人工智能与机器学习报告

3. 口罩佩戴检测

3220101111 洪晨辉

1. 实验介绍

1.1 实验背景

本次实验力图建立一个目标检测的模型，可以识别图中的人是否佩戴了口罩。此实验在新冠疫情中编写，具有浓厚的时代特色。

1.2 实验要求

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）学习训练时的方法。

1.3 实验环境

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 Pytorch 等深度学习框架训练模型。

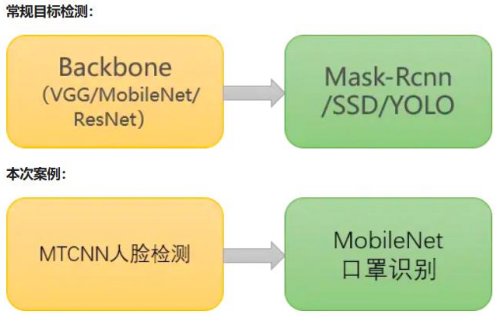
1.4 实验思路

针对目标检测的任务，可以分为两个部分：目标识别和位置检测。

通常情况下，特征提取需要由特有的特征提取神经网络来完成，如 VGG、MobileNet、ResNet 等，这些特征提取网络往往被称为 Backbone 。而在 BackBone 后面接全连接层(FC)就可以执行分类任务。

但 FC 对目标的位置识别乏力。经过算法的发展，当前主要以特定的功能网络来代替 FC 的作用，如 Mask-Rcnn、SSD、YOLO 等。

我们选择充分使用已有的人脸检测的模型，再训练一个识别口罩的模型，从而提高训练的开支、增强模型的准确率。



2. 实验操作

2.1 数据集

主要是一些拥有口罩和没有口罩的人的图片样本。一个典型的正负样本如下所示：

共计约 600 多个样本。

2.2 制作数据集

数据集的预处理主要采用尺寸调整和图像增强的方法。代码如下：

```
# Training data generator
train_data = ImageDataGenerator (
rescale =1. / 255 , shear_range =0.1 ,
zoom_range =0.1 , width_shift_range =0.1 ,
height_shift_range =0.1 , horizontal_flip
=
True , vertical_flip = True ,
validation_split = test_split
)
# Testing data generator
test_data = ImageDataGenerator ( rescale
=1. /
255 , validation_split = test_split )
```

2.3 模型选用

MTCNN 仍然相同。但令人伤心的是，MobileNetV1 的识别效果较差。因此，本报告选用的是微软更新后的 MobileNetV2。值得注意的是，该模型也有一定历史了——其提出时间为 2018 年。

