第 5 章

两阶段目标检测实战

在 CNN 还未提出之前，针对目标检测这个问题，一般使用“设计受用特征＋分类器”这个思路。当卷积神经网络一系列的模型被提出后，如 ResNet、GoogleNet、AlexNet 等，图像中的特征能够很好地提取出来。因此，目标检测的检测精确度得到极大的提高[19]。到后来推出了一系列的目标检测模型，如 RCNN[20]、Faster RCNN[21]、RPN[22]、MTCNN[23]等，进一步提升了检测精确度。本章主要介绍如何在 TensorFlow2 环境上实现基于 RPN、MTCNN 的两阶段目标检测。

5.1 基于 RPN 实现目标检测

本项目实现了 RPN(Region Proposal Network)的目标检测方法，RPN 区域生成网络作为双阶段目标检测中候选框的提取网络，可以加速检测过程。例如实际应用中，可以将 RPN 网络和 Fast RCNN(与 Faster RCNN 不一样，两种网络)网络结合到一起，将 RPN 获取到的输出框直接连到 ROI 池化层，实现一个 CNN 网络实现端到端目标检测的框架。

本节主要通过 Unity3D 产生虚拟数据，然后用 RPN 进行训练，再在真实数据上进行微调，实现行人检测实战，并利用 RPN 提高检测鲁棒性。

5.1.1 背景原理

RPN 候选框提取网络的输入输出如下。

输入：feature map、物体标签，即训练集中所有物体的类别与边框位置。

输出：Proposal、分类 Loss、回归 Loss。其中，Proposal 作为生成的区域，供后续模块分类与回归。两部分损失用作优化网络。

RPN 主要包含 3 部分：首先，生成 anchor boxes；其次，判断每个 anchor box 为前景(包含物体)或者后景(背景)；最后，二分类边界框回归对 anchor box 进行微调，使得 positive anchor 和真实框(Ground Truth Box)更加接近。

在实现中，默认在每一个点上抽取 9 种 anchor，具体 Scale 为(8,1,32)，Ratio 为(0.5,1,2)，将这 9 种 anchor 的大小反算到原图上，即得到不同的原始 Proposal。然后通过分类网络与回归网络得到每一个 anchor 的前景背景概率和偏移量，回归网络将预测偏移量作用到 anchor 上使得 anchor 更接近于真实物体的真实坐标。参数选择中，在 feature map 上用 3×3 的卷积进行更深的特征提取。

在分类网络分支中，首先使用 1×1 卷积输出 18×37×50 的特征，由于每个点默认有 9 个 anchor，并且对每个 anchor 值预测其属于前景还是背景，因此通道数为 18。随后利用函

数将特征映射到 2×333×75，这样第一维仅仅是一个 anchor 的前景背景得分，并送到 Softmax 函数中进行概率计算，得到的特征再变换到 18×37×50 的维度，最终输出的是每个 anchor 属于前景与背景的概率。

在回归分支中，利用 1×1 卷积输出 36×37×50 的特征，第一位的 36 包含 9 个 anchor 的预测，每一个 anchor 有 4 个数据，分别代表了每一个 anchor 的中心点横、纵坐标及宽、高 4 个量相对于真值的偏移量。RPN 网络结构如图 5-1 所示。

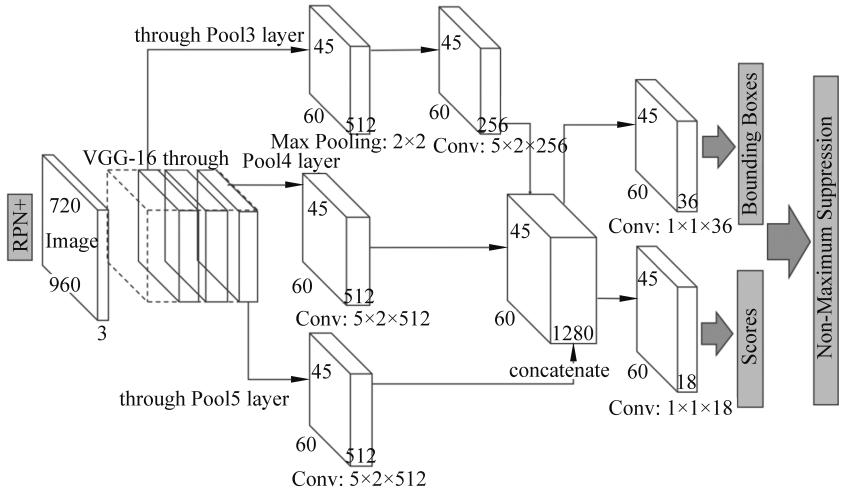


图 5-1　RPN 网络结构[24]

