程序报告

姓名：秦嘉俊　学号：3210106182

一、问题重述

1.1 背景

今年一场席卷全球的新型冠状病毒给人们带来了沉重的生命财产的损失。有效防御这种传染病毒的方法就是积极佩戴口罩。我国对此也采取了严肃的措施，在公共场合要求人们必须佩戴口罩。在本次实验中，我们要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。

1.2 任务

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。

二、设计思想

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。

2.1 MTCNN

MTCNN（Multi-Task Cascaded Convolutional Networks）是 2016 年提出的，如其名，算法采用了级联 CNN 的结构实现了多任务学习——人脸检测和人脸对齐，最终能够预测出人脸框（bounding box）和关键点（landmarks）的位置。MTCNN 在 FDDB、WIDER FACE 和 AFLW 数据集上均取得了当时（2016 年 4 月）最好的效果，而且速度也快（在当时来说），被广泛用于人脸识别流程中的前端部分。

总地来说，MTCNN 的效果主要得益于以下几点：

1. 级联的 CNN 架构是一个 coarse-to-fine（由粗到细）的过程；
2. 在训练过程中使用了在线困难样本挖掘（OHEM，Online Hard Example Mining）的策略；
3. 加入了人脸对齐的联合学习（Joint Face Alignment Learning）

该算法由 3 个阶段组成：

1. 通过 CNN 快速产生候选框体。
2. 通过更复杂一点的 CNN 精炼候选窗体，丢弃大量的重叠窗体。
3. 使用更强大的 CNN，实现候选窗体去留，同时回归 5 个面部关键点。

第一阶段是使用一种叫做 PNet(Proposal Network) 的卷积神经网络，获得候选窗体和边界回归向量。同时，候选窗体根据边界框进行校准。然后利用非极大值抑制去除重叠窗体。第二阶段是使用 R-Net(Refine Network) 卷积神经网络进行操作，将经过 P-Net 确定的包含候选窗体的图片在 R-Net 中训练，最后使用全连接网络进行分类。利用边界框向量微调候选窗体，最后还是利用非极大值抑制算法去除重叠窗体。第三阶段，使用 Onet(Output Network) 卷积神经网络进行操作，该网络比 R-Net 多一层卷积层，功能与 R-Net 类似，只是在去除重叠候选窗口的同时标定 5 个人脸关键点位置。

MTCNN 网络在经过 3 个卷积网络处理之前，先进行了多尺度变换，将一幅人脸图像缩放为不同尺寸的图片，这样就构成了图像金字塔。然后这些不同尺度的图像作为 3 个阶段的输入数据进行训练，这样可以令 MTCNN 检测到不同尺寸的人脸。

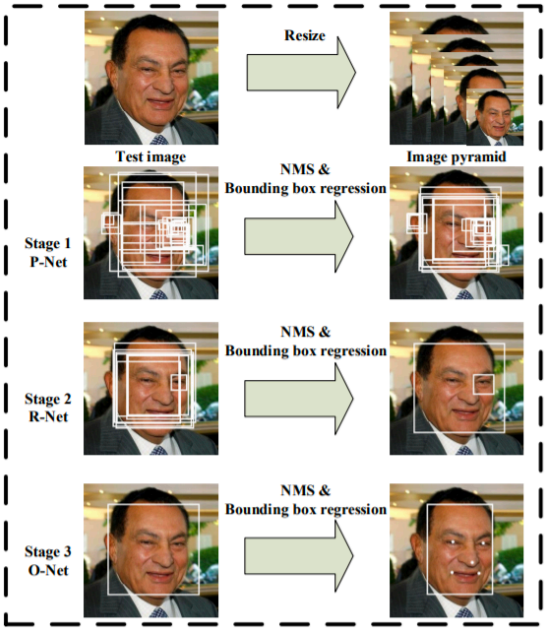

Fig. 1. Pipeline of our cascaded framework that includes three-stage multi-task deep convolutional networks. Firstly, candidate windows are produced through a fast Proposal Network (P-Net). After that, we refine these candidates in the next stage through a Refinement Network (R-Net). In the third stage, The Output Network (O-Net) produces final bounding box and facial landmarks position.

