

南开大学

Python 语言程序设计 实验报告



南开大学 网络空间安全学院 信息安全专业

姓名 刘修铭

学号 2112492

2022 年 12 月 7 日

目 录

第一章 作业题目.....	1
第二章 运行环境.....	2
2.1 开发软件.....	2
2.2 开发环境.....	2
2.3 硬件环境.....	2
第三章 论文模型复现.....	3
3.1 模型总述.....	3
3.2 复现困难点及解决方案.....	4
3.2.1 环境不适配	4
3.2.2 显存分配不足	4
3.3 复现截图.....	5
第四章 基于自定义网络的模型实现.....	8
4.1 模型总述.....	8
4.2 模型实现困难点及解决方案.....	8
4.2.1 数据集处理	8
4.2.2 自定义网络的搭建	8
4.3 模型实现过程.....	9
4.3.1 数据集加载	9
4.3.2 模型搭建.....	11
4.4 模型实现截图.....	14
4.6 评价.....	16
4.7 模型改进方案——迁移学习 Transfer learning	16
4.7.1 改进方案总述	16
4.7.2 改进方案实现过程	17
4.7.3 改进方案实现截图	17
4.7.5 改进方案评价	19
第五章 总结与展望.....	20

参考文献.....	21
-----------	----

第一章 作业题目

参考附件论文，在 DDR 数据集的训练样本上，基于 Pytorch 框架训练一个用于图像分类的卷积神经网络，并在测试样本上完成测试。发现模型存在的问题，并提出可能的解决方案。

第二章 运行环境

2.1 开发软件

Visual Studio Code

2.2 开发环境

python 3.10.6

torch 1.13.0

torchvision 0.14.0

visdom 0.2.3

numpy 1.23.4

keras 2.10.0

tensorflow 2.10.0

2.3 硬件环境

笔记本电脑：联想拯救者 Y9000P

CPU：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz

显卡：NVIDIA GeForce RTX 3060 Laptop GPU

内存：16GB

第三章 论文模型复现

3.1 模型总述

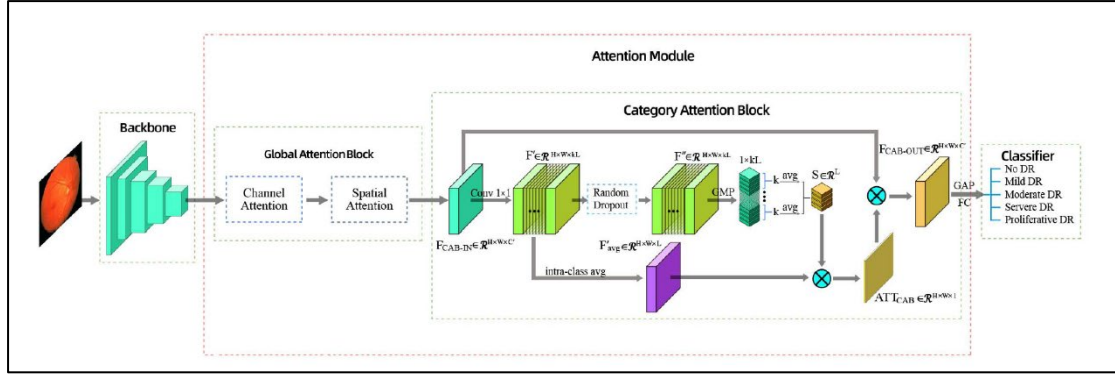


图 1

如图 1, *CABNet* 以一张眼底图像为输入, 用骨干网作为特征提取器, 获取全局特征图。我们可以采用 *ImageNet*[32] 上预训练过的任意 *CNN* 作为骨干网, 提取特征图 $F \in R^{H \times W \times C}$ 从最后一个卷积层, 其中包含眼底图像的高级语义特征, 其中 H 、 W 和 C 分别表示特征图中的高度、宽度和通道数。接下来, 为了减少计算成本和内存使用, 我们在 F 上应用 1×1 卷积层进行信道约简, 得到 $F_{reduce} \in R^{H \times W \times C'}$, 其中 $C' = C/2$, 这是 *GAB* 的输入。然后, *GAB* 学习通道型和空间型注意特征图 $F_{GAB-OUT}$ 以保存更详细的小病变信息, 抑制较少有用的信息。接下来, $F_{GAB-OUT}$ 作为 *CAB* 的输入, 迫使网络对每个 DR 等级学习不同的区分区域特征, 并产生输出 $F_{CAB-OUT}$ 。 *CAB* 与 *GAB* 兼容, 它们连接在 *CABNet* 的注意模块中。最后, 我们使用一个全局平均池化 (*GAP*) 层和一个完全连接 (*FC*) 层来执行分类任务, 预测每个输入图像的类标签。

CABNet 基于一个经过大规模图像数据集预训练的骨干网。我们应用随机水平翻转、垂直翻转和随机旋转作为数据增强的形式, 以减少过拟合, 我们网络的输入分辨率为 512×512 。初始学习率设置为 0.005, 如果验证数据集上的性能不能在 3 个周期内得到改善, 则衰减 0.8 倍。所有模型都用 *Adam* 优化器和交叉熵损失函数训练了 70 个周期。并且对于每个骨干, 在验证集上选择表现最好 (损失最小) 的模型作为最终模型。

3.2 复现困难点及解决方案

3.2.1 环境不适配

3.2.1.1 困难点描述

原文中的模型是基于 python 3.6、keras 2.3.1 与 tensorflow 1.13.1 等环境下运行的,与现有运行环境存在一定差异,导致部分工具包出现更新而无法直接使用。