5.1.2　安装操作

本节所用代码文件结构如表 5-1 所示。

表 5-1　代码文件结构

文件名称	实现功能	文件名称	实现功能
kmeans.py	采用 k 均值算法生成 anchor	rpn.py	实现 RPN 模型网络结构
test.py	测试 RPN 模型	train.py	训练 RPN 模型
synthetic_dataset	合成行人数据集	utils.py	实现检测行人的工具函数，包括对 IOU 的计算等

(1) 为了训练 RPN，需要使用合成数据集，它包含一组 8239 幅图像，其中只有一个类别(行人)。

(2) 在训练过程中，如果候选边界框与真实框的交集和重叠度超过 50%，则将其视为正值；如果边界框与真实框的交集和重叠度小于 10%，则将被视为负值。运行一个 demo：python demo.py。

实验中采用 k 均值算法生成 9 个 anchor。可以运行 python kmeans.py 来查看生成过程。

(3) 对网络进行训练，命令为 python train.py。

模型将在每个 epoch 中自动保存权重文件“./RPN.h5”。在神经网络训练期间，可以在

Tensorboard 中访问“http://localhost:6006/”跟踪损失函数曲线。

```
=> epoch 1  step 1  total_loss: 0.402951  score_loss: 0.346327  boxes_loss:
0.056625
=> epoch 1  step 2  total_loss: 0.399650  score_loss: 0.344363  boxes_loss:
0.055287
…
=> epoch 10  step 4000  total_loss: 0.001989  score_loss: 0.000015  boxes_loss:
0.001973
```

(4) 用 200 张图像对模型进行测试，命令为 python test.py，预测结果存储在“./prediction”中。

5.1.3 代码解析

在 kmeans.py 中，使用 k 均值聚类方法求 9 个 anchor。通常，bounding box 由左上角顶点和右下角顶点表示。在对 box 做聚类时，只需要 box 的宽和高作为特征，并且由于数据集中图像的大小可能不同，还需要先使用图像的宽和高对 box 的宽和高做归一化。但如果直接使用标准 k 均值聚类中的欧几里得距离作为度量，则会有个问题，就是在聚类结果中，大 box 簇会比小 box 簇产生更大的误差(squared error)。由于只关心 anchor 与 box 的 IOU，不关心 box 的大小，因此，使用 IOU 作为度量更加合适。当 box 与 anchor 完全重叠，即 IOU=1 时，它们之间的距离为 0。因此，首先计算 box 与 anchor 之间的 IOU。

```
def iou(box, clusters):
    x = np.minimum(clusters[:, 0], box[0])
    y = np.minimum(clusters[:, 1], box[1])
    intersection = x * y
    box_area = box[0] * box[1]
    cluster_area = clusters[:, 0] * clusters[:, 1]
    iou_ = intersection / (box_area + cluster_area - intersection)
return iou_
```

接着，对 box 进行 k 均值聚类，步骤如下。

(1) 随机选取 k 个 box 作为初始 anchor。

(2) 再使用 IOU 度量，将每个 box 分配给与其距离最近的 anchor。

(3) 计算每个簇中所有 box 宽和高的均值，更新 anchor。

重复(2)、(3)步，直到 anchor 不再变化，或者达到了最大迭代次数。

```
def kmeans(boxes, k, dist=np.median,seed=1):
    rows = boxes.shape[0]
    distances = np.empty((rows, k))  ##N 行 x N 列
    last_clusters = np.zeros((rows,))
    np.random.seed(seed)
    #随机选取 k 个 box 作为初始 anchor
    clusters = boxes[np.random.choice(rows, k, replace=False)]
    while True:
```

```
        for icluster in range(k):
            distances[:,icluster] = 1 - iou(clusters[icluster], boxes)
        nearest_clusters = np.argmin(distances, axis=1)
        if (last_clusters == nearest_clusters).all():
            break
        for cluster in range(k):
            clusters[cluster] = dist(boxes[nearest_clusters == cluster], axis=0)
        last_clusters = nearest_clusters
    return clusters
```

