人工智能作业4

1852824 吴杨婉婷

1. 作业需求描述

- 给出的数据集中的数据为贴片电阻的焊点图片。图片是某芯片产线上焊接电阻后由视觉检测系统所拍摄的焊点图片。数据集按照电阻的类型分为三类（101K,102K,331K）。每类又分为正常焊点（normal）和异常焊点（abnormal）。将normal图片的分类标签记为1，将abnormal图片的分类标签记为0。
- 训练数据在3300，正常图片2800，异常图片480

2. Requirements

- **Development Environment:**

  Win 10

- **Development Software:**

  PyCharm *2020.3.5.PC-191.6605.12*

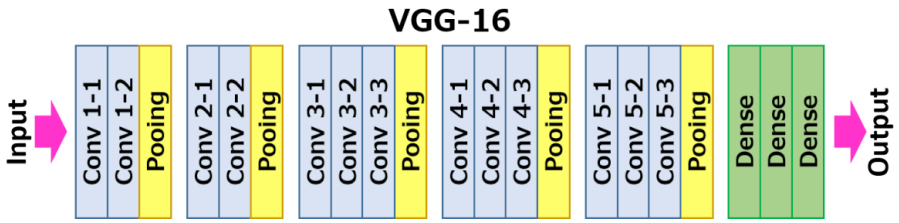
- **Development Language:**

  Python

- **Mainly Reference Count:**

1. torchvision
2. matplotlib
3. os
4. torch
5. numpy
6. random

3. 模型代码详解

number	描述	model
1	简单cnn先用6个5* 5的卷积核，然后进行池化，再用16个5*5的卷积核，最后经过三层全连接	Simple_CNN_Net
2	对vgg16net做了少许改动，删去其中的一些层 	VGG

```
class Simple_CNN_Net(nn.Module):
    def __init__(self):
        super(Simple_CNN_Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(17424, 120)
        self.fc2 =nn.Sequential(nn.Linear(120,84), torch.nn.Dropout(0.3), )
        self.fc3 = nn.Linear(84, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 17424)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

- vgg16中删除一些层
- 只保留

1. 64个3*3卷积核
2. 64个3*3卷积核
3. 128个3*3卷积核
4. 128个3*3卷积核
5. 256个3*3卷积核
6. 256个3*3卷积核
7. 512个3*3卷积核
8. 512个3*3卷积核
9. 全连接
10. 全连接

```
class VGG(nn.Module):
    def __init__(self, num_classes=2):
```

```
        super(VGG15, self).__init__()
        self.features = nn.Sequential(
            # 1
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            # 2
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # 3
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            # 4
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # 5
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            # 6
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # 7
            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # 8
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.AvgPool2d(kernel_size=1, stride=1),
        )
        self.classifier = nn.Sequential(
            # 9
            nn.Linear(8192, 100),
            nn.ReLU(True),
            nn.Dropout(),
            # 10
            nn.Linear(100, num_classes),
        )
        # self.classifier = nn.Linear(512, 2)

    def forward(self, x):
        out = self.features(x)
        #        print(out.shape)
```