3.2.1.2 解决方案

借助百度、CSDN、help 指令等工具寻找替代工具包,代替源代码中已失效工具包。

3.2.2 显存分配不足

3.2.2.1 困难点描述

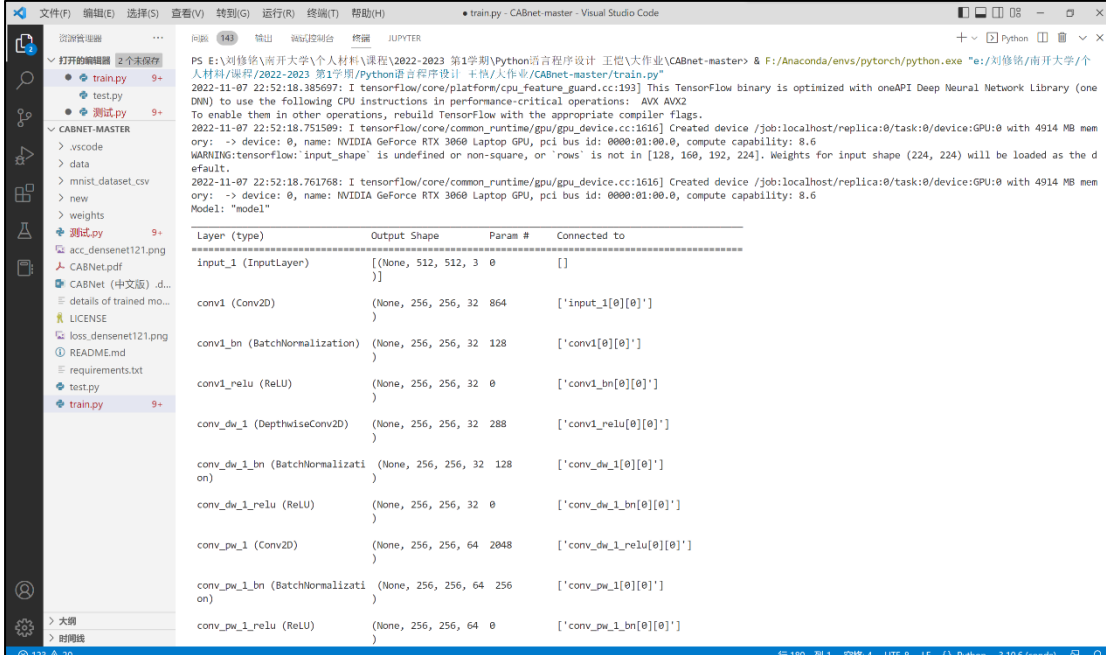
在运行代码时,系统提示“RESOURCE_EXHAUSTED:failed to allocate memory”。经查阅 CSDN 得知,此为显存分配不足报错。

3.2.2.2 解决方案

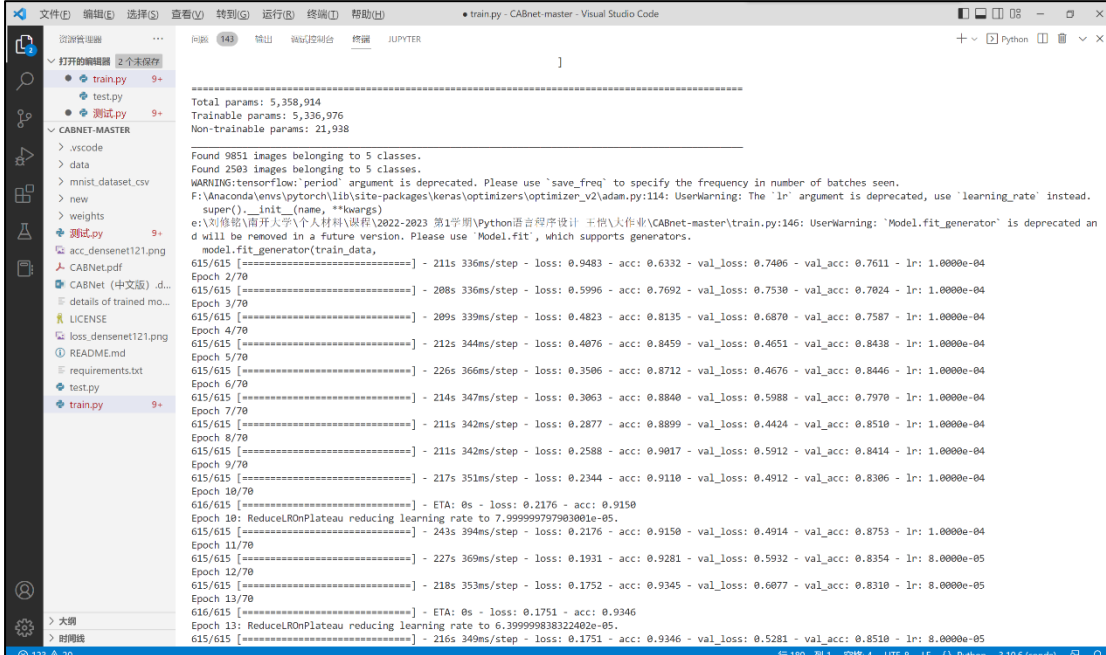
添加相关代码,调整为动态分配 GPU 内存,并设置最大利用率为 80%,避免电脑损坏。

```
import tensorflow.compat.v1 as tf
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
config=tf.compat.v1.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction=0.8
config.gpu_options.allow_growth = True # 设置动态分配 GPU 内存
sess=tf.compat.v1.Session(config=config)
```

3.3 复现截图



Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 512, 512, 3)]	0	['input_1[0][0]']
conv1 (Conv2D)	(None, 256, 256, 32)	864	['conv1[0][0]']
conv1_bn (BatchNormalization)	(None, 256, 256, 32)	128	['conv1[0][0]']
conv1_relu (ReLU)	(None, 256, 256, 32)	0	['conv1_bn[0][0]']
conv_dw_1 (DepthwiseConv2D)	(None, 256, 256, 32)	288	['conv1_relu[0][0]']
conv_dw_1_bn (BatchNormalization)	(None, 256, 256, 32)	128	['conv_dw_1[0][0]']
conv_dw_1_relu (ReLU)	(None, 256, 256, 32)	0	['conv_dw_1_bn[0][0]']
conv_pw_1 (Conv2D)	(None, 256, 256, 64)	2048	['conv_dw_1_relu[0][0]']
conv_pw_1_bn (BatchNormalization)	(None, 256, 256, 64)	256	['conv_pw_1[0][0]']
conv_pw_1_relu (ReLU)	(None, 256, 256, 64)	0	['conv_pw_1_bn[0][0]']



Total params: 5,358,914
 Trainable params: 5,336,976
 Non-trainable params: 21,938