在训练 train.py 中，encode_label()完成从 groudtruth 的 box 坐标(x1，y1，x2，y2)到 regression deltas(dx，dy ，dw，dh)的转换。

```
def encode_label(gt_boxes):
    target_scores = np.zeros(shape=[45, 60, 9, 2])   #0: 背景。1: 前景,
    target_bboxes = np.zeros(shape=[45, 60, 9, 4])   #t_x, t_y, t_w, t_h
    target_masks  = np.zeros(shape=[45, 60, 9])      #负样本: -1。正样本: 1
    for i in range(45):                              #y: 高度
        for j in range(60):                          #x: 宽度
            for k in range(9):
                center_x = j * grid_width + grid_width * 0.5
                center_y = i * grid_height + grid_height * 0.5
                xmin = center_x - wandhG[k][0] * 0.5
                ymin = center_y - wandhG[k][1] * 0.5
                xmax = center_x + wandhG[k][0] * 0.5
                ymax = center_y + wandhG[k][1] * 0.5
                #忽略跨边界 anchor
                if (xmin > -5) & (ymin > -5) & (xmax < (image_width+5)) & (ymax <
(image_height+5)):
                    anchor_boxes = np.array([xmin, ymin, xmax, ymax])
                    anchor_boxes = np.expand_dims(anchor_boxes, axis=0)
                    #计算此 anchor 和图像中所有的真实框之间的 IOU
                    ious = compute_iou(anchor_boxes, gt_boxes)
                    positive_masks = ious >= pos_thresh
                    negative_masks = ious <= neg_thresh
                    if np.any(positive_masks):
                        target_scores[i, j, k, 1] = 1.
                        target_masks[i, j, k] = 1      #分类为正样本
                        #找出匹配这个 anchor 的真实框
                        max_iou_idx = np.argmax(ious)
                        selected_gt_boxes = gt_boxes[max_iou_idx]
                        target_bboxes[i, j, k] = compute_regression(selected_gt_
boxes, anchor_boxes[0])
                    if np.all(negative_masks):
                        target_scores[i, j, k, 0] = 1.
                        target_masks[i, j, k] = -1     #分类为负样本
```

```
return target_scores, target_bboxes, target_masks
```

计算损失函数：RPN 训练时要把 RPN classification 和 RPN bounding box regression 的 loss 加到一起来实现联合训练。loss 计算如下。

```
def compute_loss(target_scores, target_bboxes, target_masks, pred_scores, pred_
bboxes):
    score_loss=tf.nn.softmax_cross_entropy_with_logits(labels=target_scores,
  logits=pred_scores)
    foreground_background_mask = (np.abs(target_masks) == 1).astype(np.int)
    score_loss = tf.reduce_sum(score_loss * foreground_background_mask, axis=
[1,2,3])   / np.sum(foreground_background_mask)
    score_loss = tf.reduce_mean(score_loss)
    boxes_loss = tf.abs(target_bboxes - pred_bboxes)
    boxes_loss = 0.5 * tf.pow(boxes_loss, 2) * tf.cast(boxes_loss<1, tf.float32)
+ (boxes_loss - 0.5) * tf.cast(boxes_loss >=1, tf.float32)
    boxes_loss = tf.reduce_sum(boxes_loss, axis=-1)
    foreground_mask = (target_masks > 0).astype(np.float32)
    boxes_loss = tf.reduce_sum(boxes_loss * foreground_mask, axis=[1,2,3]) /
np.sum(foreground_mask)
    boxes_loss = tf.reduce_mean(boxes_loss)
    return score_loss, boxes_loss
```

以下为核心代码 test.py，在测试时，首先对输出进行编码。

```
for i in range(45):
    for j in range(60):
        for k in range(9):
            center_x = j * grid_width + grid_width * 0.5
            center_y = i * grid_height + grid_height * 0.5
            xmin = center_x - wandhG[k][0] * 0.5
            ymin = center_y - wandhG[k][1] * 0.5
            xmax = center_x + wandhG[k][0] * 0.5
            ymax = center_y + wandhG[k][1] * 0.5
            #忽略跨边界的 anchor
            if (xmin > -5) & (ymin > -5) & (xmax < (image_width+5)) & (ymax <
  (image_height+5)):
                anchor_boxes = np.array([xmin, ymin, xmax, ymax])
                anchor_boxes = np.expand_dims(anchor_boxes, axis=0)
                #计算此 anchor 和图像中所有的真实框之间的 IOU
            ious = compute_iou(anchor_boxes, gt_boxes)
                positive_masks = ious > pos_thresh
                negative_masks = ious < neg_thresh
                if np.any(positive_masks):
                    plot_boxes_on_image(encoded_image, anchor_boxes, thickness=1)
                    print("=> Encoding positive sample: %d, %d, %d" %(i, j, k))
```

```
                cv2.circle(encoded_image, center=(int(0.5 * (xmin+xmax)),
                int(0.5 * (ymin+ymax))),radius=1, color=[255,0,0], thickness=4)
                target_scores[i, j, k, 1] = 1.
                target_masks[i, j, k] = 1       #设置标签为负样本
                #找出匹配这个 anchor 的真实框
                max_iou_idx = np.argmax(ious)
                selected_gt_boxes = gt_boxes[max_iou_idx]
                target_bboxes[i, j, k] = compute_regression(selected_gt_boxes,
            anchor_boxes[0])
            if np.all(negative_masks):
                target_scores[i, j, k, 0] = 1.
                target_masks[i, j, k] = -1      #设置标签为负样本
                cv2.circle(encoded_image, center=(int(0.5 * (xmin+xmax)),
            int(0.5 * (ymin+ymax))),radius=1, color=[0,0,0], thickness=4)
        Image.fromarray(encoded_image).show()
```

