

# Python大作业展示

2113839 林帆

# 问题分析

- 机器学习
- 深度学习
- 非结构性数据
- 有监督
- 单一标签
- 分类任务

# 主要工作

- 1、数据处理
- 2、训练函数的实现
- 3、准确率测试
- 4、神经网络的选择
- 5、参数调整
- 6、优化器和损失函数的选择
- 7、问题解决和模型优化

# 一、数据处理

## • 新数据集创建

Data (D:) > test\_code > DDR

| 名称          |
|-------------|
| ^           |
| _test       |
| _train      |
| _valid      |
| final_test  |
| final_train |
| final_valid |
| test        |
| test_image  |
| test_image2 |
| train       |
| train1      |
| valid       |

| 数据集         | 数据量                      | 作用  |
|-------------|--------------------------|---|
| train       | 2992+1638+2238+613+2370  | 标准训练集   |
| test        | 1880+189+1344+71+275     | 标准测试集   |
| valid       | 1253+126+895+47+182      | 训练过程中的测试集   |
| _train      | 361+361+361+361+361      | 对应上述三个标准集中的部分数据，<br>同等级少量相等取样，规避数据不平衡的影响，同时方便测试训练函数 |
| _test       | 16+16+16+16+16           |   |
| _valid      | 18+18+18+18+18           |   |
| train1      | 2992+2880+2958+3065+2946 | 文件夹中图片扩增实现过采样后的样本                                   |
| test_image  | 1                        | 图片增强的测试样例   |
| test_image2 | 0                        |   |
| final_train | 2992+1638+2238+613+2370  | 增强后的训练集   |
| final_test  | 1880+189+1344+71+275     | 增强后的测试集   |
| final_valid | 1253+126+895+47+182      | 增强后的valid集  |

# 按部就班

- 读取(贴标签)
- Dataset

```
# ImageFolder将数据按文件夹名字分类贴上标签
trainset = ImageFolder(root='D:/test_code/DDR/train', transform=trans_form)
testset = ImageFolder(root='D:/test_code/DDR/test', transform=trans_form)
validset = ImageFolder(root='D:/test_code/DDR/valid', transform=trans_form)
```

- 加载
- Dataloader

```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=20, shuffle=True, num_workers=4) # 打乱
testloader = torch.utils.data.DataLoader(testset, batch_size=12, shuffle=True, num_workers=4)
validloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=True, num_workers=4)
```

- 显示类别

```
149     print("set is ok")
150     print(trainset.class_to_idx)
151     print(testset.class_to_idx)
152     print(validset.class_to_idx)
153
154     print(len(trainset))
155     print(len(testset))
156     print(len(validset))
```

```
set is ok
{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}
{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}
{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}
9851
3759
2503
```

- 图像预处理

```
freeze_support()
# 图像预处理
# compose整合
trans_form = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.RandomCrop(32, padding=2), # 随机位置进行裁剪
    # 32*32 (相当于没有随机)
    transforms.RandomHorizontalFlip(), # 以0.5的概率水平翻转给定的PIL图像
    transforms.ToTensor(), # 先由HWC转置为CHW格式; 再转为float类型; 最后, 每个像素除以255
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)), # 归一化
])
```

## 二、训练函数的实现

```
55 # 训练函数
56 def trainfunc(dataloader, validloader, net, lossfunc, optimizer):
```

- 优点:
- 1.改参数快
- 2.多次训练时调用简洁
- 3.加深理解

```
57 # 分batch批次训练，最终全部数据都经过一次训练
58 # 每一批训练后，对参数在原有基础上进行一次更新，对新误差进行计算
59 for batch, (data, target) in enumerate(dataloader):
60     # 现有参数下迭代一次网络
61     output = net(data)
62     # forward: 将数据传入模型，前向传播求出预测的值
63     # 求误差，优化，更新参数
64     loss = 0 # 误差初始化为0
65     optimizer.zero_grad() # 梯度初始化为零，把loss关于weight的导数变成0
66     loss = lossfunc(output, target) # 用给定损失函数lossfunc求loss
67     loss.backward() # backward: 反向传播求梯度
68     optimizer.step() # optimizer: 更新所有参数
69     # 网络中参数将损失函数和优化器连接
```