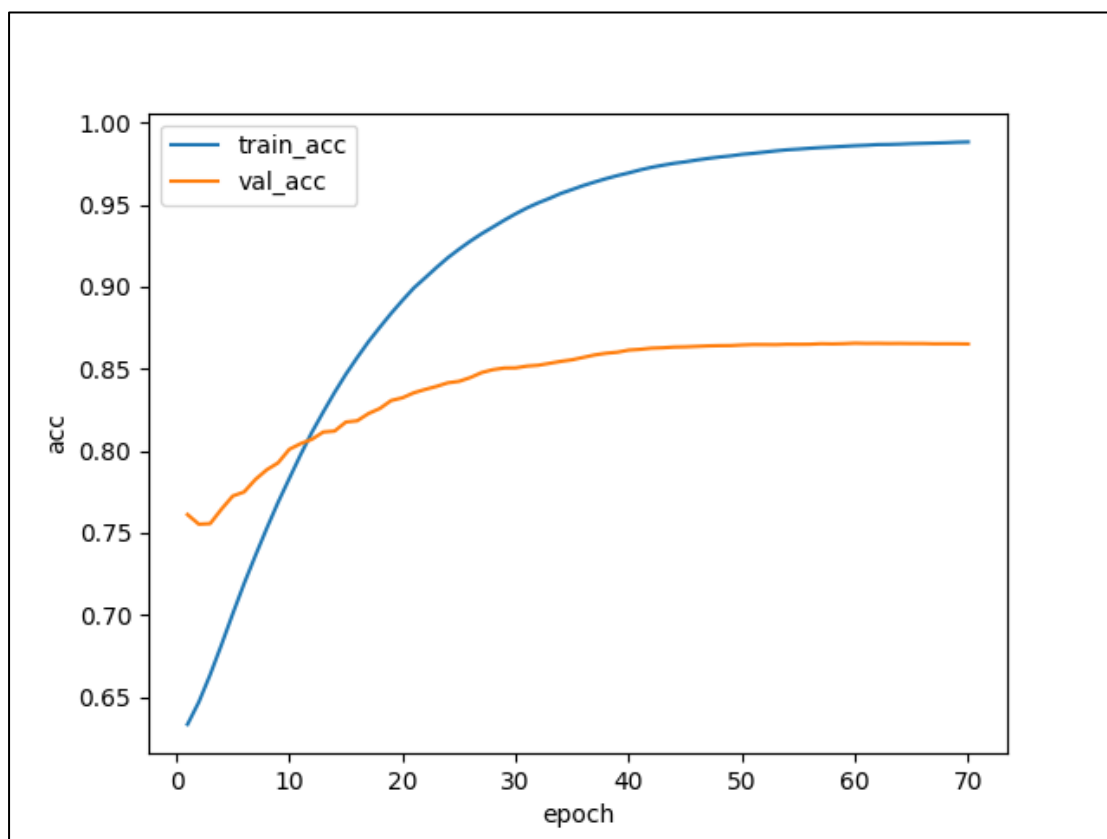
Found 9851 images belonging to 5 classes.
 Found 2503 images belonging to 5 classes.
 WARNING:tensorflow: 'period' argument is deprecated. Please use 'save_freq' to specify the frequency in number of batches seen.
 F:\Anaconda\envs\pytorch\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The 'lr' argument is deprecated, use 'learning_rate' instead.
 e:\刘修铭\南开大学\个人材料\课程\2022-2023 第1学期\Python语言程序设计 主修\大作业\CABNet-master\train.py:146: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit', which supports generators.

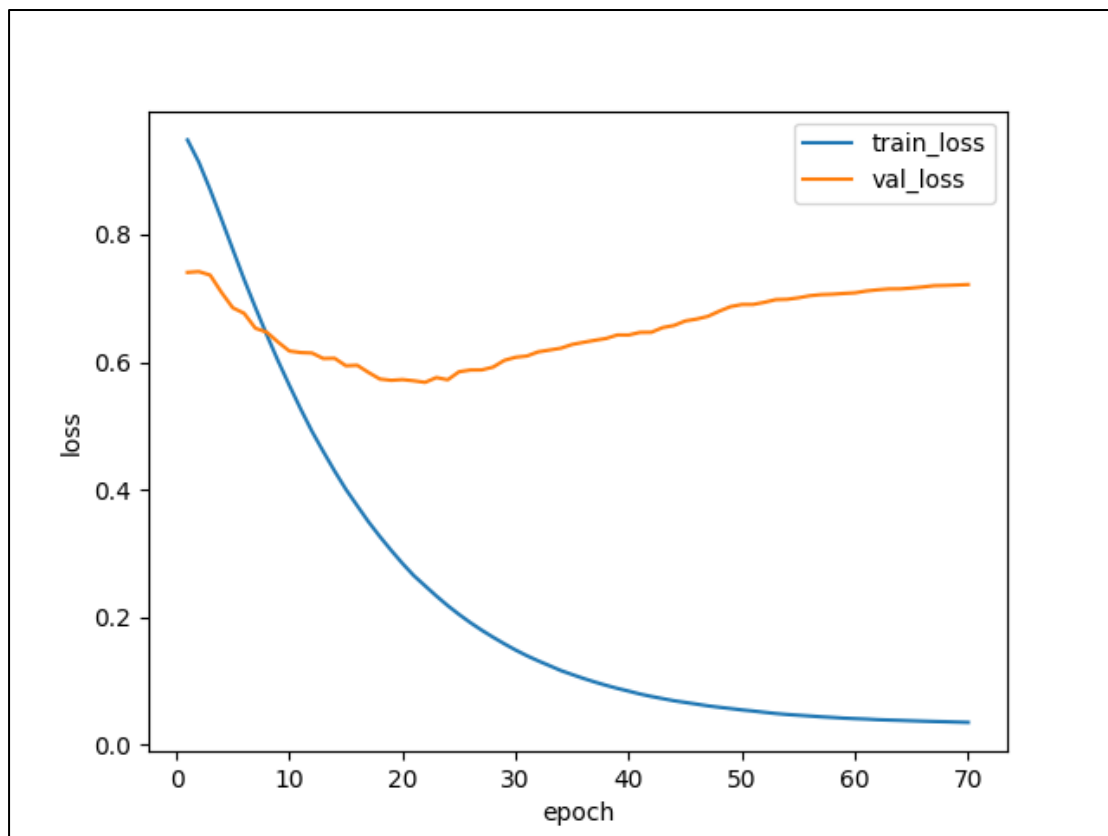
Epoch	Time	Loss	Acc	Val Loss	Val Acc	LR
Epoch 2/70	615/615	0.9483	0.6332	0.7406	0.7611	1.0000e-04
Epoch 3/70	615/615	0.5996	0.7692	0.7530	0.7824	1.0000e-04
Epoch 4/70	615/615	0.4823	0.8135	0.6870	0.7587	1.0000e-04
Epoch 5/70	615/615	0.4076	0.8459	0.4651	0.8438	1.0000e-04
Epoch 6/70	615/615	0.3506	0.8712	0.4676	0.8446	1.0000e-04
Epoch 7/70	615/615	0.3063	0.8840	0.5988	0.7970	1.0000e-04
Epoch 8/70	615/615	0.2877	0.8899	0.4424	0.8510	1.0000e-04
Epoch 9/70	615/615	0.2588	0.9017	0.5912	0.8414	1.0000e-04
Epoch 10/70	615/615	0.2344	0.9110	0.4912	0.8306	1.0000e-04
Epoch 11/70	616/615	ETA: 0s	0.2176	acc: 0.9150		
Epoch 12/70	615/615	0.2176	0.9150	0.4914	0.8753	1.0000e-04
Epoch 13/70	615/615	0.1931	0.9281	0.5932	0.8354	8.0000e-05
Epoch 14/70	615/615	0.1752	0.9345	0.6077	0.8310	8.0000e-05
Epoch 15/70	616/615	ETA: 0s	0.1751	acc: 0.9346		
Epoch 16/70	615/615	0.1751	0.9346	0.5281	0.8510	8.0000e-05



```
train.py - CABNet-master - Visual Studio Code

615/615 [=====] - 188s 304ms/step - loss: 0.0385 - acc: 0.9861 - val_loss: 0.7512 - val_acc: 0.8646 - lr: 5.4976e-06
Epoch 50/70
615/615 [=====] - 187s 303ms/step - loss: 0.0354 - acc: 0.9889 - val_loss: 0.7204 - val_acc: 0.8678 - lr: 4.3980e-06
Epoch 51/70
615/615 [=====] - 187s 303ms/step - loss: 0.0366 - acc: 0.9867 - val_loss: 0.6909 - val_acc: 0.8670 - lr: 4.3980e-06
Epoch 52/70
616/615 [=====] - ETA: 0s - loss: 0.0328 - acc: 0.9889
Epoch 52: ReduceLRonPlateau reducing learning rate to 3.51843706277964e-06.
615/615 [=====] - 187s 303ms/step - loss: 0.0328 - acc: 0.9889 - val_loss: 0.7240 - val_acc: 0.8646 - lr: 4.3980e-06
Epoch 53/70
615/615 [=====] - 186s 301ms/step - loss: 0.0321 - acc: 0.9895 - val_loss: 0.7357 - val_acc: 0.8642 - lr: 3.5184e-06
Epoch 54/70
615/615 [=====] - 185s 300ms/step - loss: 0.0336 - acc: 0.9892 - val_loss: 0.7028 - val_acc: 0.8670 - lr: 3.5184e-06
Epoch 55/70
Epoch 61: ReduceLRonPlateau reducing learning rate to 1.801439793780446e-06.
615/615 [=====] - 188s 305ms/step - loss: 0.0341 - acc: 0.9887 - val_loss: 0.7376 - val_acc: 0.8642 - lr: 2.2518e-06
Epoch 62/70
615/615 [=====] - 186s 302ms/step - loss: 0.0305 - acc: 0.9902 - val_loss: 0.7299 - val_acc: 0.8658 - lr: 1.8014e-06
Epoch 63/70
615/615 [=====] - 187s 303ms/step - loss: 0.0332 - acc: 0.9873 - val_loss: 0.7268 - val_acc: 0.8646 - lr: 1.8014e-06
Epoch 64/70
616/615 [=====] - ETA: 0s - loss: 0.0333 - acc: 0.9886
Epoch 64: ReduceLRonPlateau reducing learning rate to 1.441151835024357e-06.
615/615 [=====] - 186s 301ms/step - loss: 0.0333 - acc: 0.9886 - val_loss: 0.7156 - val_acc: 0.8658 - lr: 1.8014e-06
Epoch 65/70
615/615 [=====] - 188s 305ms/step - loss: 0.0319 - acc: 0.9898 - val_loss: 0.7280 - val_acc: 0.8646 - lr: 1.4412e-06
Epoch 66/70
615/615 [=====] - 186s 302ms/step - loss: 0.0317 - acc: 0.9889 - val_loss: 0.7331 - val_acc: 0.8654 - lr: 1.4412e-06
Epoch 67/70
616/615 [=====] - ETA: 0s - loss: 0.0318 - acc: 0.9891
Epoch 67: ReduceLRonPlateau reducing learning rate to 1.1529215043992736e-06.
615/615 [=====] - 189s 305ms/step - loss: 0.0318 - acc: 0.9891 - val_loss: 0.7376 - val_acc: 0.8638 - lr: 1.4412e-06
Epoch 68/70
615/615 [=====] - 187s 302ms/step - loss: 0.0321 - acc: 0.9894 - val_loss: 0.7238 - val_acc: 0.8654 - lr: 1.1529e-06
Epoch 69/70
615/615 [=====] - 188s 304ms/step - loss: 0.0309 - acc: 0.9907 - val_loss: 0.7264 - val_acc: 0.8650 - lr: 1.1529e-06
Epoch 70/70
616/615 [=====] - ETA: 0s - loss: 0.0309 - acc: 0.9898
Epoch 70: ReduceLRonPlateau reducing learning rate to 9.22372217093129e-07.
615/615 [=====] - 186s 302ms/step - loss: 0.0309 - acc: 0.9898 - val_loss: 0.7281 - val_acc: 0.8642 - lr: 1.1529e-06
PS E:\刘修路\南开大学\个人材料\课程\2022-2023 第1学期\Python语言程序设计_王旭\大作业\CABNet-master>
```