图 1: MCTNN 的过程

2.2 MobileNet

MobileNets 基于流线型架构，使用深度可分离卷积来构建轻量级深度神经网络，用于移动和嵌入式视觉应用。该网络引入了两个简单的全局超参数——宽度乘数和分辨率乘数，可以有效地在延迟和准确性之间进行权衡。这些超参数允许模型构建者根据问题的限制条件为其应用程序选择合适大小的模型。

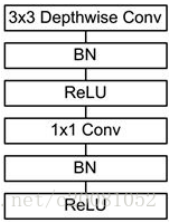

图 2: MobileNet 网络结构

MobileNet 的网络结构如上图所示。首先是一个 3×3 的标准卷积，然后后面就是堆积 depthwise separable convolution，并且可以看到其中的部分 depthwise convolution 会通过

strides=2 进行 down sampling。然后采用 average pooling 将 feature 变成 1x1，根据预测类别大小加上全连接层，最后是一个 softmax 层。

三、代码内容

3.1 使用数据集

```
# 数据集路径
basic_path = "./datasets/5f680a696ec9b83bb0037081-momodel/data/"

def letterbox_image(image, size): # 调整图片尺寸，返回经过调整的照片
    new_image = cv.resize(image, size, interpolation=cv.INTER_AREA)
    return new_image
read_img = cv.imread("test1.jpg")
print("调整前图片的尺寸:", read_img.shape)
read_img = letterbox_image(image=read_img, size=(50, 50))
print("调整前图片的尺寸:", read_img.shape)

def processing_data(data_path, height, width, batch_size=32, test_split=0.1):
    # 数据处理，batch_size默认大小为32

    train_data = ImageDataGenerator(
        # 对图片的每个像素值均乘上这个放缩因子，把像素值放缩到0和1之间有利于模型的收敛
        rescale=1. / 255,
        # 浮点数，剪切强度（逆时针方向的剪切变换角度）
        shear_range=0.1,
        # 随机缩放的幅度，若为浮点数，则相当于[lower,upper] = [1 - zoom_range, 1+
            zoom_range]
        zoom_range=0.1,
        # 浮点数，图片宽度的某个比例，数据提升时图片水平偏移的幅度
        width_shift_range=0.1,
        # 浮点数，图片高度的某个比例，数据提升时图片竖直偏移的幅度
        height_shift_range=0.1,
        # 布尔值，进行随机水平翻转
        horizontal_flip=True,
        # 布尔值，进行随机竖直翻转
        vertical_flip=True,
        # 在 0 和 1 之间浮动。用作验证集的训练数据的比例
        validation_split=test_split
    )

```

```
    # 接下来生成测试集，可以参考训练集的写法
    test_data = ImageDataGenerator(
          rescale=1. / 255,
          validation_split=test_split)

    train_generator = train_data.flow_from_directory(
          # 提供的路径下面需要有子目录
          data_path,
          # 整数元组 (height, width)，默认：(256, 256)。 所有的图像将被调整到的尺寸。
          target_size=(height, width),
          # 一批数据的大小
          batch_size=batch_size,
          # "categorical", "binary", "sparse", "input" 或 None 之一。
          # 默认："categorical",返回one-hot 编码标签。
          class_mode='categorical',
          # 数据子集 ("training" 或 "validation")
          subset='training',
          seed=0)
    test_generator = test_data.flow_from_directory(
          data_path,
          target_size=(height, width),
          batch_size=batch_size,
          class_mode='categorical',
          subset='validation',
          seed=0)

    return train_generator, test_generator
# 数据路径
data_path = basic_path + 'image'

# 图像数据的行数和列数
height, width = 160, 160

# 获取训练数据和验证数据集
train_generator, test_generator = processing_data(data_path, height, width)

# 通过属性class_indices可获得文件夹名与类的序号的对应字典。
labels = train_generator.class_indices
print(labels)

# 转换为类的序号与文件夹名对应的字典
labels = dict((v, k) for k, v in labels.items())
```

```
print(labels)
```