### 三、准确率的测试

```
175     # 测试集
176     total = len(testset)
177     correct = 0
178     for i, (images, lables) in enumerate(testloader):
179         outputs = net(images)
180         _, predicted = torch.max(outputs.data, 1)
181         correct += (predicted == lables).sum().item()
182     print("accuracy:", correct / total)
183
```



# 四、神经网络的选择

```
class Net(nn.Module):  
  
    def __init__(self):  
        super(Net, self).__init__()  
        self.layer1 = nn.Sequential(  
            nn.Conv2d(3, 64, kernel_size=3, padding=1),  
            nn.BatchNorm2d(64),  
            nn.ReLU(),  
            nn.MaxPool2d(2, 2)  
        )  
        self.layer2 = nn.Sequential(  
            nn.Conv2d(64, 64, kernel_size=3, padding=1),  
            nn.BatchNorm2d(64),  
            nn.ReLU(),  
            nn.MaxPool2d(2, 2)  
        )  
        self.layer3 = nn.Sequential(  
            nn.Conv2d(64, 128, kernel_size=3, padding=1),  
            nn.BatchNorm2d(128),  
            nn.ReLU(),  
            nn.MaxPool2d(2, 2)  
        )  
        self.fc1 = nn.Linear(4 * 4 * 128, 5)  
  
    def forward(self, x):  
        x = self.layer1(x)  
        x = self.layer2(x)  
        x = self.layer3(x)  
        x = x.view(-1, 128 * 4 * 4)  
        x = self.fc1(x)  
        return x
```

- CNN>NN>ResNet
- RNN, DNN, CAB  
尝试但未实现

## 五、参数调整

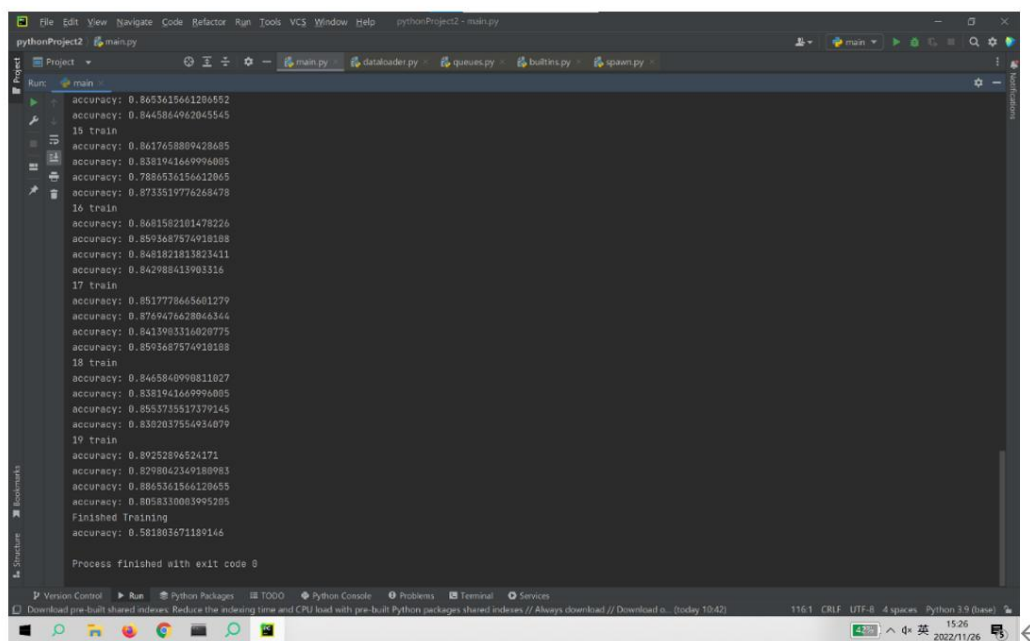
## 六、损失函数和优化器选择

- **Batch\_size:20(15~50)**
- 网络参数
- 优化器: RMS (SGD, Adadel $\epsilon$ ta, Adagrad, Adam, RMS)
- 损失函数: `nn.CrossEntropyLoss()`

# 训练结果

原始模型：58.18%（20 次训练）←

最终经过学习过程，确定使用 CNN 网络，batch\_size=20，选择最适优化器和损失函数，在未优化（数据平衡/图像增强/减噪处理）的情况下，经过 20 次训练，得到训练结果如下所示：←