再对输出进行解码。

```
decode_image = np.copy(raw_image)
pred_boxes = []
pred_score = []
for i in range(45):
    for j in range(60):
        for k in range(9):
            #预测框坐标
            center_x = j * grid_width + 0.5 * grid_width
            center_y = i * grid_height + 0.5 * grid_height
            anchor_xmin = center_x - 0.5 * wandhG[k, 0]
            anchor_ymin = center_y - 0.5 * wandhG[k, 1]
            xmin = target_bboxes[i, j, k, 0] * wandhG[k, 0] + anchor_xmin
            ymin = target_bboxes[i, j, k, 1] * wandhG[k, 1] + anchor_ymin
            xmax = tf.exp(target_bboxes[i, j, k, 2]) * wandhG[k, 0] + xmin
            ymax = tf.exp(target_bboxes[i, j, k, 3]) * wandhG[k, 1] + ymin
            if target_scores[i, j, k, 1] > 0:       #正样本
                print("=> Decoding positive sample: %d, %d, %d" %(i, j, k))
                cv2.circle(decode_image,center=(int(0.5 * (xmin+xmax)), int(0.5
* (ymin+ymax))),
                           radius=1, color=[255,0,0], thickness=4)
                pred_boxes.append(np.array([xmin, ymin, xmax, ymax]))
                pred_score.append(target_scores[i, j, k, 1])
pred_boxes = np.array(pred_boxes)
plot_boxes_on_image(decode_image, pred_boxes, color=[0, 255, 0])
Image.fromarray(np.uint8(decode_image)).show()
```

进行快速编码输出。

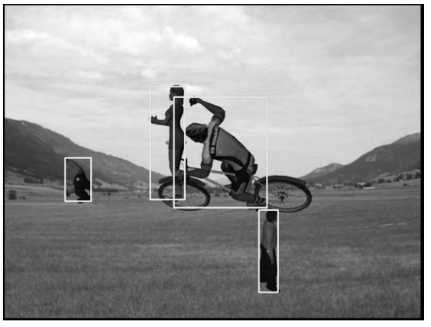
```
faster_decode_image = np.copy(raw_image)
```

```
pred_bboxes = np.expand_dims(target_bboxes, 0).astype(np.float32)
pred_scores = np.expand_dims(target_scores, 0).astype(np.float32)
pred_scores, pred_bboxes = decode_output(pred_bboxes, pred_scores)
plot_boxes_on_image(faster_decode_image, pred_bboxes, color=[255, 0, 0]) Image.
fromarray(np.uint8(faster_decode_image)).show()
```

5.1.4 训练测试

从图 5-2(a)可以看到,RPN 准确地将识别出的人用方框框了出来。从图 5-2(b)可以看到,在目标的中心位置用了一个点来锚定物体的位置。

(a) 用方框框出识别出的人



(b) 用一个点来锚定物体的位置

图 5-2 运行命令与效果展示

5.2 应用 MTCNN 实现人脸目标检测

本次实战将使用多任务卷积神经网络(Multi-Task Convolutional Neural Network,MTCNN)专门针对人脸进行目标检测,并且同时对人面部的 5 个标志点位(左眼、右眼、鼻尖、左右嘴角)进行目标检测。将人脸检测与人脸关键点对齐相结合是 MTCNN 的一大创新点,提高了人脸检测的性能。但同时 MTCNN 也存在着训练需要大量数据、训练时间长以及收敛速度慢等问题。

