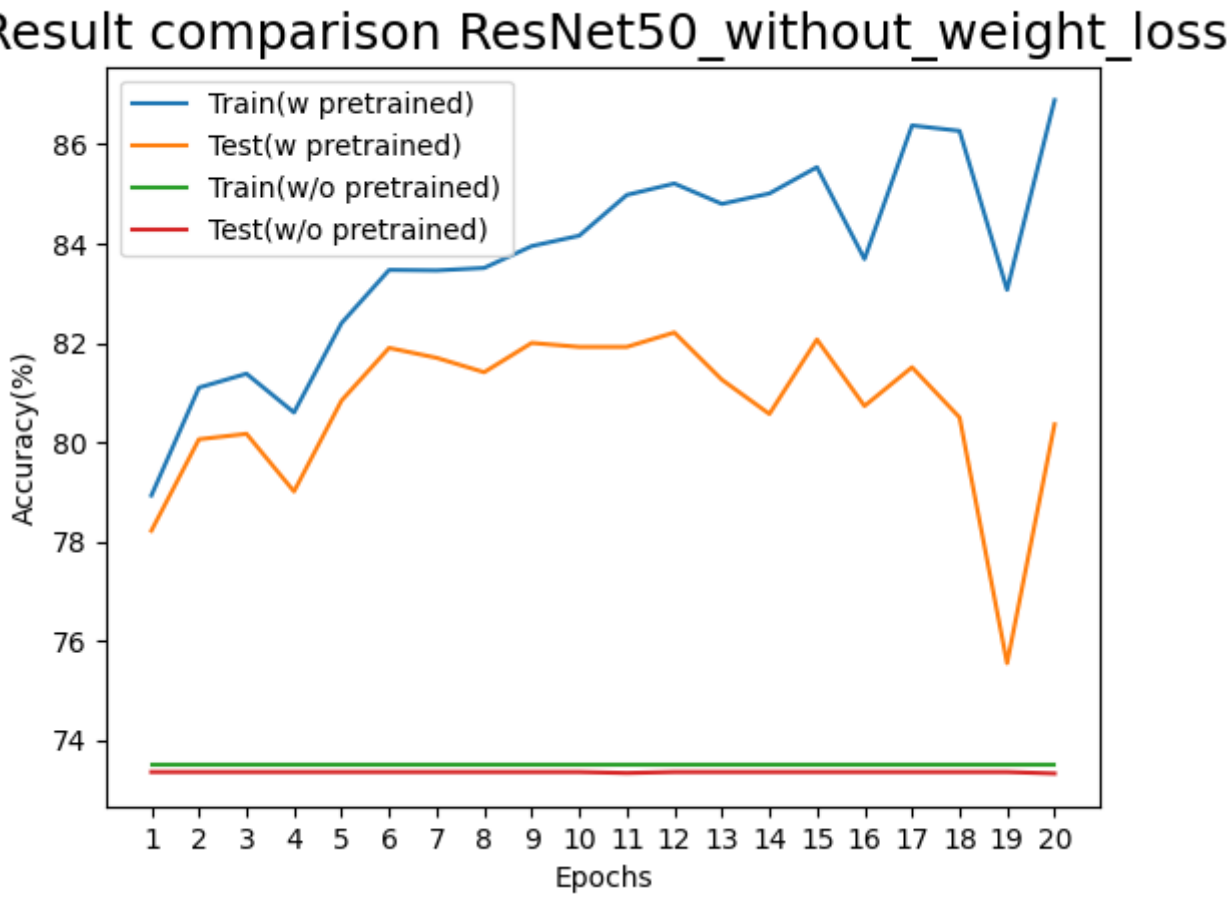

Fig.11 ResNet50 without weight loss

Fig.12 ResNet50 with weight loss

```
#!/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(with_pretrained, without_pretrained, title):

    plt.title("Result comparison {0}".format(title), fontsize = 18)

    w_train, w_test = read(with_pretrained)
    wo_train, wo_test = read(without_pretrained)

    e = [x for x in range(1,21)]
```

```
    plt.plot(e, w_train, label="Train(w pretrained)")
    plt.plot(e, w_test, label="Test(w pretrained)")
    plt.plot(e, wo_train, label="Train(w/o pretrained)")
    plt.plot(e, wo_test, label="Test(w/o pretrained)")

    plt.legend(loc='upper left')

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

    plt.xticks(e)

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

def create_argparse():
    parser = argparse.ArgumentParser(prog="DLP homework 4",
    description='This code will show with pretrained and without pretrained result \
    with matpltlib')

    parser.add_argument("with_pretrained_file", type=str, default="none",
    help="input txt file with pretrained result")
    parser.add_argument("without_pretrained_file", type=str, default="none",
    help="input txt file without pretrained result")
    parser.add_argument("title", type=str, default="none",
    help="save title in the figure")

    return parser

if __name__ == "__main__":

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

    plt_result(args.with_pretrained_file, args.without_pretrained_file, args.title)
```

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_homework4 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_homework4"
VIRTUALENV_FOLDER=$(pwd)/${virtualenv_name}
virtualenv ${virtualenv_name}

source ${VIRTUALENV_FOLDER}/bin/activate
python3 -m pip install -r requirements.txt
deactivate
echo "alias DLP_homework4='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
pandas==1.1.5
gdown==3.12.2
```

B. 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.