The screenshot shows a Python IDE window with a terminal output displaying training accuracy results. The output lists accuracy values for each epoch from 15 to 19, followed by a 'Finished Training' message and the final accuracy. The accuracy values range from approximately 0.84 to 0.89.

```
accuracy: 0.8453615661206552
accuracy: 0.8445864962045545
15 train
accuracy: 0.8417458809428605
accuracy: 0.8381941669996005
accuracy: 0.7886536156612065
accuracy: 0.8733519776268478
16 train
accuracy: 0.8681582101478226
accuracy: 0.859368754918108
accuracy: 0.8481821813823411
accuracy: 0.842988413903316
17 train
accuracy: 0.8517778665601279
accuracy: 0.8769476628066364
accuracy: 0.8413983316020775
accuracy: 0.859368754918108
18 train
accuracy: 0.8465840990811027
accuracy: 0.8381941669996005
accuracy: 0.95375517379145
accuracy: 0.8382037554936079
19 train
accuracy: 0.89252896524171
accuracy: 0.8298042149180981
accuracy: 0.886361866120655
accuracy: 0.8068330003995205
Finished Training
accuracy: 0.581803671189166
Process finished with exit code 0
```

# 七、问题解决及模型优化

## 1.图像区分度不够高——图像增强

- 对比度增强

- >矫正光照影响导致的阴影，提高图像中病变区域和背景之间的对比度

- 锐化处理

- >病变更加清晰

- 色度增强

- >病变更明显

# 单张图片增强——PIL.ImageEnhance

```
123 img = Image.open('D:/test_code/DDR/test_image/1.jpg')
124 img.show()
125
126 # 对比度增强
127 enh_con = ImageEnhance.Contrast(img)
128 contrast = 1.5
129 img_contrasted = enh_con.enhance(contrast)
130 img_contrasted.show()
131
132 # 色度增强
133 enh_col = ImageEnhance.Color(img)
134 color = 1.5
135 image_colored = enh_col.enhance(color)
136 image_colored.show()
137
138 # 锐度增强
139 enh_sha = ImageEnhance.Sharpness(img)
140 sharpness = 3.0
141 image_sharped = enh_sha.enhance(sharpness)
142 image_sharped.show()
143
144 img_contrasted.save("D:/test_code/DDR/test_image2/2.jpg")
```



原图



处理图

# 批量图片增强——augly.image

```
img_path = "D:/test_code/DDR/final_valid/44" # 需要增强的图像路径
save_path = "D:/test_code/DDR/final_valid/4" # 保存路径
|
def augly_augmentation(aug_image):
    aug = [
        imaugs.contrast(aug_image, factor=random.uniform(1, 1.5)), # 对比度增强
        imaugs.sharpen(aug_image, factor=1.5), # 锐化
    ]
    return aug[1]
```



