《计算机视觉》实验报告

姓名：严昕宇 学号：20121802

实验 9

一. 任务 1

a) 核心代码：

实验9 运动目标检测

采用基于背景建模的方法实现运动目标检测，建模方法自选
算法步骤：
(1) 读取视频（逐帧读取）
(2) 累加权重重构背景（中值/均值建模），可选其他建模方法
(3) 计算像素差，即每一帧图像与背景模板的差值
(4) 图像去噪：中值滤波、均值滤波、形态学变换（腐蚀、膨胀）
(5) 画出候选框，可用cv2.findContours()函数
(6) 通过非极大值抑制筛选去除多余候选框
(7) 输出检测结果

以下部分为算法步骤分析，其中的代码不完整，完整代码请见末尾
1. 读取视频

[ ]:
```
import cv2

FilePath="./test.avi"
cap = cv2.VideoCapture(FilePath) # 创建摄像头识别类
if not cap.isOpened():
    # 如果没有检测到视频/摄像头输入，报错
    raise Exception('Please check the video file!')
while cap.isOpened():
    ret, frame = cap.read()  # 读取每一帧图片，返回的frame就是某一时刻视频的图片
    cv2.imshow("Video", frame) # 在window上显示图片
    key = cv2.waitKey(10)
    if not ret:
        break
    if key & 0xFF == ord('q'):
        # 按q退出
        break
```

```
# 释放摄像头
cap.release()
cv2.destroyAllWindows()
```

2. 累加权重重构背景

通过权重累加前面的每一帧图片，重构视频背景。delta参数就是前面累加背景的权重。

[ ]:
```
delta = 0.9
cv2.accumulateWeighted(gray, background, delta)
```

3. 计算像素差

计算视频中这一帧图片与重构背景图像素的绝对值差。 其中黑色区域是由于像素相同得0，灰色区域越接近白色，表示两帧图片该点的像素差越大。

[ ]:
```
# background 是上一帧图片的灰度图
diff = cv2.absdiff(gray, background)
```

再将帧差图进行二值化，即图片像素差大于某一阈值threshold的标注为运动点，赋值为255（白色），其余点赋值为0（黑色）。

[ ]:
```
_, mask = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)
```

4. 图像去噪

此时虽然得到了帧差图的二值化图，但图中有许多干扰。这时候我们就需要通过一定的方法去噪，如中值滤波、均值滤波、形态学变换（腐蚀、膨胀）

4.1 中值滤波

中值滤波是一种图像平滑处理算法，基本原理就是，测试像素周围邻域像素集中的中值代替原像素，能够有效去除孤立的噪声点或较细的噪声线。

[ ]:
```
mask = cv2.medianBlur(mask, 3) # 中值滤波
```

4.2 图像腐蚀和膨胀

腐蚀和膨胀也是图片平滑处理的一种算法，一般先腐蚀再膨胀能够有效去除干扰线。

[ ]:
```
es = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9, 4))
m = cv2.erode(median, es, iterations=2) # 膨胀
m = cv2.dilate(m, es, iterations=2) # 腐蚀
```

5. 画出候选框

此处使用cv2.findContours()函数来查找检测运动目标的轮廓，标注在每一帧图片上

[ ]:
```
# 寻找轮廓
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL
                                   , cv2.CHAIN_APPROX_SIMPLE)
# 绘制矩形框
for contour in contours:
    (x, y, w, h) = cv2.boundingRect(contour)
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

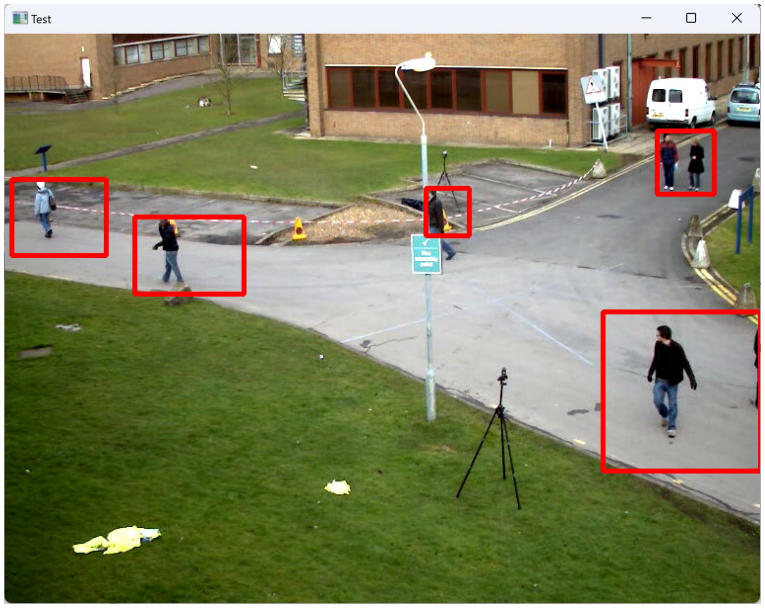
6. 通过非极大值抑制筛选去除多余候选框

非极大值抑制的作用是去除相同目标的重叠的多余的轮廓框，只保留得分最大的
此处将得分定义为运动点占轮廓框总像素点的比例

[ ]:
```
import numpy as np