```
        out = out.view(out.size(0), -1)
        #         print(out.shape)
        out = self.classifier(out)
        #         print(out.shape)
        return out
```

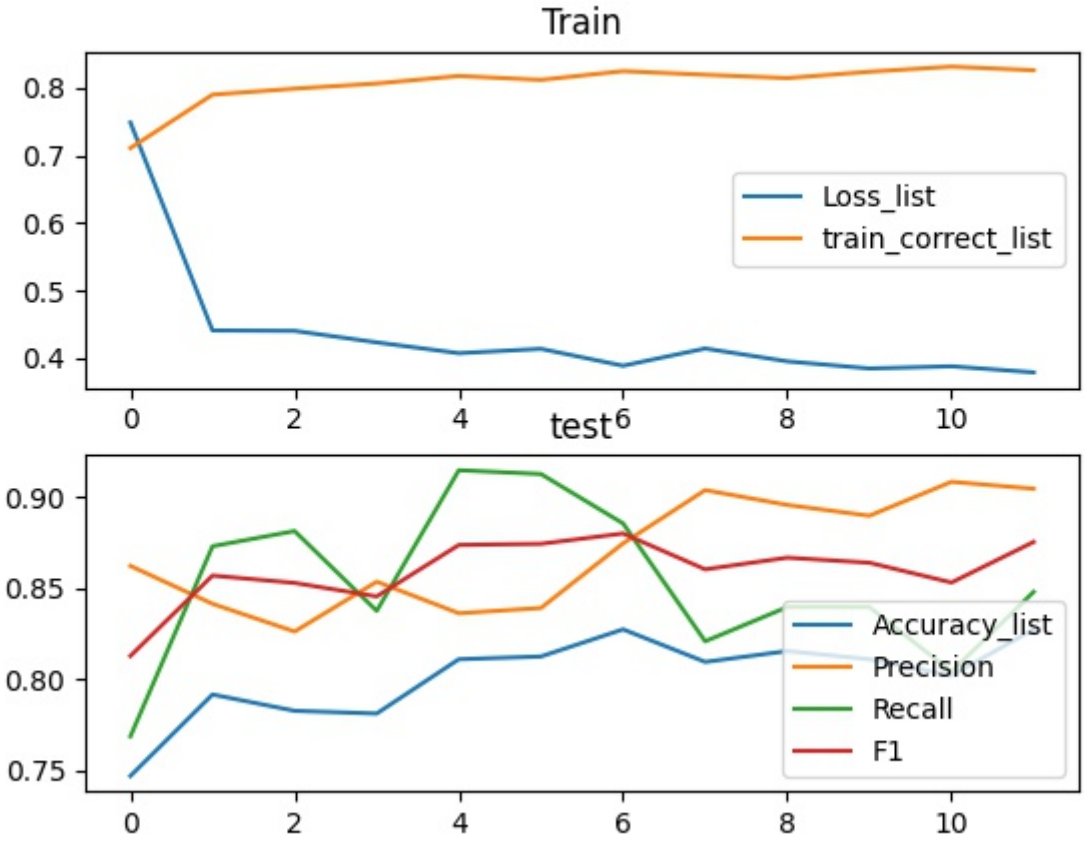
4.实验训练与结果

number	model	acc	Precision	Recall	F1
1	两层cnn	0.9091	0.8630	0.91875	0.8900
2	vgg改良	0.8892	0.9044	0.8479	0.8752

4.1 两层cnn

- 详细训练过程见 **4.2 vgg改良**，此处仅有训练结果
- 首先训练30个epoch

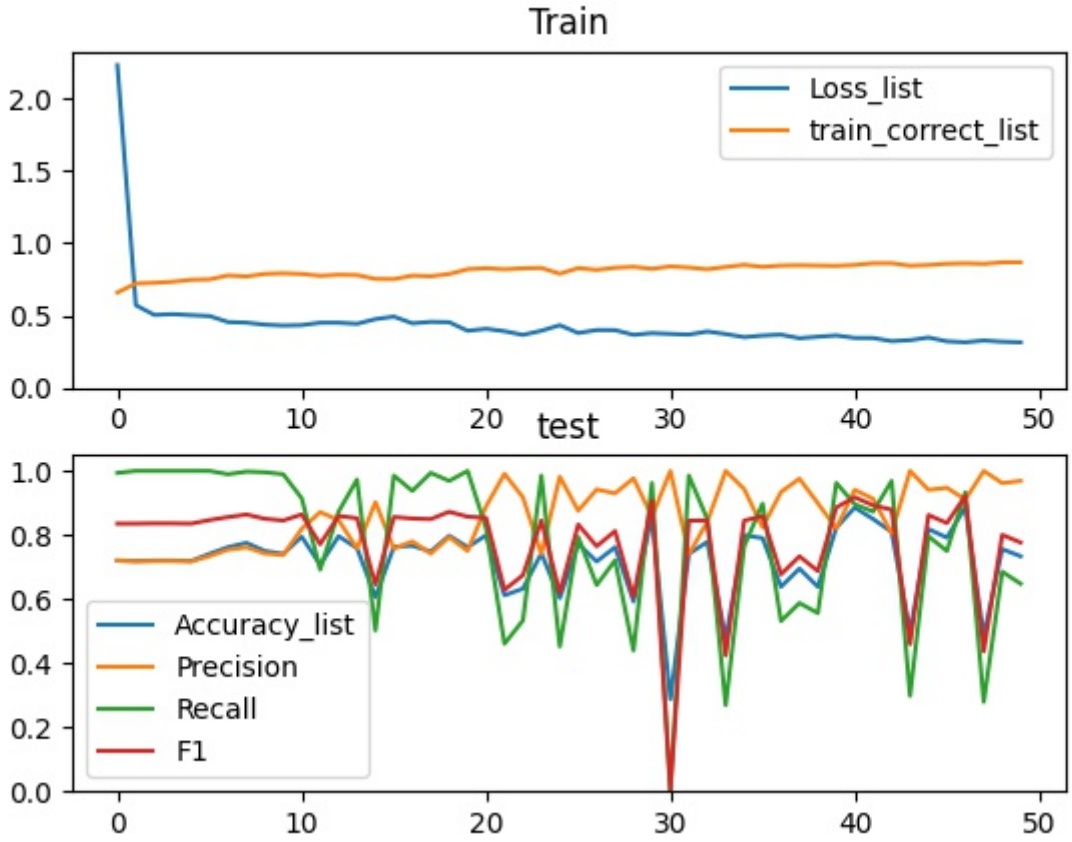