原始



对比增强后



对比增强+锐化后←

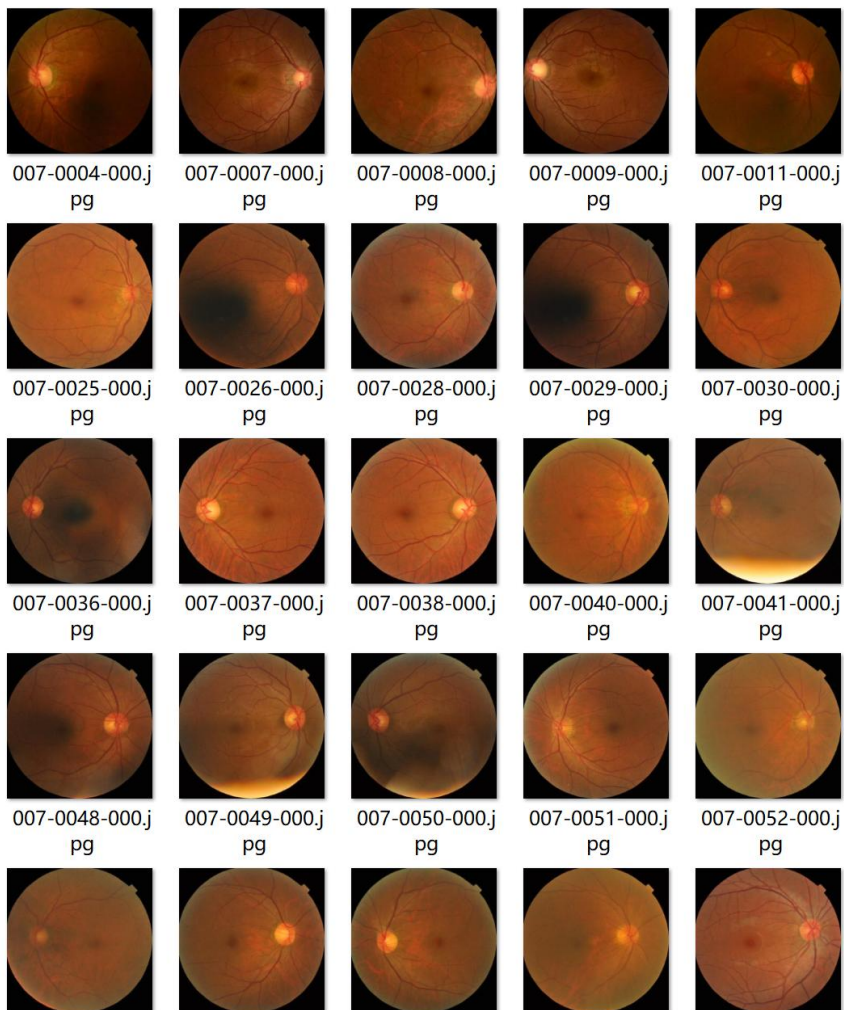


# train

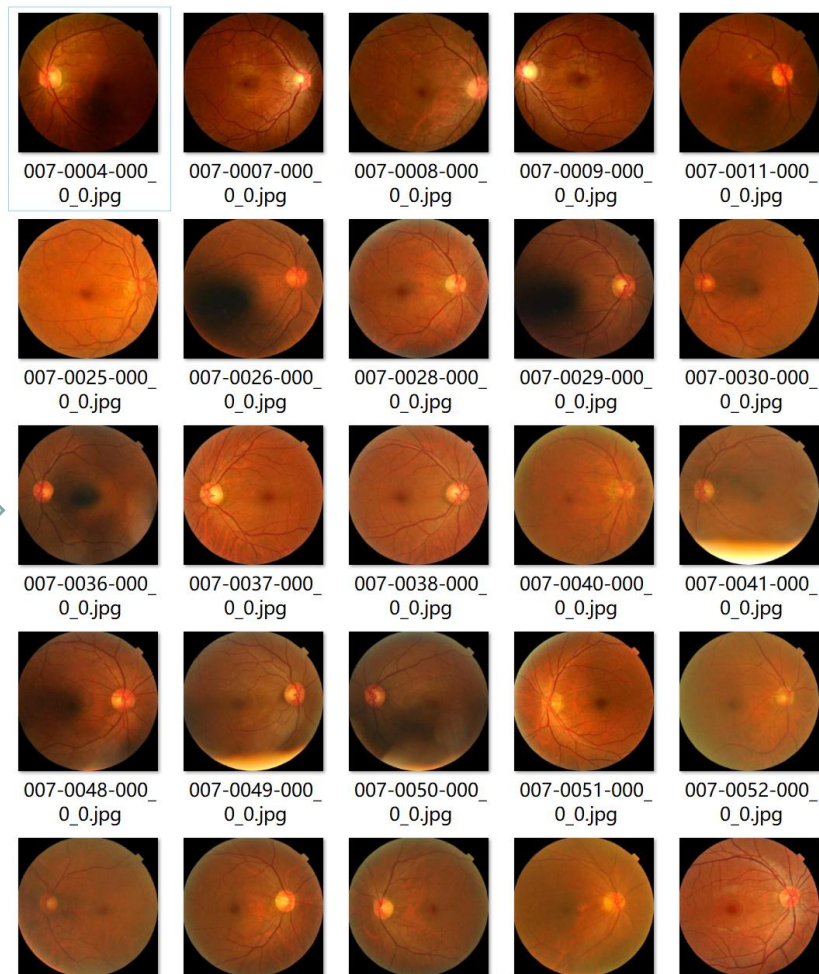


# final\_train

> test\_code > DDR > train > 0



> test\_code > DDR > final\_train > 0

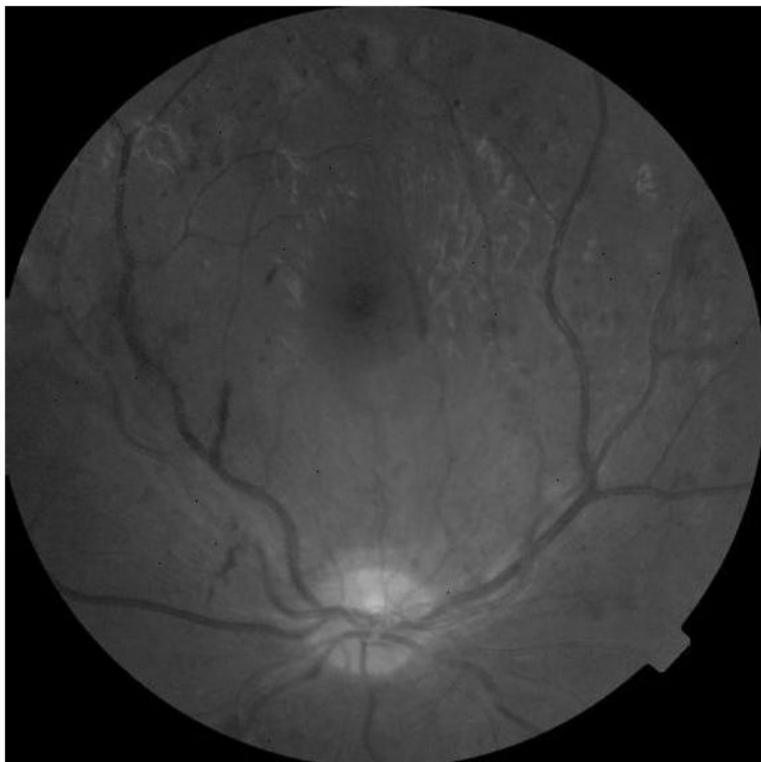


## 2. 无关因素干扰——高斯滤波减噪（卷积实现）

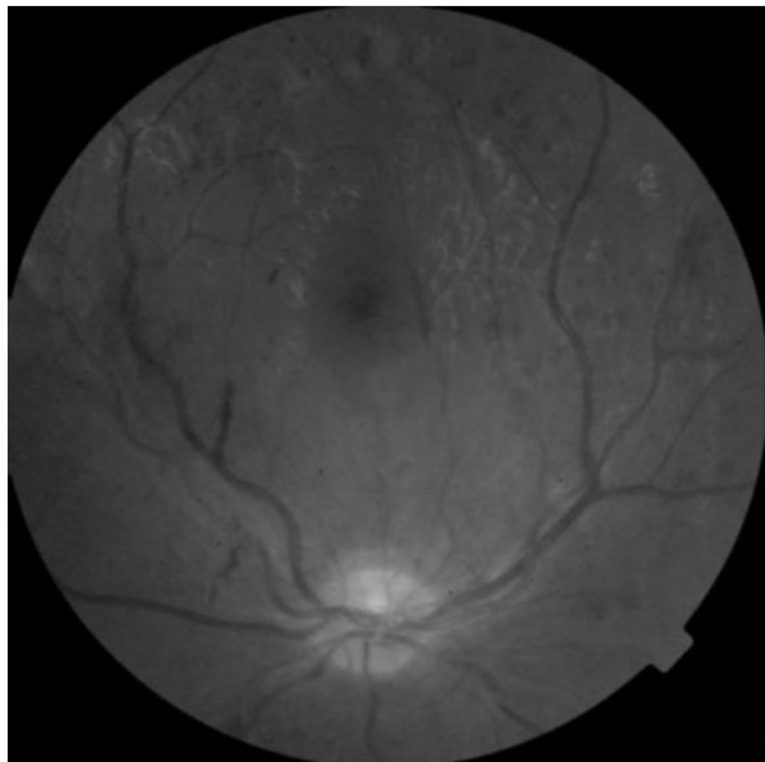
```
# 高斯滤波
def fil(imag, delta=0.7):
    k3 = compute(delta)
    k = k3[0]
    H = k3[1]
    k1 = (k - 1) / 2
    [a, b] = imag.shape
    k1 = int(k1)
    new1 = np.zeros((k1, b))
    new2 = np.zeros(((a + (k - 1)), k1))
    imag1 = np.r_[new1, imag]
    imag1 = np.r_[imag1, new1]
    imag1 = np.c_[new2, imag1]
    imag1 = np.c_[imag1, new2]
    y = np.zeros((a, b))
    sum2 = sum(sum(H))
    for i in range(k1, (k1 + a)):
        for j in range(k1, (k1 + b)):
            y[(i - k1), (j - k1)] = relate(imag1[(i - k1):(i + k1 + 1), (j - k1):(j + k1 + 1)], H, k) / sum2
    return y
```