第四章 基于自定义网络的模型实现

4.1 模型总述

该模型为在自己搭建的 18 层神经网络的框架下对给定的眼底图片数据集进行训练，并利用现有的工具包中 ResNet18 网络进行优化。

4.2 模型实现困难点及解决方案

4.2.1 数据集处理

4.2.2.1 困难点描述

对于 Pytorch 框架而言，课堂上讲授其使用方法主要针对库中现有数据集展开。而本次作业要求使用给定数据集，即在给定数据集的基础上进行训练。且数据集的处理又直接关乎模型训练的准确率。故如何更好地完成对数据集的处理成了整个任务最基础也是最核心的部分。

4.2.2.2 解决方案

通过查阅资料，学习成熟的模型结构，在借助现有工具包的基础上，自己编写相关函数，实现对于自定义数据集的加载跟预处理。

4.2.2 自定义网络的搭建

4.2.2.1 困难点描述

对于卷积神经网络比较陌生，不知该如何从零编写网络。

4.2.2.2 解决方案

通过查阅资料，学习成熟的模型结构，在借助课堂中有关网络讲解的基础上，

自己编写相关函数，实现自定义网络的编写。

4.3 模型实现过程

4.3.1 数据集加载

4.3.1.1 构造函数

修改传入图片尺寸使其适应所设定的网络。同时将每张图片的 label 确定。

```
def __init__(self, root, resize, mode):
    super(DDR, self).__init__()

    self.root = root
    self.resize = resize

    self.name2label = {}
    for name in sorted(os.listdir(os.path.join(root))):
        if not os.path.isdir(os.path.join(root, name)):
            continue

        self.name2label[name] = len(self.name2label.keys())

    # image, label
    self.images, self.labels = self.load_csv('images.csv')

    if mode=='train': # 60%
        self.images = self.images[:int(0.6*len(self.images))]
        self.labels = self.labels[:int(0.6*len(self.labels))]
    elif mode=='val': # 20% = 60%->80%
        self.images = self.images[int(0.6*len(self.images)):
int(0.8*len(self.images))]
        self.labels = self.labels[int(0.6*len(self.labels)):
int(0.8*len(self.labels))]
    else: # 20% = 80%->100%
        self.images = self.images[int(0.8*len(self.images)):]
        self.labels = self.labels[int(0.8*len(self.labels)):]
```

4.3.1.2 文件格式转化与读取

将数据集加载成 CSV 文件，防止读取数据时内存不足。

```
def load_csv(self, filename):
```

```
if not os.path.exists(os.path.join(self.root, filename)):
    images = []
    for name in self.name2label.keys():
        # 'DDR\\0\\00001.png'
        images += glob.glob(os.path.join(self.root, name,
        '*.png'))
        images += glob.glob(os.path.join(self.root, name,
        '*.jpg'))
        images += glob.glob(os.path.join(self.root, name,
        '*.jpeg'))

    # 1167, 'DDR\\0\\00000000.png'
    print(len(images), images)

    random.shuffle(images)
    with open(os.path.join(self.root, filename), mode='w',
    newline='') as f:
        writer = csv.writer(f)
        for img in images: # 'DDR\\0\\00000000.png'
            name = img.split(os.sep)[-2]
            label = self.name2label[name]
            # 'DDR\\0\\00000000.png', 0
            writer.writerow([img, label])
        print('written into csv file:', filename)

    # read from csv file
    images, labels = [], []
    with open(os.path.join(self.root, filename)) as f:
        reader = csv.reader(f)
        for row in reader:
            # 'DDR\\0\\00000000.png', 0
            img, label = row
            label = int(label)

            images.append(img)
            labels.append(label)

    assert len(images) == len(labels)

    return images, labels
```