- 再调整学习率到1e-5进行微调，同时调整l2参数



4.2 vgg改良

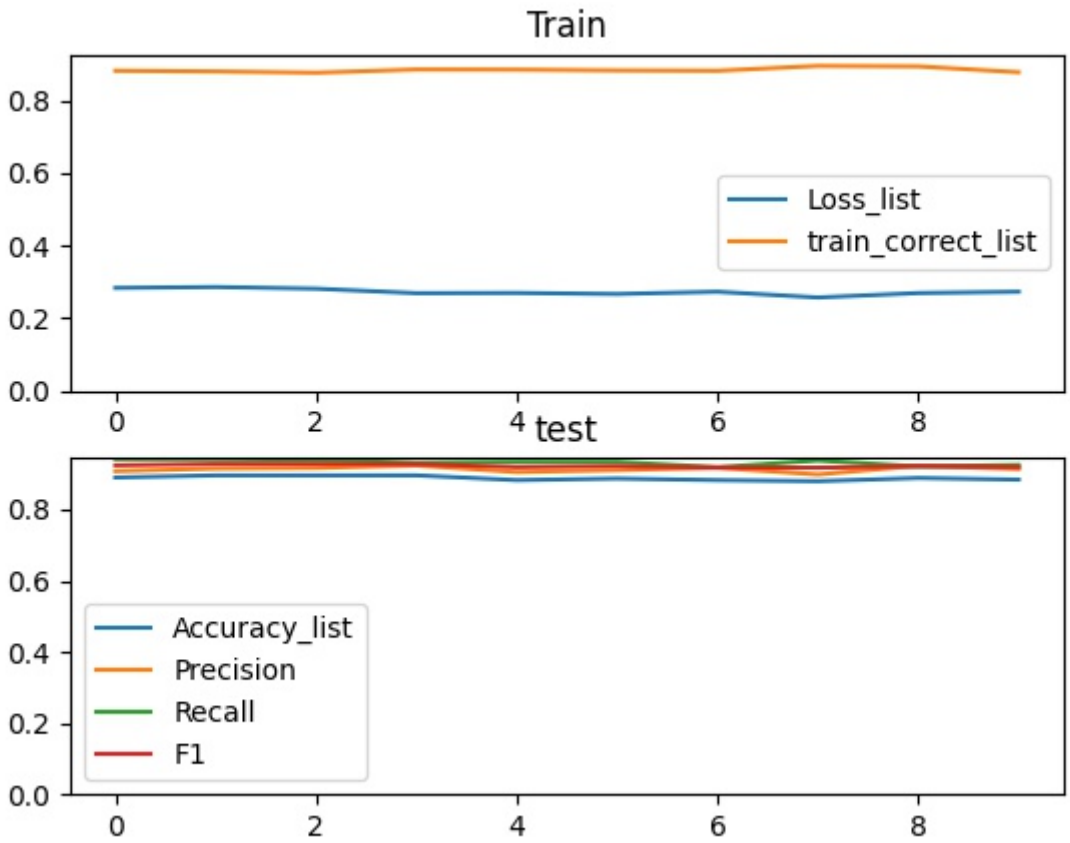
- 首先训练30个epoch,不知道为什么这里精确率以及召回率的波动这么大，我觉的应该是正样本和负样本的数量不大一致，虽然我自己又增加了新的数据，让负样本的值达到了1500，但效果就是会来回震荡，

考虑是权重衰减参数比较小，没有起到足够大的平滑作用



- 再调整学习率到1e-5进行微调，同时调整l2参数，但可以从图中看出来，学习率到1e-7之后学习效果并没有上升，无疑是数据量太小了，总量4000左右的数据集不够大，考虑使用gan对抗网络来进行数据增

强，由于使用的dcgan训练出来的数据肉眼可见不大清晰，放弃使用gan来进行数据增强



- 核心代码
- 首先是数据增强，正样本有2800，但负样本只有480，于是把负样本扩充达到1500来提高训练效果