对单张图像灰度处理和使用高斯滤波处理后，结果如下所示：←



原图像



减噪后图像←

### 3.数据不平衡——过采样

- 方法1:

文件夹扩增

- 方法2:

权重随机采样(WeightedRandomSampler)

## 方法1：低样本量的文件夹图片扩增

- Control+C
- Control+V

```
113     n=[0,0,0,0,0,0]
114     for i in range(len(trainset)):
115         num=trainset[i]
116         n[num]+=1
117     print(n[0],n[1],n[2],n[3],n[4])
```

- 扩增前： [2992,1638,2238,613,2370]
- 扩增后： [2992,2880,2958,3065,2946]

# 方法2：权重随机采样(WeightedRandomSampler)

## 7.1.2.1 采样器功能介绍

WeightedRandomSampler() 函数会根据给定的权重数组，对数据进行尽可能贴近权重的随机采样。采样器会根据采样的个数和权重，对不同数量的数据集进行过采样和欠采样等操作，使得数据尽可能平衡。函数中有三个重要的参数，如下表所示：

↩

| 参数↩                     | 作用↩                        | ↩ |
|-------------------------|----------------------------|---|
| <u>samples_weight</u> ↩ | 权重数组↩                      | ↩ |
| <u>samples_num</u> ↩    | 需要采样的个数↩                   | ↩ |
| replacement↩            | 为 True 时可重复采样，为 False 时不可↩ | ↩ |

↩

(2) 通过 7.1.1 提到的分类别计数的语句，统计样本中各类别的个数，输出结果如下图所示：↵

```
preparing data...
985
采样
[517, 36, 297, 9, 126] ↵
```

```
113     n=[0,0,0,0,0,0]
114     for i in range(len(trainset)):
115         num=trainset[i]
116         n[num]+=1
117     print(n[0],n[1],n[2],n[3],n[4])
```

(3) 计算权重数组↵

为实现数据采样后的平衡，应当使得数量较大的类别采样权重赋小，数量较小的类别权重赋大。因而不同类别数据，权重赋为其数量占比的倒数，是最简单的方法。即  $weight[i]=len(dataset)/n[i]$ ↵