4.3.1.3 自定义数据集的载入

通过调用工具包中的 `transforms` 函数将数据集进行一定的处理，包括随机剪裁、随机旋转、调整尺寸、格式转换等，使其更加适用于模型训练。

```
def __getitem__(self, idx):
    # idx~[0~len(images)]
    # self.images, self.labels
    # img: 'DDR\\0\\00000000.png'
    # label: 0
    img, label = self.images[idx], self.labels[idx]

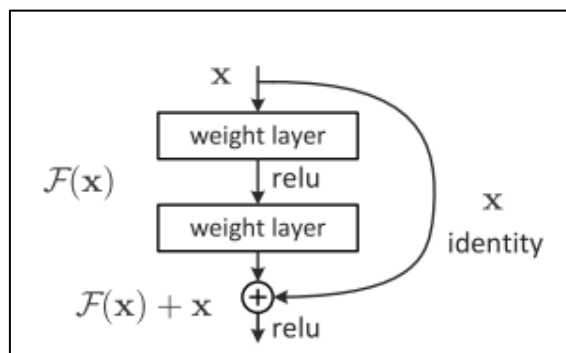
    tf = transforms.Compose([
        lambda x:Image.open(x).convert('RGB'), # string path= >
image data
        transforms.Resize((int(self.resize*1.25),
int(self.resize*1.25))),
        transforms.RandomRotation(15),
        transforms.CenterCrop(self.resize),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    img = tf(img)
    label = torch.tensor(label)

    return img, label
```

4.3.2 模型搭建

4.3.2.1 基本残差块



它对每层的输入做一个*reference* (X)，学习形成残差函数，而不是学习一些没有*reference* (X) 的函数。这种残差函数更容易优化，能使网络层数大大加深。

```
class ResBlk(nn.Module):
    """
    resnet block
```

```
def __init__(self, ch_in, ch_out, stride=1):
    """
    :param ch_in:
    :param ch_out:
    """
    super(ResBlk, self).__init__()

    self.conv1 = nn.Conv2d(ch_in, ch_out, kernel_size=3,
stride=stride, padding=1)
    self.bn1 = nn.BatchNorm2d(ch_out)
    self.conv2 = nn.Conv2d(ch_out, ch_out, kernel_size=3, stride=1,
padding=1)
    self.bn2 = nn.BatchNorm2d(ch_out)

    self.extra = nn.Sequential()
    if ch_out != ch_in:
        # [b, ch_in, h, w] => [b, ch_out, h, w]
        self.extra = nn.Sequential(
            nn.Conv2d(ch_in, ch_out, kernel_size=1, stride=stride),
            nn.BatchNorm2d(ch_out)
        )

def forward(self, x):
    """
    :param x: [b, ch, h, w]
    :return:
    """
    out = F.relu(self.bn1(self.conv1(x)))
    out = self.bn2(self.conv2(out))
    # short cut.
    # extra module: [b, ch_in, h, w] => [b, ch_out, h, w]
    # element-wise add:
    out = self.extra(x) + out
    out = F.relu(out)

    return out
```