```
def copyImg():
    sourcePath = './sample/0/'
    # 指定图片原始路径A
    targetPath = './sample/0/'
    # 指定图片存放目录B
    for i in range(480):
        shutil.copy(sourcePath + '{}.png'.format(i) , targetPath
+'{}.png'.format(i+480) )
```

- 优化方式为mini-batch momentum-SGD，并采用L2正则化（权重衰减）LR = 0.0007

```
# 损失函数为交叉熵，多用于多分类问题
optimizer = optim.SGD(net.parameters(), lr=LR, momentum=0.9,
                      weight_decay=5e-4)
# 优化方式为mini-batch momentum-SGD，并采用L2正则化（权重衰减）
net = mymodel.ResNet18().to(device)
```

- 将图片进行Resize，RandomCrop，RandomHorizontalFlip，和归一化

```
transform=transforms.Compose([

transforms.Resize(160),

transforms.RandomCrop(144, padding=4),  # 先四周填充0，在吧图像随机裁剪成144
transforms.RandomHorizontalFlip(),  # 图像一半的概率翻转，一半的概率不翻转
transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

5.结果分析与心得体会

5.1 指标说明

	预测类别			
		Yes	No	总计
实际类别	Yes	TP	FN	P（实际为Yes）
	No	FP	TN	N（实际为No）
	总计	P'（被分为Yes）	N'（被分为No）	P+N

1. 准确率（Accuracy）

- 准确率(accuracy)计算公式为：准确率（正确率）=所有预测正确的样本/总的样本数量 =（TP+TN）/总
- 准确率是我们最常见的评价指标，而且很容易理解，就是被分对的样本数除以所有的样本数，通常来说，正确率越高，分类器越好。 准确率确实是一个很好很直观的评价指标，但是有时候准确率高并不能代表一个算法就好。比如某个地区某天地震的预测，假设我们有一堆的特征作为地震分类的属性，类别只有两个：0：不发生地震、1：发生地震。一个不加思考的分类器，对每一个测试用例都将类别划分为0，那那么它就可能达到99%的准确率，但真的地震来临时，这个分类器毫无察觉，这个分类带来的损失是巨大的。为什么99%的准确率的分类器却不是我们想要的，因为这里数据分布不均衡，类别1的数据太少，完全错分类别1依然可以达到很高的准确率却忽视了我们关注的东西。再举个例子说明下。在正负样本不平衡的情况下，准确率这个评价指标有很大的缺陷。比如在互联网广告里面，点击的数量是很少的，一般只有千分之几，如果用acc，即使全部预测成负类（不点击）acc也有 99% 以上，没有意义。因此，单纯靠准确率来评价一个算法模型是远远不够科学全面的，这一点在本次作业中也有体现。

2. 精确率(Precision)

- 精确率(precision)定义为：精确率= 将正类预测为正类 / 所有预测为正类 =TP/（TP+FP）
- 表示被分为正例的示例中实际为正例的比例。

3. 召回率（recall）

- 召回率是覆盖面的度量，度量有多个正例被分为正例，召回率 = 将正类预测为正类 / 所有真正的正类= TP/（TP+FN）

4. 综合评价指标（F-Measure）

- P和R指标有时候会出现的矛盾的情况，这样就需要综合考虑他们，最常见的方法就是F-Measure（又称为F-Score）。 F-Measure是Precision和Recall加权调和平均： 最常见的F1，也即F1 = 2 * 精确率 * 召回率 / (精确率 + 召回率) （F 值即为精确率和召回率的调和平均值） 可知F1综合了P和R的结果，当F1较高时则能说明试验方法比较有效。

