口罩佩戴检测程序报告

学号：2111460　　姓名：张洋

一、问题重述

1. 建立深度学习模型，检测出图中的人是否佩戴了口罩，并将其尽可能调整到最佳状态。
2. 学习经典的模型 mtcnn 和 mobilenet 的结构。
3. 可以使用基于 Python 的 OpenCV 、PIL 库进行图像相关处理，使用 Numpy 库进行相关数值运算，使用 MindSpore、Keras 等框架建立深度学习模型等。
4. 学习训练时的方法。

二、设计思想

1. 加载数据并进行数据预处理

第一步：创建一个 Dataset 对象

torchvision.datasets.ImageFolder 是一个通用的数据加载器，常见的用法如下：dataset=torchvision.datasets.ImageFolder(root, transform=None, target_transform=None, loader=, is_valid_file=None)

第二步：创建一个 DataLoader 对象

DataLoader 是 torch 用来包装数据的工具，所以要将(numpy array 或其他) 数据形式装换成 Tensor, 然后再放进这个包装器中。使用 DataLoader 帮助我们对数据进行有效地迭代处理。

2. 加载预训练模型

本实验中使用已经搭建好的 MTCNN 网络，预训练模型 MobileNetV1。

第一步：MTCNN 网络实现人脸检测功能，效果如下图：

第二步：加载预训练模型 MobileNetV1

(1)调用预处理函数，将数据集划分

(2)调用 torch.py 里面的 MobileNetV1 网络，更改网络深度和 epoch

(3)设置优化器 optimizer

3. 创建模型和训练模型，把训练好的模型保存在 results 文件夹下

第一步：调整学习率

使用 optim.lr_scheduler.ReduceLROnPlateau，根据测试指标调整学习率。当参考的评价指标停止改进时,降低学习率：factor 为每次下降的比例；训练过程中，当指标连续 patience 次数还没有改进时，降低学习率。

第二步：训练模型

不断更改 epochs 和 batch 的值，使得模型最优

第三步：把训练好的模型存入 results 文件夹下

4. 预测

将 cv2.imread 图像转化为 PIL.Image 图像，用来兼容测试输入的 cv2 读取的图像

cv2.imread 读取图像的类型是 numpy.ndarray

PIL.Image.open 读取图像的类型是 PIL.JpegImagePlugin.JpegImageFile

加载之前训练好的最优模型，利用训练好的模型进行预测

三、代码内容

1. 加载数据并进行数据处理

```
def processing_data(data_path, height=224, width=224, batch_size=32,
                    test_split=0.1):
    """
    数据处理部分
    :param data_path: 数据路径
    :param height:高度
    :param width: 宽度
    :param batch_size: 每次读取图片的数量
    :param test_split: 测试集划分比例
    :return:
    """
    transforms = T.Compose([
        T.Resize((height, width)),
        T.RandomHorizontalFlip(0.1),  # 进行随机水平翻转
        T.RandomVerticalFlip(0.1),  # 进行随机竖直翻转
        T.ToTensor(),  # 转化为张量
        T.Normalize([0], [1]),  # 归一化
    ])

    dataset = ImageFolder(data_path, transform=transforms)
    # 划分数据集
    train_size = int((1-test_split)*len(dataset))
    test_size = len(dataset) - train_size
    train_dataset, test_dataset =
torch.utils.data.random_split(dataset, [train_size, test_size])
    # 创建一个 DataLoader 对象
    train_data_loader = DataLoader(train_dataset,
batch_size=batch_size,shuffle=True)
    valid_data_loader = DataLoader(test_dataset,
batch_size=batch_size,shuffle=True)
```

```
    return train_data_loader, valid_data_loader
```

2. 加载预训练模型

```
# 加载 MobileNet 的预训练模型权
device = torch.device("cuda:0") if torch.cuda.is_available() else
torch.device("cpu")
train_data_loader, valid_data_loader =
processing_data(data_path=data_path, height=160, width=160,
batch_size=32)
modify_x, modify_y = torch.ones((32, 3, 160, 160)), torch.ones((32))

epochs = 70
model = MobileNetV1(classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)  # 优化器
```