这里我们先导入 Python 第三方库（包）和平台已经给好的 Python 文件，随后从 basic_path 中导入数据集。通过 letterbox_image 调整图片尺寸，并利用 tensorflow 中的工具 ImageDataGenerator 来制作训练时所需要的批量数据集，其具体实现在函数 processing_data 函数中。具体过程可以参见代码及其注释。这一部分的代码 ipynb 中已经提前给出，不需要自行修改。

3.2 预训练模型 MobileNet

```
# 加载 MobileNet 的预训练模型权重
weights_path = basic_path + 'keras_model_data/mobilenet_1_0_224_tf_no_top.h5'
# 图像数据的行数和列数
height, width = 160, 160
model = MobileNet(input_shape=[height,width,3],classes=2)
model.load_weights(weights_path,by_name=True)
print('加载完成...')
```

这里我们加载 MobileNet 的预训练模型权重。这一部分的代码 ipynb 中已经提前给出，不需要自行修改。

3.3 训练并保存模型

```
def save_model(model, checkpoint_save_path, model_dir): # 保存模型

   if os.path.exists(checkpoint_save_path):
      print("模型加载中")
      model.load_weights(checkpoint_save_path)
      print("模型加载完毕")
   checkpoint_period = ModelCheckpoint(
      # 模型存储路径
      model_dir + 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5',
      # 检测的指标
      monitor='val_acc',
      # 'auto'，'min'，'max'中选择
      mode='max',
      # 是否只存储模型权重
      save_weights_only=False,
      # 是否只保存最优的模型
      save_best_only=True,
      # 检测的轮数是每隔2轮
```

```
      period=2
   )
   return checkpoint_period
checkpoint_save_path = "./results/last_one88.h5"
model_dir = "./results/"
checkpoint_period = save_model(model, checkpoint_save_path, model_dir)
# 学习率下降的方式，acc三次不下降就下降学习率继续训练
reduce_lr = ReduceLROnPlateau(
                  monitor='accuracy', # 检测的指标
                  factor=0.5, # 当acc不下降时将学习率下调的比例
                  patience=3, # 检测轮数是每隔三轮
                  verbose=2   # 信息展示模式
               )
early_stopping = EarlyStopping(
                     monitor='val_accuracy', # 检测的指标
                     min_delta=0.0001,   # 增大或减小的阈值
                     patience=5,    # 检测的轮数频率
                     verbose=1       # 信息展示的模式
                  )
# 一次的训练集大小
batch_size = 64
# 图片数据路径
data_path = basic_path + 'image'
# 图片处理
train_generator, test_generator = processing_data(data_path, height=160,
    width=160, batch_size=batch_size, test_split=0.1)
# 编译模型
model.compile(loss='binary_crossentropy', # 二分类损失函数
           optimizer=Adam(lr=0.001), # 优化器
           metrics=['accuracy']) # 优化目标
# 训练模型
history = model.fit(train_generator,
                epochs=10, # epochs: 整数，数据的迭代总轮数。
                # 一个epoch包含的步数,通常应该等于你的数据集的样本数量除以批量大小。
                steps_per_epoch=637 // batch_size,
                validation_data=test_generator,
                validation_steps=70 // batch_size,
                initial_epoch=2, # 整数。开始训练的轮次（有助于恢复之前的训练）。
                callbacks=[checkpoint_period, reduce_lr])
# 保存模型
model.save_weights(model_dir + 'temp.h5')
plt.plot(history.history['loss'],label = 'train_loss')
```

图 3: 调参 1

```
plt.plot(history.history['val_loss'],'r',label = 'val_loss')
plt.legend()
plt.show()

plt.plot(history.history['accuracy'],label = 'acc')
plt.plot(history.history['val_accuracy'],'r',label = 'val_acc')
plt.legend()
plt.show()
```