5.2 VGG

- VGG16一个改进是采用连续的几个3x3的卷积核代替AlexNet中的较大卷积核（11x11，7x7，5x5）。对于给定的与输出有关的输入图片的局部大小，采用堆积的小卷积核是优于采用大的卷积核，因为多层非线性层可以增加网络深度来保证学习更复杂的模式，而且代价还比较小（参数更少）。在VGG中，使用了3个3x3卷积核来代替7x7卷积核，使用了2个3x3卷积核来代替5*5卷积核，这样做的主要目的是在保证具有相同感知野的条件下，提升了网络的深度，在一定程度上提升了神经网络的效果，减少了参数，而且3x3卷积核有利于更好地保持图像性质。

- 我这里进行调整，我的VGG包含了10个隐藏层（8个卷积层和2个全连接层）

- VGG优点

  VGGNet的结构非常简洁，整个网络都使用了同样大小的卷积核尺寸（3x3）和最大池化尺寸（2x2）。几个小滤波器（3x3）卷积层的组合比一个大滤波器（5x5或7x7）卷积层好：验证了通过不断加深网络结构可以提升性能。

- VGG缺点

  VGG耗费更多计算资源，并且使用了更多的参数（这里不是3x3卷积的），导致更多的内存占用（25M）。其中绝大多数的参数都是来自于第一个全连接层。

5.3 微调与总结

- 我认为最有用的微调是根据训练数据准确率，loss,方差等调整学习率，考虑引入早衰，在loss变化的特别小甚至因为过拟合准确率下降的时候提前结束训练
- 采用optim.SGD优化器
- 这里有个疑惑的地方，在准确率低于85时不断调整学习率对准确度的提高很有帮助，但最后的准确率达到0.90左右的时候无论如何调整参数都没有办法再提高学习率了，不知道是否是陷入了局部最优解
- 首先训练30个epoch,不知道为什么这里精确率以及召回率的波动这么大，我觉的应该是正样本和负样本的数量不大一致，虽然我自己又增加了新的数据，让负样本的值达到了1500，但效果就是会来回震荡，考虑是权重衰减参数比较小，没有起到足够大的平滑作用
- 再调整学习率到1e-5进行微调，同时调整l2参数，但可以从图中看出来，学习率以及到1e-7之后学习效果并没有上升，无疑是数据量太小了，总量4000左右的数据集不够大，考虑使用gan对抗网络来进行数据增强，由于使用的dcgan训练出来的数据肉眼可见不大清晰，放弃使用gan来进行数据增强
- 最后可以看出来简单cnn的效果会比有8层的深度卷积网络效果更好，考虑还是数据集太小了，太深度的网络反而会导致梯度消失等，参数太多也不太好，这一次作业我没有考虑数据集不够多，大型网络不好用，此处应该考虑在vgg基础上加上dropout和残差网络，或许会有些帮助。

6.主体代码解释

- 首先是数据增强，正样本有2800，但负样本只有480，于是把负样本扩充达到1500来提高训练效果

```
def copyImg():
    sourcePath = './sample/0/'
    # 指定图片原始路径A
    targetPath = './sample/0/'
    # 指定图片存放目录B
    for i in range(480):
        shutil.copy(sourcePath + '{}.png'.format(i) , targetPath
+'{}.png'.format(i+480) )
```

- 数据集下载

```
# 准备数据集并预处理
    train_data = torchvision.datasets.ImageFolder('./sample',
                                                  transform=transforms.Compose([

                                                      transforms.Resize(160),

                                                      transforms.RandomCrop(144,
padding=4),  # 先四周填充0，在吧图像随机裁剪成144


                                                    transforms.RandomHorizontalFlip(),
# 图像一半的概率翻转，一半的概率不翻转

transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
                                                  )
# 取20%作为测试集，并且打乱顺序
n = len(train_data)
lista=[i for i in range(0,n,5)]
random.shuffle(lista)
listb=[i for i in range(n) if i not in lista]
random.shuffle(listb)
# print(listb)
# print(lista)
test_loader = torch.utils.data.Subset(train_data,lista)
test_loader = torch.utils.data.DataLoader(test_loader, batch_size=BATCH_SIZE,
                                                   shuffle=True, num_workers=2)
train_loader = torch.utils.data.Subset(train_data,listb)
train_loader = torch.utils.data.DataLoader(train_loader, batch_size=BATCH_SIZE,
                                                  shuffle=False, num_workers=2)
```