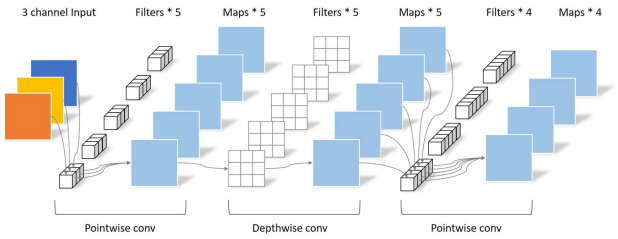
该模型仍然遵循经典的步骤，先进行模型预处理（已经存在 ProcessingData.py 中，未改），再进行模型训练。模型部分（存在 MobileNetV2.py 中）如下：

```python
import torch
import torch.nn as nn
import torchvision


def Conv3x3BNReLU(in_channels, out_channels, stride, groups):
    return nn.Sequential(
        # stride=2 wh 减半，stride=1 wh 不变
        nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                  kernel_size=3, stride=stride, padding=1, groups=groups),
        nn.BatchNorm2d(out_channels),
        nn.ReLU6(inplace=True)
    )

# PW 卷积


def Conv1x1BNReLU(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                  kernel_size=1, stride=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU6(inplace=True)
    )

# # PW 卷积(Linear) 没有使用激活函数


def Conv1x1BN(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                  kernel_size=1, stride=1),
        nn.BatchNorm2d(out_channels)
    )


class InvertedResidual(nn.Module):
    # t = expansion_factor,也就是扩展因子，文章中取6
    def __init__(self, in_channels, out_channels, expansion_factor, stride):
        super(InvertedResidual, self).__init__()
        self.stride = stride
        self.in_channels = in_channels
        self.out_channels = out_channels
        mid_channels = (in_channels * expansion_factor)
        # print("expansion_factor:", expansion_factor)
        # print("mid_channels:",mid_channels)

        # 先 1x1 卷积升维，再 1x1 卷积降维
        self.bottleneck = nn.Sequential(
            # 升维操作：扩充维度是 in_channels * expansion_factor (6 倍)
            Conv1x1BNReLU(in_channels, mid_channels),
            # DW 卷积,降低参数量
            Conv3x3BNReLU(mid_channels, mid_channels,
                          stride, groups=mid_channels),
            # 降维操作：降维度 in_channels * expansion_factor(6 倍) 降维到指定 out_channels 维度
            Conv1x1BN(mid_channels, out_channels)
        )

        # 第一种：stride=1 才有shortcut 此方法让原本不相同的channels 相同
        if self.stride == 1:
            self.shortcut = Conv1x1BN(in_channels, out_channels)

        # 第二种：stride=1 切 in_channels=out_channels 才有 shortcut
```

```
        # if self.stride == 1 and in_channels == out_channels:
        #     self.shortcut = ()

    def forward(self, x):
        out = self.bottleneck(x)
        # 第一种:
        out = (out + self.shortcut(x)) if self.stride == 1 else out
        # 第二种:
        # out = (out + x) if self.stride == 1 and self.in_channels == self.out_channels else out
        return out


class MobileNetV2(nn.Module):
    # classes 为分类个数, t 为扩充因子
    def __init__(self, classes=2, t=6):
        super(MobileNetV2, self).__init__()

        # 3 -> 32 groups=1 不是组卷积 单纯的卷积操作
        self.first_conv = Conv3x3BNReLU(3, 32, 2, groups=1)

        # 32 -> 16 stride=1 wh 不变
        self.layer1 = self.make_layer(
            in_channels=32, out_channels=16, stride=1, factor=1, block_num=1)
        # 16 -> 24 stride=2 wh 减半
        self.layer2 = self.make_layer(
            in_channels=16, out_channels=24, stride=2, factor=t, block_num=2)
        # 24 -> 32 stride=2 wh 减半
        self.layer3 = self.make_layer(
            in_channels=24, out_channels=32, stride=2, factor=t, block_num=3)
        # 32 -> 64 stride=2 wh 减半
        self.layer4 = self.make_layer(
            in_channels=32, out_channels=64, stride=2, factor=t, block_num=4)
        # 64 -> 96 stride=1 wh 不变
        self.layer5 = self.make_layer(
            in_channels=64, out_channels=96, stride=1, factor=t, block_num=3)
        # 96 -> 160 stride=2 wh 减半
        self.layer6 = self.make_layer(
            in_channels=96, out_channels=160, stride=2, factor=t, block_num=3)
        # 160 -> 320 stride=1 wh 不变
        self.layer7 = self.make_layer(
            in_channels=160, out_channels=320, stride=1, factor=t, block_num=1)
        # 320 -> 1280 单纯的升维操作
        self.last_conv = Conv1x1BNReLU(320, 1280)

        self.avgpool = nn.AvgPool2d(kernel_size=7, stride=1)
        self.dropout = nn.Dropout(p=0.2)
        self.linear = nn.Linear(in_features=1280, out_features=classes)
        self.init_params()

    def make_layer(self, in_channels, out_channels, stride, factor, block_num):
        layers = []
        # 与ResNet 类似，每层Bottleneck 单独处理，指定stride。此层外的stride 均为1
        layers.append(InvertedResidual(
            in_channels, out_channels, factor, stride))
        # 这些叠加层stride 均为1，in_channels = out_channels，其中 block_num-1 为重复次数
        for i in range(1, block_num):
            layers.append(InvertedResidual(
                out_channels, out_channels, factor, 1))
        return nn.Sequential(*layers)

    # 初始化权重操作
    def init_params(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
```

```
                nn.init.kaiming_normal_(m.weight)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear) or isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

    def forward(self, x):

        x = self.first_conv(x)  # torch.Size([1, 32, 112, 112])
        x = self.layer1(x)      # torch.Size([1, 16, 112, 112])
        x = self.layer2(x)      # torch.Size([1, 24, 56, 56])
        x = self.layer3(x)      # torch.Size([1, 32, 28, 28])
        x = self.layer4(x)      # torch.Size([1, 64, 14, 14])
        x = self.layer5(x)      # torch.Size([1, 96, 14, 14])
        x = self.layer6(x)      # torch.Size([1, 160, 7, 7])
        x = self.layer7(x)      # torch.Size([1, 320, 7, 7])
        x = self.last_conv(x)   # torch.Size([1, 1280, 7, 7])
        x = self.avgpool(x)     # torch.Size([1, 1280, 1, 1])
        x = x.view(x.size(0), -1)    # torch.Size([1, 1280])
        x = self.dropout(x)
        x = self.linear(x)      # torch.Size([1, 5])
        return x
```

其结构图示如下：



训练得到的模型最终交由 FaceRec2.py 进行分类。

3. 结果分析

测试点	状态	时长	结果
在 5 张图片上测试模型	✓	8s	得分:100.0

模型已足够优秀，不必再升级至更新的 MobileNetV3 或是 V4。

4. 总结与讨论

当然，如果足够自信，本例还可以采用 MobileNetV1。但是相对的，其结果并不是那么理想：

测试点	状态	时长	结果
在 5 张图片上测试模型	✓	21s	得分:88.33