5.2.1 背景原理

MTCNN 是一个实现人脸识别和面部关键点检测的神经网络,它包括 P-Net、R-Net 和 O-Net 3 层网络结构[25]。

P-Net、R-Net 和 O-Net 是 3 个级联的网络,MTCNN 使用这 3 个网络进行逐层精化,一步步得到更好的预测结果。

P-Net(Proposal Network,提案网络)实现了人脸的区域识别,它通过使用较为浅层、较为简单的卷积神经网络快速生成人脸候选窗口。

R-Net(Refine Network,精化网络)使用了一个更为复杂的卷积神经网络,对 P-Net 初步得到的识别结果进行进一步的选择和调整,舍去大部分的错误输入。之后 R-Net 又再一次进行人脸区域的识别和关键点的定位,最终输出较为可信的识别结果。

O-Net(Output Network,输出网络)与 R-Net 类似,但它比 R-Net 更为复杂。O-Net 使用一个复杂的网络对模型进行优化,并输出最终的预测结果。

总的来说,MTCNN 的整体思想就是先用简单的方法得到一个粗略的结果,然后再用复杂的方法从粗略的结果里找到最终的结果。很显然,这一做法的好处是,使用简单的方法排除了大量的“错误答案”后,再用复杂的方法在大大缩小的范围内寻找“正确答案”,从而大大提升了问题求解的效率。

5.2.2　安装操作

本节所用代码文件结构如表 5-2 所示。

表 5-2　代码文件结构

文 件 名 称	实 现 功 能
mtcnn.py	定义了神经网络的结构
main.py	实现使用现有模型进行人脸识别
utils.py	进行人脸识别需要用到的一些函数,包括检测人脸、生成人脸候选框

本次实战依赖的第三方库有 tensorflow、cv2、numpy、pillow 等。读者首先要在 Python 运行环境中安装好这些库,再在 Python 环境中运行 python main.py 对目标图像进行人脸检测。

5.2.3　代码解析

本次实战的核心代码包括 mtcnn.py、main.py 和 utils.py 3 个文件。在此主要针对 mtcnn.py 进行介绍,mtcnn.py 中实现了 P-Net、R-Net 和 O-Net 3 个网络的定义。

从代码中可以看出 P-Net 的网络结构是非常简单的,仅仅只有 3 层(此处没有计算用于得到人脸候选框以及得分的 2 个卷积层)。P-Net 依次经过 3 层神经网络,网络之间用使用 PReLU 激活函数,在第一层和第二层之间还有一个池化层。

```
class PNet(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(10, 3, 1, name='conv1')
        self.prelu1 = tf.keras.layers.PReLU(shared_axes=[1,2], name="PReLU1")
        self.conv2 = tf.keras.layers.Conv2D(16, 3, 1, name='conv2')
        self.prelu2 = tf.keras.layers.PReLU(shared_axes=[1,2], name="PReLU2")
        self.conv3 = tf.keras.layers.Conv2D(32, 3, 1, name='conv3')
        self.prelu3 = tf.keras.layers.PReLU(shared_axes=[1,2], name="PReLU3")
        self.conv4_1 = tf.keras.layers.Conv2D(2, 1, 1, name='conv4-1')
        self.conv4_2 = tf.keras.layers.Conv2D(4, 1, 1, name='conv4-2')
    def call(self, x, training=False):
        out = self.prelu1(self.conv1(x))
        out = tf.nn.max_pool2d(out, 2, 2, padding="SAME")
```

```
        out = self.prelu2(self.conv2(out))
        out = self.prelu3(self.conv3(out))
        score = tf.nn.softmax(self.conv4_1(out), axis=-1)
        boxes = self.conv4_2(out)
        return boxes, score
```