```
from .ResNet import ResNet18, ResNet50

def get_model(name, num_class, feature_extract, use_pretrained):

    models = {
        'ResNet18' : ResNet18(num_class, feature_extract, use_pretrained),
        'ResNet50' : ResNet50(num_class, feature_extract, use_pretrained)
    }

    return models[name]
```

C. 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 itertools
import copy

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

from dataset.dataset import RetinopathyDataset
from model import get_model

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

        # data
        trainset = RetinopathyDataset("train")
        testset = RetinopathyDataset("test")

        # file name
        text = "with_pretrained" if args.pretrained else "without_pretrained"
        finetune = "feature_extract" if args.feature_extract else "finetune"
        we = "with_weight_loss" if args.weight_loss else "without_weight_loss"
        self.file_name ="{0}_{1}_{2}_{3}_{4}_{5}_{6}".format(args.network, text,
        finetune, we, args.batch_size, args.epochs, args.learning_rate)

        # 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=4)
        self.testloader = DataLoader(dataset=testset,
        batch_size=args.batch_size,
        shuffle=False, num_workers=4)

        weight_loss = torch.from_numpy(trainset.weight_loss).float()
```

```
        print(weight_loss)

        # model
        self.model = get_model(args.network, 5,
        args.feature_extract, args.pretrained)

        # show model parameter
        print(self.model)

        # optimizer
        self.optimizer = torch.optim.SGD(self.model.parameters(),
        lr=args.learning_rate, momentum=0.9, weight_decay = 5e-4)

        # criterion
        self.criterion = nn.CrossEntropyLoss(weight=weight_loss) \
        if args.weight_loss else 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))
                print("Load model weight.")
            else:
                raise("This path is not exist!")

        self.evaluate(self.testloader) if args.test else self.training()

    # training
    def training(self):

        best_model_weight = None
        best_accuracy = 0.0
        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())

                self.optimizer.zero_grad()
                loss.backward()
```

```
                self.optimizer.step()
                train_loss += loss.item()

                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)

            if(te_ac > best_accuracy):
                best_accuracy = te_ac
                best_model_weight = copy.deepcopy(self.model.state_dict())

        self.plot_confusion_matrix(5)
        # save the best model
        torch.save(best_model_weight, os.path.join(self.weight_path,
        self.file_name + '.pkl'))

        # plot learning curve
        self.plt_lr_cur()

    # evaluation
    def evaluate(self, d):

        correct = 0.0
        total = 0.0
        tbar = tqdm(d)
        self.model.eval()
        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()

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

            text = "Train accuracy" if d == self.trainloader else "Test accuracy"
            tbar.set_description('{0}: {1:2.2f}% '.format(text,
            100. * correct / total))

        return 100.0 * correct / total

    # plot learning curve
    def plt_lr_cur(self):

        plt.figure(2)
        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.png'))
        # 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}, test accurancy : {3:.2f}\n".format(epochs, loss, tr_ac, te_ac))
```

D. 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 4",
        description='This lab implement ResNet-18 and ResNet-50 to analysis \
        diabetic retinopathy')

        # training hyper parameters
        parser.add_argument("--learning_rate", type=float, default=0.001,
        help="Learning rate for the trainer, default is 1e-3")
        parser.add_argument("--epochs", type=int, default=10,
        help="The number of the training, default is 10")
        parser.add_argument("--batch_size", type=int, default=8,
        help="Input batch size for training, default is 8")

        # 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")
        parser.add_argument("--test", action="store_true", default=False,
        help="True is test model, False is keep train, default is False")

        # 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="ResNet18",
        help="select network ResNet18 and ResNet50, default is ResNet18.")
        parser.add_argument("--pretrained", action='store_true',
        default=False, help="load pretrained weight to train, default is False")
        parser.add_argument("--feature_extract", action='store_true',
        default=False, help="If feature_extract = False, the model is finetuned and \
        all model parameters are updated. If feature_extract = True, only the last \
        layer parameters are updated, the others remain fixed. Default is False")
        parser.add_argument("--weight_loss", action="store_true",default=False,
        help="use weight loss to solve imbalance data. default is False.")

        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
```

E. 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 or test.

homework4
├── dataset
│ ├── dataset.py
│ ├── RetinopathyDataset
│ ├── train_img.csv │ ├── test_img.csv
│ ├── train_label.csv
│ └── test_label.csv
├── install.sh
├── main.py
├── model
│ ├── ResNet.py
│ └── *_init_*.py
├── option.py
├── picture
├── record
├── requirements.txt

```
├── trainer.py
├── visual.py
└── weight
```

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

from option import Option
from trainer import Trainer
import time

if __name__ == '__main__':

    # paramters
    args = Option().create()

    time_start = time.time()
    # trainer
    trainer = Trainer(args)

    time_end = time.time()

    print("Use {0} finished.".format(time_end - time_start))
```

tags: `DLP2021`