4.3.2.2 网络的构建

18 层的网络有五个部分组成, 从 conv2 开始, 每层都有两个有残差块, 并且每个残差块具有 2 个卷积层。最后有一个全连接层。

```
class ResNet18(nn.Module):
```

```
    def __init__(self, num_class):
        super(ResNet18, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3,
                stride=3, padding=0),
            nn.BatchNorm2d(16)
        )
        # followed 4 blocks
        # [b, 16, h, w] => [b, 32, h, w]
        self.blk1 = ResBlk(16, 32, stride=3)
        # [b, 32, h, w] => [b, 64, h, w]
        self.blk2 = ResBlk(32, 64, stride=3)
        # [b, 64, h, w] => [b, 128, h, w]
        self.blk3 = ResBlk(64, 128, stride=2)
        # [b, 128, h, w] => [b, 256, h, w]
        self.blk4 = ResBlk(128, 256, stride=2)

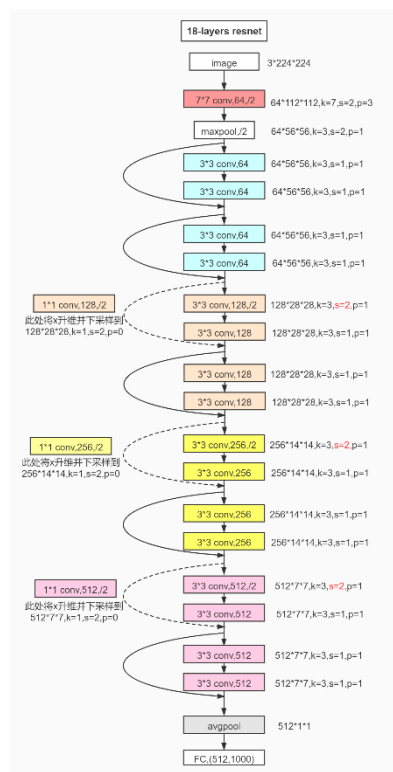
        # [b, 256, 7, 7]
        self.outlayer = nn.Linear(256*3*3, num_class)

    def forward(self, x):
        """
        :param x:
        :return:
        """
        x = F.relu(self.conv1(x))

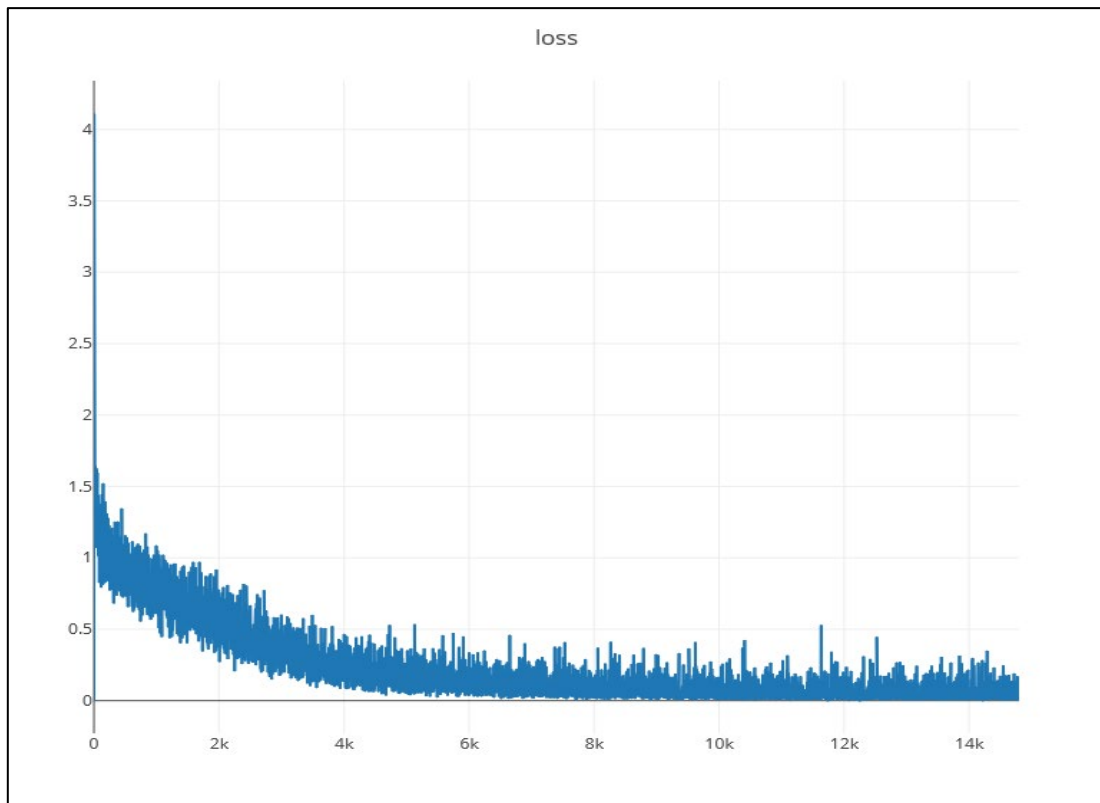
        # [b, 64, h, w] => [b, 1024, h, w]
        x = self.blk1(x)
        x = self.blk2(x)
        x = self.blk3(x)
        x = self.blk4(x)

        # print(x.shape)
        x = x.view(x.size(0), -1)
        x = self.outlayer(x)

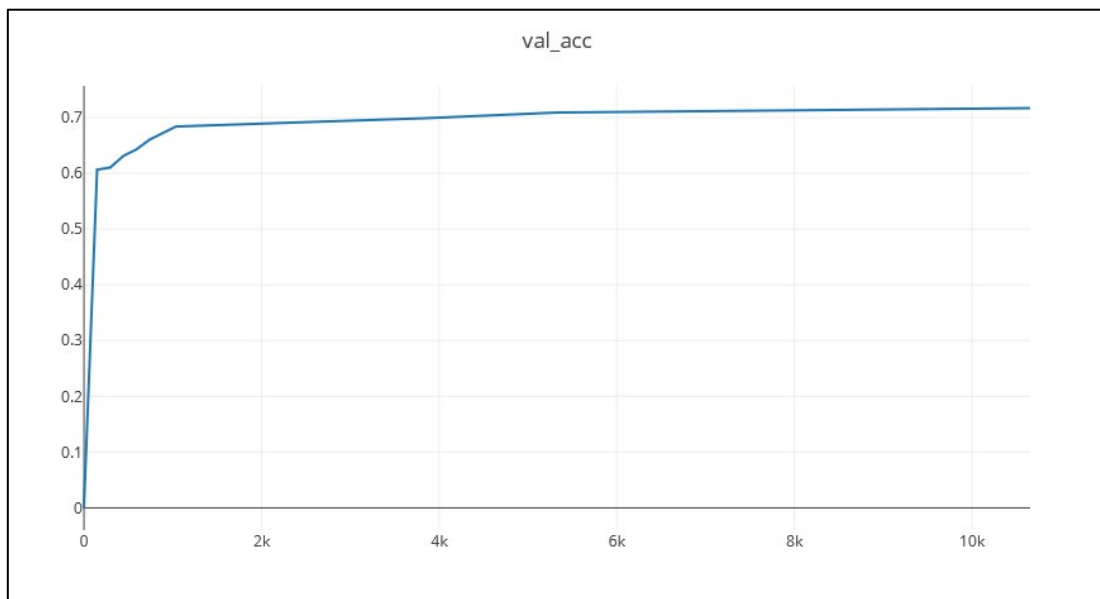
        return x
```



4.4.3 loss 损失函数



4.4.4 val_acc



4.6 评价

该网络训练得到的模型性能较低，准确率最高仅为 69%。

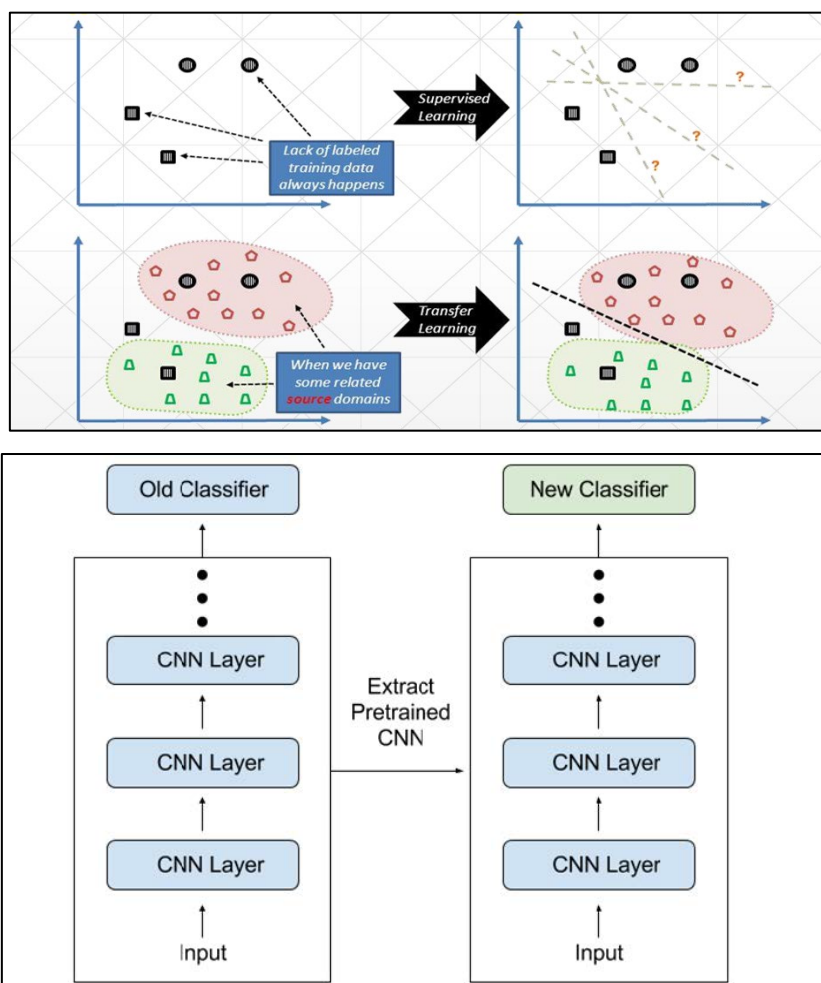
部分数据集数据量少，容易出现神经网络过拟合 **overfitting**，即表现为模型在训练上表现越来越好，但在未见过的数据上表现较差，缺少泛化能力。