- 模型定义

```
# 模型定义
    net = VGG().to(device)
    # 定义损失函数和优化方式
    criterion = torch.nn.CrossEntropyLoss()
```

```
    # 损失函数为交叉熵，多用于多分类问题
    optimizer = optim.SGD(net.parameters(), lr=LR, momentum=0.9,
                          weight_decay=5e-4)
    # 优化方式为mini-batch momentum-SGD，并采用L2正则化（权重衰减）
    net = mymodel.ResNet18().to(device)
    # 取得之前的参数
    # net.load_state_dict(torch.load('./parameterForme/parameter1.pkl'))
```

- 训练开始

```
# 训练
for _ in range(epoch):
net.train()
sum_loss = 0.0
train_correct=0
total_train=0
for i, data in enumerate(train_loader):
    inputs, labels = data
    # print(i)
    # print(labels)
    # print(labels)
    # print('type(images) = ', type(inputs))
    # print('type(labels) = ', type(labels))
    # labels=torch.from_numpy(labels)
    inputs, labels = Variable(inputs).to(device), Variable(labels).to(device)
    optimizer.zero_grad()  # 将梯度归零
    outputs = net(inputs)  # 将数据传入网络进行前向运算
    loss = criterion(outputs, labels)  # 得到损失函数
    loss.backward()  # 反向传播
    optimizer.step()  # 通过梯度做一步参数更新
    # print(loss)
    sum_loss += loss.item()
    __, predicted = torch.max(outputs.data, 1)
    total_train += labels.size(0)
    train_correct += (predicted == labels).sum()
    if i % 35 == 34:
        print('[%d,%d] loss:%.03f,test_acc::%.03f' %
              (_ + 1, i + 1, sum_loss / 35,train_correct/total_train))
        Loss_list.append(sum_loss /35)
        sum_loss = 0.0
        train_correct_list.append(train_correct/total_train)
    # 每训练完一个epoch测试一下准确率
    print("Waiting Test!")
    net.eval()
    with torch.no_grad():
        correct = 0
        total = 0
        TP=0
        TN=0
        FN=0
        FP=0
        for data in test_loader:
```

```
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            # 取得分最高的那个类 (outputs.data的索引号)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum()
            TP += ((predicted == labels) & (labels ==
torch.ones_like(labels))).sum().item()
            TN += ((predicted == labels) & (labels ==
torch.zeros_like(labels))).sum().item()
            FN += ((predicted != labels) & (labels ==
torch.ones_like(labels))).sum().item()
            FP += ((predicted != labels) & (labels ==
torch.zeros_like(labels))).sum().item()
        print('测试分类准确率为：%.3f%%' % (100 * correct / total))
        a1 = (TP + TN) / (FN + FP + TP + TN)
        a2 = TP / (TP + FP)
        a3 = TP / (TP + FN)
        a4 = 2 * a2 * a3 / (a2 + a3)
        print("准确率={},精确率={},召回率={},F值={}".format(a1, a2, a3, a4))
        acc = correct / total
        Accuracy_list.append(acc)
        a2_list.append(a2)
        a3_list.append(a3)
        a4_list.append(a4)
```

- 开始测试

```
  net.eval()  # 将模型变换为测试模式
    correct = 0
    total = 0
    for data_test in test_loader:
        images, labels = data_test
        images, labels = Variable(images).to(device), Variable(labels).to(device)
        output_test = net(images)
        _, predicted = torch.max(output_test, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum()
    print("correct: ", correct)
    print("Test acc: {0}".format(correct.item() / len(test_loader.dataset)))
```

- 保存参数

```
# 保存
if not os.path.exists("./parameterForme"):
    os.mkdir("./parameterForme")
torch.save(net.state_dict(), './parameterForme/parameter1.pkl')
```

- 作图

```
# 作图
    plt.subplot(2, 1, 1)

    plt.plot(Loss_list, label='Loss_list')
    plt.plot(train_correct_list, label='train_correct_list')
    plt.title("Train")

    plt.subplot(2, 1, 2)
    plt.plot(Accuracy_list, label='Accuracy_list')
    plt.plot(a2_list, label='Precision')
    plt.plot(a3_list, label='Recall')
    plt.plot(a4_list, label='F1')
    plt.title("test")
    plt.legend()
    plt.savefig("./parameterForme/pic_1_0.jpg")
```

7.作者

ID	Name
1852824	吴杨婉婷

指导老师 唐堂老师

联系方式 email: 1852824@tongji.edu.cn