R-Net 相比 P-Net 更为复杂，无论是网络层数（R-Net 有 4 层，而 P-Net 只有 3 层）还是每一层卷积网络的大小以及网络间的池化层和 flatten 层，R-Net 都比 P-Net 更为复杂。R-Net 处理来自 P-Net 的粗略结果，最后输出更为精确的预测结果交由 O-Net 处理。

```
class RNet(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(28, 3, 1, name='conv1')
        self.prelu1 = tf.keras.layers.PReLU(shared_axes=[1,2], name="prelu1")
        self.conv2 = tf.keras.layers.Conv2D(48, 3, 1, name='conv2')
        self.prelu2 = tf.keras.layers.PReLU(shared_axes=[1,2], name="prelu2")
        self.conv3 = tf.keras.layers.Conv2D(64, 2, 1, name='conv3')
        self.prelu3 = tf.keras.layers.PReLU(shared_axes=[1,2], name="prelu3")
        self.dense4 = tf.keras.layers.Dense(128, name='conv4')
        self.prelu4 = tf.keras.layers.PReLU(shared_axes=None, name="prelu4")
        self.dense5_1 = tf.keras.layers.Dense(2, name="conv5-1")
        self.dense5_2 = tf.keras.layers.Dense(4, name="conv5-2")
        self.flatten = tf.keras.layers.Flatten()
    def call(self, x, training=False):
        out = self.prelu1(self.conv1(x))
        out = tf.nn.max_pool2d(out, 3, 2, padding="SAME")
        out = self.prelu2(self.conv2(out))
        out = tf.nn.max_pool2d(out, 3, 2, padding="VALID")
        out = self.prelu3(self.conv3(out))
        out = self.flatten(out)
        out = self.prelu4(self.dense4(out))
        score = tf.nn.softmax(self.dense5_1(out), -1)
        boxes = self.dense5_2(out)
        return boxes, score
```

O-Net 比 R-Net 更为复杂，它经过 5 层卷积神经网络，对来自 R-Net 的多个预测结果进行处理后得到最终的输出结果。

```
class ONet(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(32, 3, 1, name="conv1")
        self.prelu1 = tf.keras.layers.PReLU(shared_axes=[1,2], name="prelu1")
        self.conv2 = tf.keras.layers.Conv2D(64, 3, 1, name="conv2")
```

```
        self.prelu2 = tf.keras.layers.PReLU(shared_axes=[1,2], name="prelu2")
        self.conv3 = tf.keras.layers.Conv2D(64, 3, 1, name="conv3")
        self.prelu3 = tf.keras.layers.PReLU(shared_axes=[1,2], name="prelu3")
        self.conv4 = tf.keras.layers.Conv2D(128, 2, 1, name="conv4")
        self.prelu4 = tf.keras.layers.PReLU(shared_axes=[1,2], name="prelu4")
        self.dense5 = tf.keras.layers.Dense(256, name="conv5")
        self.prelu5 = tf.keras.layers.PReLU(shared_axes=None, name="prelu5")
        self.dense6_1 = tf.keras.layers.Dense(2  , name="conv6-1")
        self.dense6_2 = tf.keras.layers.Dense(4  , name="conv6-2")
        self.dense6_3 = tf.keras.layers.Dense(10 , name="conv6-3")
        self.flatten = tf.keras.layers.Flatten()
    def call(self, x, training=False):
        out = self.prelu1(self.conv1(x))
        out = tf.nn.max_pool2d(out, 3, 2, padding="SAME")
        out = self.prelu2(self.conv2(out))
        out = tf.nn.max_pool2d(out, 3, 2, padding="VALID")
        out = self.prelu3(self.conv3(out))
        out = tf.nn.max_pool2d(out, 2, 2, padding="SAME")
        out = self.prelu4(self.conv4(out))
        out = self.dense5(self.flatten(out))
        out = self.prelu5(out)
        score = tf.nn.softmax(self.dense6_1(out))
        boxes = self.dense6_2(out)
        lamks = self.dense6_3(out)
        return boxes, lamks, score
```

main.py 通过读取已有的网络模型和调用 utils.py 中实现的人脸检测方法对输入的图像进行人脸识别和特征点定位，在此不再赘述。

需要注意的是，本次实战的代码中并没有实现网络的训练过程，有兴趣的读者可以尝试自行实现。

5.2.4　训练测试

项目安装完成之后即可在命令行中输入 python main.py 对目标图像进行人脸检测。如果需要更换图像，可以在 main.py 中“image = cv2.cvtColor(cv2.imread("./multiface.jpg"), cv2.COLOR_BGR2RGB)”这行代码里将"./multiface.jpg"改为自己想要的图像的路径，如果想直接在命令行中指定要识别的图像路径或者对多张图像进行批量处理，可以尝试修改 main.py 文件自行实现。

输入 python main.py，代码运行的结果如图 5-3 所示，可以看到，MTCNN 可以准确地识别出一张图像中的各张人脸以及每张人脸上的关键点定位。

模型准确识别出了图像中的 6 张人脸，每张人脸以方框框出，同时在每张人脸上对 5 个关键点进行了定位，以点标出，分别是左眼、右眼、鼻尖、左嘴角和右嘴角。