口罩佩戴检测

姓名：胡天扬

学号：3190105708

专业：自动化（控制）

课程：人工智能与机器学习

指导教师：张建明 徐巍华

题目分析

本题针对图像识别，要求对是否佩戴口罩进行二分类，基础知识是 `OpenCV` 的应用和卷积神经网络 `CNN` 的理解。算法主要由两部分组成，首先用 `MTCNN` 进行人脸检测，然后用 `MobileNet` 进行口罩识别，即先定位目标，后进行分类。

数据处理

对图像的处理主要在类型转换上，另外也用了 `torchvision.transforms` 类，比如 `transforms.ToTensor()` 将 `PILImage` 转变为 `torch.FloatTensor` 的数据类型。

同时也将数据按 `9:1` 划分了训练集和验证集。

```
    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
```

MTCNN

$MTCNN$ 采用了三个级联的网络，通过候选框加分类器的思想，进行快速高效的人脸检测。这三个级联的网络分别是快速生成候选窗口的 $P-Net$、进行高精度候选窗口过滤选择的 $R-Net$ 和生成最终边界框与人脸关键点的 $O-Net$。和很多处理图像问题的卷积神经网络模型一样，该模型也用到了图像金字塔、边框回归、非最大值抑制等技术。

题目已经有了训练好的 PRO 权重，可以修改 `detector.py` 中的阈值，但是效果不大。

```
        thresholds=[0.6, 0.7, 0.8],
        nms_thresholds=[0.7, 0.7, 0.7],
```

MobileNet

$MobileNet$ 也已经实现好了，可以自行决定卷积块 `_conv_dw` 的个数和其中的层数。$MobileNet$ 的优势在于精简的计算量和参数量，这是 $GoogleNet$ 和 VGG 不可比拟的。

可以适当扩大 $MobileNet$ 的规模，让 `dw` 卷积块有更多的神经元，并让其内部有两个 `conv+BN+ReLu` 的组合。

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

        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),)
```

scheduler

`optim.lr_scheduler.ReduceLROnPlateau` 用于调整学习率，可以将损失函数或准确率等指标作为评估对象，当该指标在一定容忍度 `patience` 后仍然不增大或减小，则按 `factor` 作为因子降低学习率。

```
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,'max', factor=0.5,
 patience=4)
```

例如将验证集的准确率作为评估对象，在每个 `epoch` 训练后加入验证部分：

```
    model.eval()
    total_eval_accuracy = 0
    for batch_idx, (x, y) in tqdm(enumerate(valid_data_loader, 1)):
        with torch.no_grad():
            pred_y = model(x)
            total_eval_accuracy += accuracy(pred_y, y)
    val_acc = total_eval_accuracy / len(valid_data_loader)
    scheduler.step(val_acc)
```

但是实际应用中效果不大，因为优化器 `Adam` 足够智能了。。

超参数

调参的数据两页都放不下，截取一下部分调参的数据。

score	factor	patience	epoch	batch	lr
85	0.5	4	30	32	1e-3
90	0.5	4	20	32	1e-3
70	0.5	3	20	32	1e-3
80	0.5	2	20	32	1e-3
74	0.5	5	20	32	1e-3
70	0.5	6	20	32	1e-3
70	0.5	10	20	32	1e-3
70	0.5	4	20	32	1e-2
86/70	0.5	4	20	32	1e-1
87.5/46	0.5	4	30	32	1e-1
74/46	0.5	3	30	32	1e-1
46/46	0.5	3	40	32	1e-1
70/46	0.3	3	20	32	1e-1
69/74	0.7	3	20	32	1e-1
46	0.5	6	20	32	1e-1
70	0.5	6	50	32	1e-1
89	0.9	4	50	32	1e-1
87.5/90	0.5	4	50	32	1e-4
90	0.5	4	100	32	1e-4
95	0.5	4	50	32	5e-3

以上只是很小一部分调参的内容，最终的效果可以稳定在95左右。

其他尝试

Keras 框架

$Pytorch$ 准确率卡在95左右，所以也尝试了 $Keras$ 框架，因为两者有不同的优化策略 `scheduler`，在 $Keras$ 框架中尝试了 `early stop` 早停的功能，从而防止过拟合。并且将一定迭代次数后的最优模型进行保存，以此作为初始化权重继续训练。但是同样只能做到90分左右。

```
early_stopping = EarlyStopping( monitor='val_loss',   # 检测的指标
                                min_delta=0,          # 增大或减小的阈值
                                patience=10,          # 检测的轮数频率
                                verbose=1 )           # 信息展示的模式
```

VGG 模型

感觉 $MobileNet$ 毕竟是轻量化的网络，存在上限，所以换了个 $VGG16$ 进行尝试。

```
class VggNet(nn.Module):
    def __init__(self):
        super(VggNet, self).__init__()
        super().__init__()
        self.model = models.vgg16()
        self.model.classifier._modules['6'] = nn.Linear(4096, 2)
        self.model.load_state_dict(torch.load("./weights/vgg.pth"))

    def forward(self, x):
        return self.model(x)
```

虽然 VGG 的参数模型大的离谱，大约半个G，但是效果很好，几乎没怎么调参就可以做到100%的准确率。

心得

炼丹炼得快吐了呀，调参真的是件很玄学的事情，尤其是对于 $MobileNet$ 这样的轻量级网络来说，有时候不知道究竟是模型鲁棒性不够还是参数没调好，所以会在加深网络层数还是继续调参这二者中不知所措。训练模型把原来的和续费的GPU时间都用完了，最后只能建个小号继续炼丹。由于每次训练我没有固定初始权重，所以同一模型训练两次也会有不同的结果，最后还算是运气比较好，在小号上训练的模型准确率是91.3%，而自己提交时候则是100%的准确率，总算结束了暗无天日的炼丹。