这里我们先加载和保存模型，这里我们可以通过 ModelCheckpoint 规定在固定迭代次数后保存模型。同时，我们设置在下一次重启训练时，会检查是否有上次训练好的模型，如果有，就先加载已有的模型权重。这样就可以在上次训练的基础上继续模型的训练了。

随后就可以开始准备训练模型，可以通过手动调节学习率，早停法，调节好参数后就可以开始训练模型，展示模型训练过程，并利用模型检测图片中的人数及戴口罩的人.(这里的参数仅用作呈现代码，实际调参中可能与展示代码的参数不同)

具体过程可以参见代码及其注释。这一部分的代码 ipynb 中已经提前给出，基本不需要修改，我们要做的就是修改代码中的参数，以便模型能取得更好的效果。

3.4 调节参数

在我们不调节任何参数的时候（既使用默认参数），我们已经能得到较高的分数: 此时默认的参数为:

```
# 学习率下降的方式，acc三次不下降就下降学习率继续训练
reduce_lr = ReduceLROnPlateau(
                    monitor='acc', # 检测的指标
                    factor=0.5, # 当acc不下降时将学习率下调的比例
                    patience=2, # 检测轮数是每隔两轮
                    verbose=2    # 信息展示模式
                )
```

```
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
                )
                # 一次的训练集大小
batch_size = 8
...
# 训练模型
history = model.fit(train_generator,
              epochs=3, # epochs: 整数，数据的迭代总轮数。
              # 一个epoch包含的步数,通常应该等于你的数据集的样本数量除以批量大小。
              steps_per_epoch=637 // batch_size,
              validation_data=test_generator,
              validation_steps=70 // batch_size,
              initial_epoch=0, # 整数。开始训练的轮次（有助于恢复之前的训练）。
              callbacks=[checkpoint_period, reduce_lr])
...
```

经过反复调节参数后（主要包括 batch_size monitor patience min_delta lr 以及 epochs，我们确定最终使用参数如下

```
# 学习率下降的方式，acc三次不下降就下降学习率继续训练
reduce_lr = ReduceLROnPlateau(
                  monitor='accuracy', # 检测的指标
                  factor=0.5, # 当acc不下降时将学习率下调的比例
                  patience=3, # 检测轮数是每隔三轮
                  verbose=2   # 信息展示模式
              )
early_stopping = EarlyStopping(
                    monitor='val_accuracy', # 检测的指标
                    min_delta=0.0001,   # 增大或减小的阈值
                    patience=3,     # 检测的轮数频率
                    verbose=1       # 信息展示的模式
                )
# 一次的训练集大小
batch_size = 64
...
# 训练模型
history = model.fit(train_generator,
              epochs=10, # epochs: 整数，数据的迭代总轮数。
              # 一个epoch包含的步数,通常应该等于你的数据集的样本数量除以批量大小。
```

```
                steps_per_epoch=637 // batch_size,
                validation_data=test_generator,
                validation_steps=70 // batch_size,
                initial_epoch=0, # 整数。开始训练的轮次（有助于恢复之前的训练）。
                callbacks=[checkpoint_period, reduce_lr])
...
```

四、实验结果

图 4: 最终实验结果

五、总结

总的来说，本次实验主要是通过调整参数来不断提高准确率，几乎不需要自己进行代码的实现。在调参的过程中，每调一次参数都要训练一次，在验证集上要运行一遍整个网络，非常耗时，很难在有限时间内得到非常好的模型，单纯靠“尝试”调参是很难得到一个很好的模型的，需要一些指导性的方针来指引我们朝正确的方向走下去。因此深度学习目前应该发展借助实验和理论发展出一些好的指导原则。

对于算法可能的优化，我认为可以更多地调整参数，选用更多的训练集来使模型达到更好的状态。此外平台提供了三种构建模型的方法，最后我的模型只使用了其中一种，猜测可能可以将几个模型融合在一起，或许能取得不错的效果，具有更强的普适性。

最后，期待下次实验带给我新的惊喜。