因此考虑引入迁移学习来优化模型。

4.7 模型改进方案——迁移学习 Transfer learning

4.7.1 改进方案总述

迁移学习(Transfer Learning)是一种机器学习方法，就是把为任务 A 开发的模型作为初始点，重新使用在为任务 B 开发模型的过程中。此处，利用工具包中已开发的 ResNet18 模型进行训练以提高模型准确率。



4.7.2 改进方案实现过程

4.7.2.1 数据集加载

同 4.3.1 数据集加载。

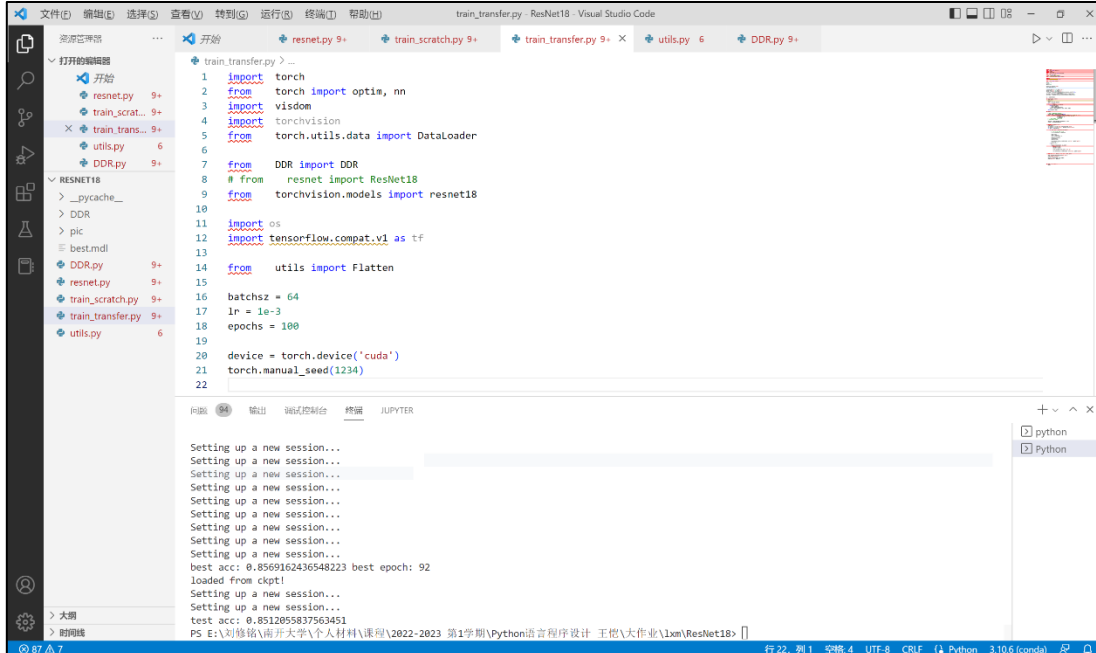
4.7.2.2 模型搭建

同 4.3.2 模型搭建。

4.7.2.3 模型的训练与测试

4.7.3 改进方案实现截图

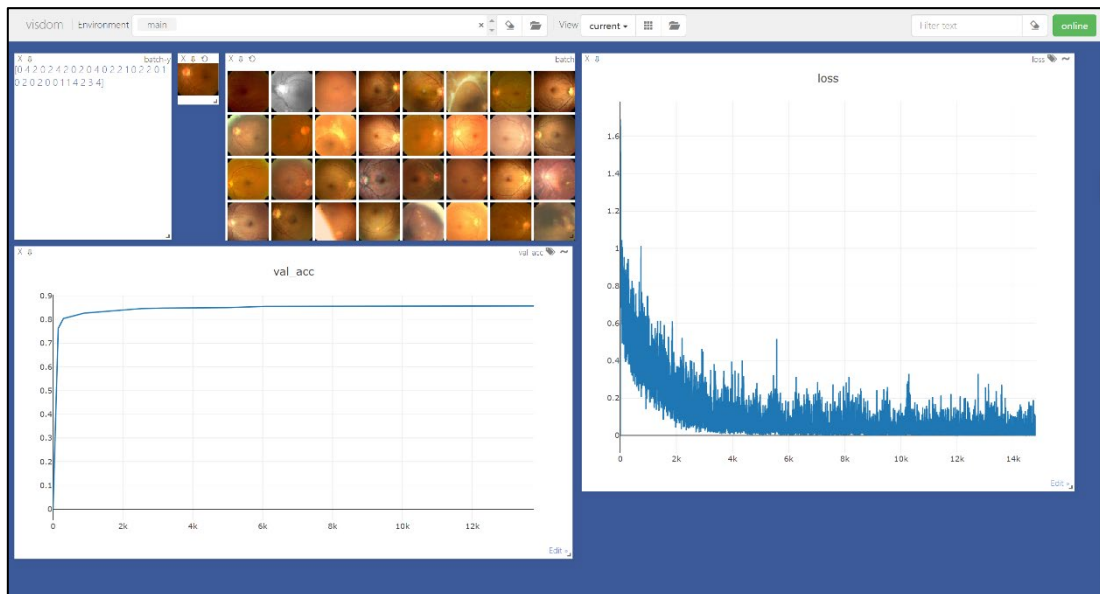
4.7.3.1 Visual Studio Code 运行截图



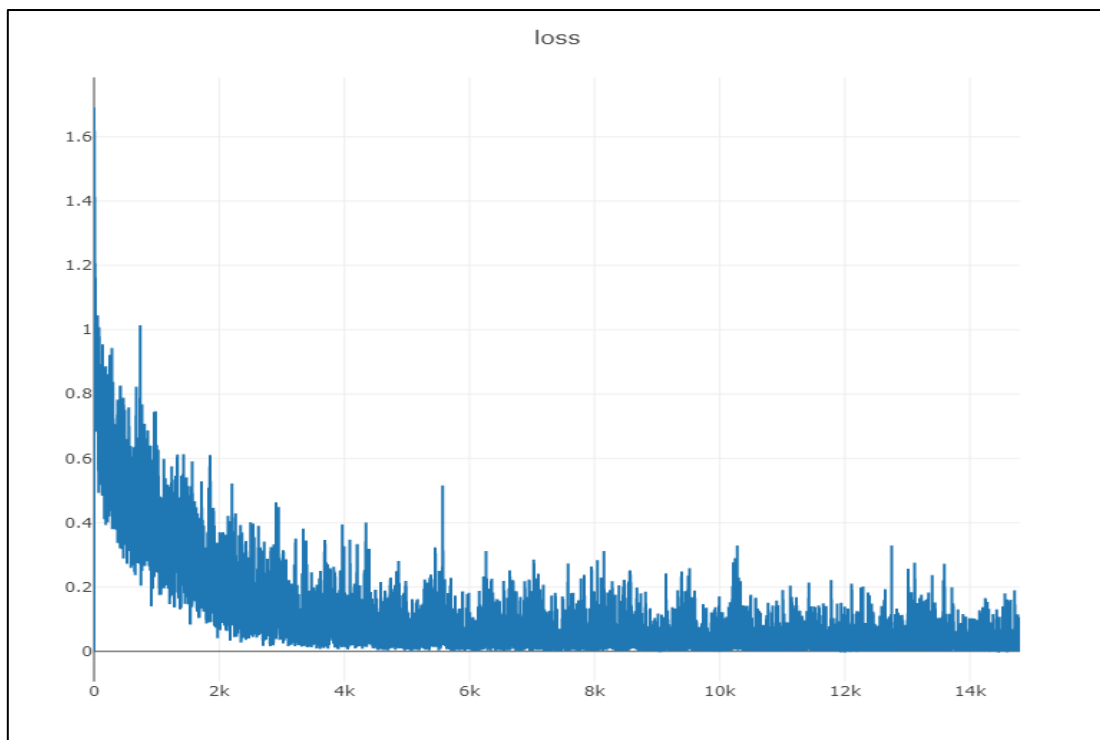
```
1 import torch
2 from torch import optim, nn
3 import visdom
4 import torchvision
5 from torch.utils.data import DataLoader
6
7 from DDR import DDR
8 # from resnet import ResNet18
9 from torchvision.models import resnet18
10
11 import os
12 import tensorflow.compat.v1 as tf
13
14 from utils import Flatten
15
16 batchsz = 64
17 lr = 1e-3
18 epochs = 100
19
20 device = torch.device('cuda')
21 torch.manual_seed(1234)
22
```