MobileNetV1 网络的代码

```
class MobileNetV1(nn.Module):
    def __init__(self, classes=2):
        super(MobileNetV1, self).__init__()
        self.mobilebone = nn.Sequential(
            self._conv_bn(3, 32, 2),
            self._conv_dw(32, 64, 1),
            self._conv_dw(64, 128, 2),
            self._conv_dw(128, 128, 1),
            self._conv_dw(128, 256, 2),
            #self._conv_dw(256, 256, 1),
            #self._conv_dw(256, 512, 2),
            #self._top_conv(512, 512, 5),
            #self._conv_dw(512, 1024, 2),
            #self._conv_dw(1024, 1024, 1),
        )
        # self.avgpool = nn.AvgPool2d(kernel_size=7, stride=1)
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(256, classes)
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, (2. / n) ** .5)
            if isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
```

```
    def forward(self, x):
        x = self.mobilebone(x)
        x = self.avg_pool(x)
        x = x.view(x.size(0), -1)
        out = self.fc(x)

        return out

    def _top_conv(self, in_channel, out_channel, blocks):
        layers = []
        for i in range(blocks):
            layers.append(self._conv_dw(in_channel, out_channel, 1))
        return nn.Sequential(*layers)

    def _conv_bn(self, in_channel, out_channel, stride):
        return nn.Sequential(
            nn.Conv2d(in_channel, out_channel, 3, stride, padding=1,
bias=False),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(inplace=True),
        )

    def _conv_dw(self, in_channel, out_channel, stride):
        return nn.Sequential(
            #nn.Conv2d(in_channel, in_channel, 3, stride, 1,
groups=in_channel, bias=False),
            #nn.BatchNorm2d(in_channel),
            #nn.ReLU(inplace=True),
            nn.Conv2d(in_channel, out_channel, 1, 1, 0, bias=False),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(inplace=False),
        )
```

3. 创建模型和训练模型，把训练好的模型保存在 results 文件夹下

```
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                 'max',
                                                 factor=0.2,
                                                patience=12)
criterion = nn.CrossEntropyLoss()


best_loss = 1e9
```

```
best_model_weights = copy.deepcopy(model.state_dict())
loss_list = []  # 存储损失函数值
for epoch in range(epochs):
    model.train()

    for batch_idx, (x, y) in tqdm(enumerate(train_data_loader, 1)):
        x = x.to(device)
        y = y.to(device)
        pred_y = model(x)

        loss = criterion(pred_y, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if loss < best_loss:
            best_model_weights = copy.deepcopy(model.state_dict())
            best_loss = loss

        loss_list.append(loss)

    print('step:' + str(epoch + 1) + '/' + str(epochs) + ' || Total
Loss: %.4f' % (loss))
torch.save(model.state_dict(), './results/temp.pth')
print('Finish Training.')
```

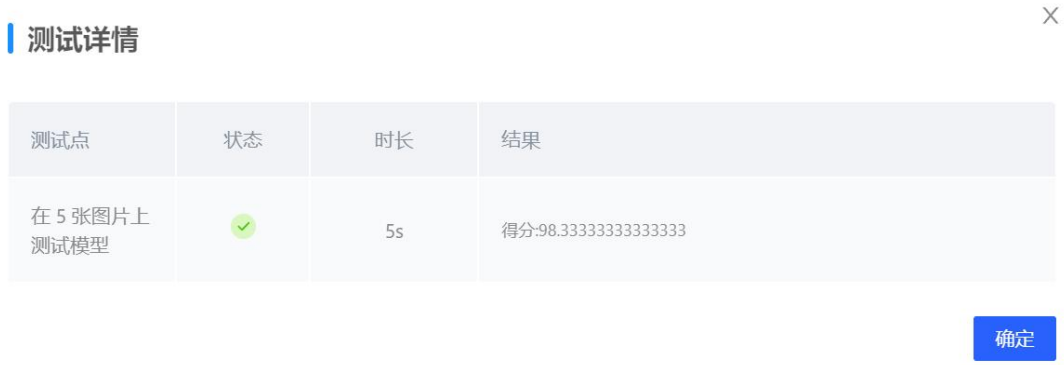
4. 预测

```
def predict(img):
    """
    加载模型和模型预测
    :param img: cv2.imread 图像
    :return: 预测的图片中的总人数、其中佩戴口罩的人数
    """
    # -------------------------- 实现模型预测部分的代码
--------------------------
    # 将 cv2.imread 图像转化为 PIL.Image 图像，用来兼容测试输入的 cv2 读取的
图像（勿删！！！）
    # cv2.imread 读取图像的类型是 numpy.ndarray
    # PIL.Image.open 读取图像的类型是 PIL.JpegImagePlugin.JpegImageFile
    if isinstance(img, np.ndarray):
        # 转化为 PIL.JpegImagePlugin.JpegImageFile 类型
        img = Image.fromarray(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