赋值之后输出采样结果，如图所示：↵

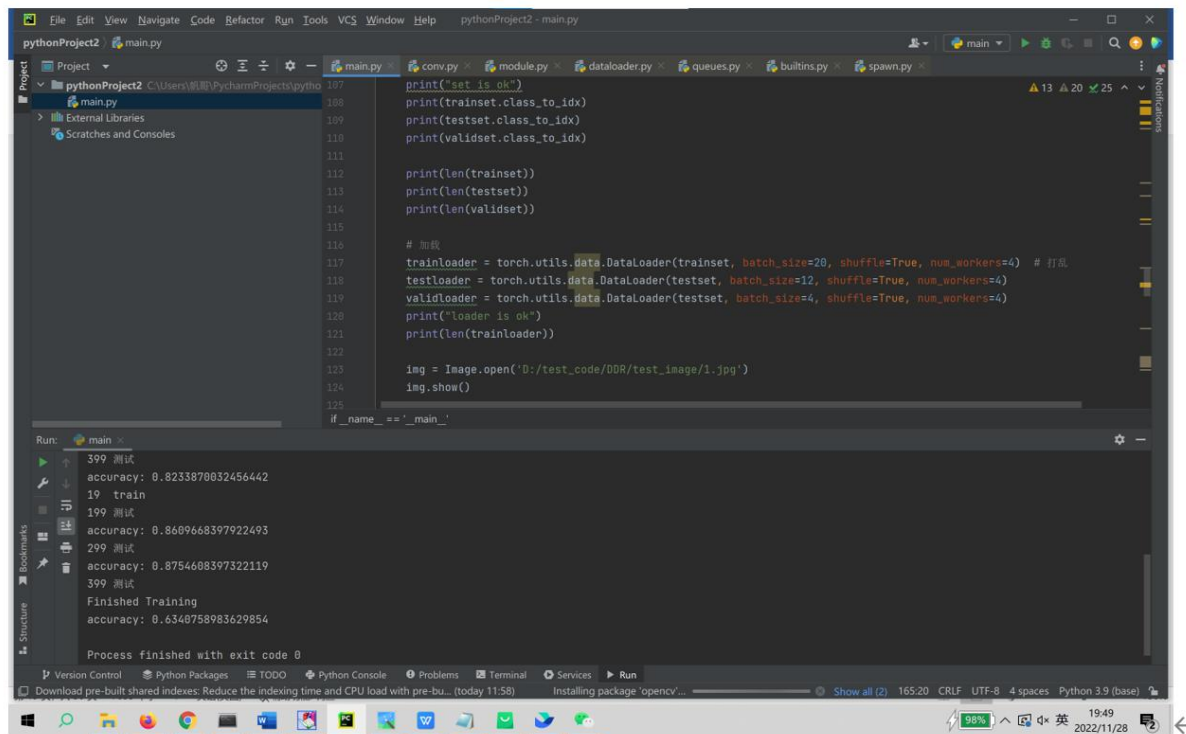
```
[985, 0, 0, 0, 0]
```

只采集了类别为 0 的数据，并且多次尝试更换参数后，依然得不到与预期相符合的结果。只能放在后期的预期优化中，日后重新尝试。↵

# 训练结果

优化模型：63.40%（20 次训练）←

其他量保持不变，经过图像增强之后的结果如下所示：←



The screenshot shows a PyCharm IDE window with a Python script and its execution results. The script is located in a file named `main.py` within a project named `pythonProject2`. The script contains the following code:

```
print("set is ok")
print(trainset.class_to_idx)
print(testset.class_to_idx)
print(validset.class_to_idx)

print(len(trainset))
print(len(testset))
print(len(validset))

# 加载
trainloader = torch.utils.data.DataLoader(trainset, batch_size=20, shuffle=True, num_workers=4) # 打印
testloader = torch.utils.data.DataLoader(testset, batch_size=12, shuffle=True, num_workers=4)
validloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=True, num_workers=4)
print("loader is ok")
print(len(trainloader))

img = Image.open('D:/test_code/DDR/test_image/1.jpg')
img.show()

if __name__ == '__main__':
```

The execution results are displayed in the Run console, showing the following output:

```
Run: main
399 测试
accuracy: 0.8233870032456442
19 train
199 测试
accuracy: 0.8609668397922493
299 测试
accuracy: 0.8754608397322119
399 测试
Finished Training
accuracy: 0.6340758983629854
Process finished with exit code 0
```

The bottom status bar of the IDE indicates the current configuration: Python 3.9 (base), 165-20, CRLF, UTF-8, 4 spaces, and the date 2022/11/28.

## 六、预期优化

### 1. 深层网络

期待能在日后深入学习卷积神经网络后，理解各层、各参数的意义，能够跑通并且学会使用更深层、性能更好的网络

### 2. 增加训练次数

由于 C 盘内存限制，本次机器学习最多只能正常训练到 20 次。而这个学习次数是远远不够训练出拟合较好的参数的。希望在日后能够留足硬盘空间，也能够学会熟练使用 Linux 系统，增加训练次数，不断优化模型

### 3. 数据不平衡问题

本文提出的两种解决数据不平衡问题的方法，文件夹扩增法没有达到预期准确率，权重随机取样法最终没能实现。在日后需要研究前者与理论相悖的原因，后者如何实现，以及尝试其他采样器或新方案实现样本数据平衡

### 4. 降噪批量处理

本次只实现了高斯滤波器的单张降噪处理，批量处理需要在日后完善

### 5. 分类别计数器的优化

逐个判断，复杂度过高，耗时过长。可以查询相关函数，或分类别装入 5 个 dataset 中，用 `len()` 计数

谢谢