Setting up a new session...
Setting up a new session...
Setting up a new session...
Setting up a new session...
Setting up a new session...
Setting up a new session...
Setting up a new session...
Setting up a new session...
best acc: 0.8569162436548223 best epoch: 92
loaded from ckpt!
Setting up a new session...
Setting up a new session...
test acc: 0.8512055837563451
PS E:\刘修敏\南开大学\个人材料\课程\2022-2023 第1学期\Python语言程序设计 王旭\大作业\1xm\ResNet18>

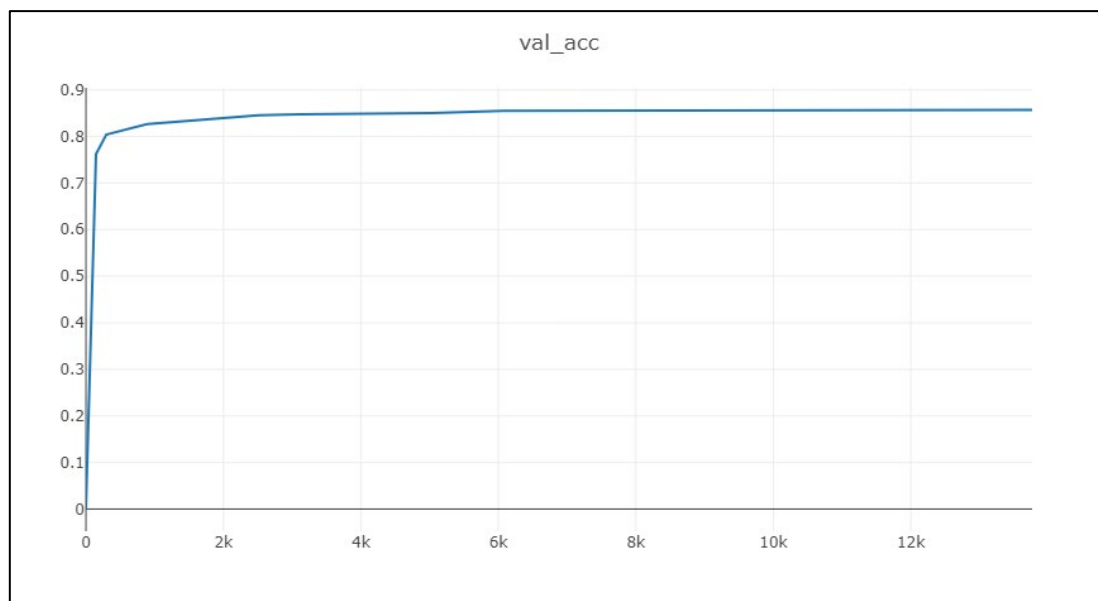
4.7.3.2 Visdom 可视化监视截图



4.7.3.3 loss 损失函数



4.7.3.4 val_acc



4.7.5 改进方案评价

模型经迁移学习优化后，准确率最高可达 85.12%，较优化前有了较大提升。因此，在满足迁移学习的条件的基础上，可以考虑使用迁移学习的方法提升模型的性能。

第五章 总结与展望

Python 语言程序设计大作业至此也就告一段落了。心中不免有些碎碎念。

上学期疫情导致推迟的期末考试，让这学期的本不富裕的教学周雪上加霜。屈指可数的教学周让我一度对自己能否坚持到底产生怀疑。

还记得最初拿到大作业题目，看着不同选题难度带来的巨大分差，自己还是硬着头皮选了当时连题目都没弄懂的难度最大的那一个。选定题目后刚开始的那段时间，每天都在琢磨这个大作业是要让我干啥。每天腆着脸去问身边的朋友，这个作业是在考察我们什么，该从哪里入手，不断地怀疑这个作业以自己的能力能否完成。直到在朋友圈看到有同学说“终于训练出一个相对稳定的模型了”，那一瞬间给我带来的冲击无疑是巨大的。后面我便琢磨从老师给出的代码入手，先跑跑试试看。逐渐的，我成了身边人中第一个跑出来的。信心也就由此逐步建立。

囿于时间与个人编程能力的限制，本次作业仍抱有一些遗憾：一是对模型参数的调整不成熟，导致模型的准确率不是特别高，甚至没能达到 90%；二是没能跑成 ResNet50、ResNet101 等更高级的模型。

感谢 Python 语言程序设计，让我认识到了 Python 语言的巨大潜力。在当今这个大数据时代，Python 语言必不可少，它将在相关领域中继续发挥更大作用。

在此，感谢王恺老师、王胤哲学长以及曹续生学长对我大作业的帮助，感谢赵廷枫同学、邓薇薇同学、曹昕城同学、汤志文同学以及周俊成同学在 Python 课堂学习上对我的帮助！

谢谢！

参考文献

1.通过 Pytorch 实现 ResNet18

<https://zhuanlan.zhihu.com/p/157134695>

2.迁移学习 (Transfer)

https://blog.csdn.net/sikh_0529/article/details/126864397

3.Python 编程基础_南开大学_中国大学 MOOC(慕课)

<https://www.icourse163.org/course/NKU-1205696807>

4.深度学习在图像处理中的应用教程

<https://github.com/WZMIAOMIAO/deep-learning-for-image-processing>

5. GitHub - NKUhealong/CABnet

<https://github.com/NKUhealong/CABnet>

6.你必须要知道 CNN 模型: ResNet

<https://zhuanlan.zhihu.com/p/31852747>

7.机器学习、深度学习、统计学习、NLP、CV 各领域的联系

https://blog.csdn.net/cloudless_sky/article/details/121358592