```
    recognize = Recognition(model_path)
    img, all_num, mask_num = recognize.mask_recognize(img)
    #
----------------------------------------------------------------------
--
    return all_num,mask_num
```

四、实验结果

平台检测结果：



平台检测评分为 98.33，模型达到了较好的效果。

尝试优化的部分：

1. 调整 epoch

Epoch 的定义：当一个完整的数据集通过了神经网络一次并且返回了一次，这个过程称为一次 epoch，即一个 epoch 就是将所有训练样本训练一次的过程。

适当增大 epoch 可以让模型更好地学习数据集中的模式和规律，因为每个 epoch 都会迭代一遍整个数据集，让模型有更多的机会学习到数据集中的特征和模式。此外，增大 epoch 还可以帮助模型克服过拟合问题，因为模型在经过多个 epoch 的训练后，可以更好地泛化到新的数据上。然而，需要注意的是，增大 epoch 也可能导致模型过拟合，因此需要在验证集上进行监测，以避免过拟合。

模型初始设置的 epoch 为 2，过于小，模型可能尚未拟合收敛。因此，我们在调整参数的过程中，逐渐增大 epoch，发现其值为 70 的时候，模型达到最好的效果。

2. 调整学习率

在训练神经网络模型时，学习率是一个非常重要的超参数。学习率决定了每次参数更新的步长大小，因此对于模型的训练效果和速度都有很大的影响。如果学习率过大，会导致模型在训练过程中发生震荡或者无法收敛；如果学习率过小，会导致模型收敛速度慢，需要更多的迭代次数才能达到最优解。

因此，在训练神经网络模型时，需要不断地调整学习率，以保证模型的训练效果和速度。我尝试使用了一些学习率调整策略，例如学习率衰减、动态调整学习率等。

```
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                 'max',
                                                 factor=0.2,
                                                patience=12)
```

```
criterion = nn.CrossEntropyLoss()
```

尝试了多次后，发现在 factor=0.2,patience=12 时模型达到较好的效果。

3. MobileNetV1 网络加深

```
self.mobilebone = nn.Sequential(
            self._conv_bn(3, 32, 2),
            self._conv_dw(32, 64, 1),
            self._conv_dw(64, 128, 2),
            self._conv_dw(128, 128, 1),
            self._conv_dw(128, 256, 2),
          )
```

五、总结

模型达到了预期目标，得到较好的检测效果。

可能改进的方向：

1. 使用更深的卷积神经网络模型，例如 ResNet、DenseNet 等，以提升模型的检测能力。
2. 调整模型的超参数，例如 Batch Size、正则化系数等，以优化模型的训练效果。
3. 使用数据增强技术，例如随机裁剪、旋转、翻转等，扩充训练数据集，以提升模型的泛化能力。
4. 尝试使用目标检测算法，例如 YOLO、SSD 等，以提升模型的检测速度和精度。

实现过程中遇到的困难：

1. 数据集的质量和数量不足，需要进行数据增强和数据清洗等操作。
2. 训练过程中模型出现过拟合或者欠拟合问题，需要进行超参数调整和模型结构优化等操作。
3. 模型的训练速度较慢，需要进行硬件优化或者使用分布式训练等操作。

提升性能的方面：

1. 调整模型的超参数
2. 卷积神经网络模型的深度加深
3. 使用数据增强技术，例如随机裁剪、旋转、翻转等，扩充训练数据集，以提升模型的泛化能力。