def py_cpu_nms(dets, thresh):
    y1 = dets[:, 1]
    x1 = dets[:, 0]
    y2 = y1 + dets[:, 3]
    x2 = x1 + dets[:, 2]
    scores = dets[:, 4]  # bbox打分
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    # 打分从大到小排列，取index
    order = scores.argsort()[::-1]
    # keep为最后保留的边框
    keep = []
    while order.size > 0:
        # order[0]是当前分数最大的窗口，肯定保留
        i = order[0]
        keep.append(i)
        # 计算窗口i与其他所有窗口的交叠部分的面积
        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])
        w = np.maximum(0.0, xx2 - xx1 + 1)
        h = np.maximum(0.0, yy2 - yy1 + 1)
        inter = w * h
        # 交/并得到iou值
        ovr = inter / (areas[i] + areas[order[1:]] - inter)
        # inds为所有与窗口i的iou值小于threshold值的窗口的index，其他窗口此次都被窗口i吸收
        inds = np.where(ovr <= thresh)[0]
        # order里面只保留与窗口i交叠面积小于threshold的那些窗口，由于ovr长度比order长度
        # 少(不包含i)，所以inds+1对应到保留的窗口
        order = order[inds + 1]
    return keep
```

b) 实验结果截图



c) 实验小结

背景差分法亦称背景减法，常用于检测视频图像中的运动目标，是目前运动目标检测的主流方法之一。其基本原理就是将图像序列中的当前帧和已经确定好或实时获取的背景参考模型（背景图像）做减法，找不同，计算出与背景图像像素差异超过一定阀值的区域作为运动区域，从而来确定运动物体位置、轮廓、大小等特征，非常适用于摄像机静止的场景。

经过实验，我们可以发现经过背景差分处理再执行二分化操作，就可以更好地找到所需的运动目标。当然只做这些处理还不够，仍存在较多噪声，所以还需使用如中值滤波、图像腐蚀与膨胀的去噪方法进行优化。

在候选框标定过程中，往往会发现最后输出的边界框数量往往远大于实际数量，而这些模型的输出边界框往往是堆叠在一起的。因此，我们需要 NMS 从堆叠的边框中挑出最好的那个。

在完成上述操作后，还是存在部分错误的标识。这需要更现代的算法，以实现更精准的检测。

附页 A：代码

```
import cv2
import numpy as np

def NMS(dets, thresh):
    y1 = dets[:, 1]
    x1 = dets[:, 0]
    y2 = y1 + dets[:, 3]
    x2 = x1 + dets[:, 2]
    scores = dets[:, 4]  # bbox 打分
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    # 打分从大到小排列，取 index
    order = scores.argsort()[::-1]
    # keep 为最后保留的边框
    keep = []
    while order.size > 0:
        # order[0]是当前分数最大的窗口，肯定保留
        i = order[0]
        keep.append(i)
        # 计算窗口 i 与其他所有窗口的交叠部分的面积
        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])
        w = np.maximum(0.0, xx2 - xx1 + 1)
        h = np.maximum(0.0, yy2 - yy1 + 1)
        inter = w * h
        # 交/并得到 iou 值
        ovr = inter / (areas[i] + areas[order[1:]] - inter)
        # inds 为所有与窗口 i 的 iou 值小于 threshold 值的窗口的 index，其他窗口此次都被窗口 i 吸收
        inds = np.where(ovr <= thresh)[0]
        # order 里面只保留与窗口 i 交叠面积小于 threshold 的那些窗口，由于 ovr 长度比 order 长度少 1(不包含 i)，所以 inds+1 对应到保留的窗口
        order = order[inds + 1]
    return keep


class Detector(object):
    def __init__(self, name='Test', FrameNum=10, KSize=7, color=(0, 0, 255)):
        self.name = name
        self.color = color
        self.nms_threshold = 0.3
```

```
        self.es = cv2.getStructuringElement(
            cv2.MORPH_ELLIPSE, (KSize, KSize))


    def VideoProcess(self, VideoIndex=0, KSize=7, delta=0.95,
                     iterations=3, threshold=20, BiasNum=1,
                     MinArea=360, TestShow=True, nms=True):
        if not BiasNum > 0:
            raise Exception('BiasNum must > 0')
        # 判断输入的是摄像头还是视频
        if isinstance(VideoIndex, str):
            self.is_camera = False
            # 如果是视频，则需要调整帧率
        else:
            self.is_camera = True
        cap = cv2.VideoCapture(VideoIndex)  # 创建摄像头识别类
        # 初始化存储视频
        videoname = "test_result.avi";
        # 设置视频解析方式
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        # 设置图片写入方式，注意画面格式为（宽，长）
        writer = cv2.VideoWriter(videoname, fourcc, 10.0, (768, 576), True)
        if not cap.isOpened():
            # 如果没有检测到视频/摄像头输入，报错
            raise Exception('Please check the video file or camera!')
        self.FrameNum = 0
        while cap.isOpened():
            mask, frame = self.WeightsBK(
                cap, KSize, delta, iterations, threshold,
                BiasNum, MinArea, TestShow, nms)
            cnts, _ = cv2.findContours(
                mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

            bounds = self.NmsCnt(cnts, mask, MinArea, nms=nms)
            for b in bounds:
                x, y, w, h = b
                thickness = (w * h) // MinArea
                thickness = thickness if thickness <= 3 else 3
                thickness = thickness if thickness >= 1 else 1
                cv2.rectangle(frame, (x, y), (x + w, y + h), self.color, thickness)
            cv2.imshow(self.name, frame)  # 展示每一帧图片
            writer.write(frame)  # 输出每一帧图片至视频文件

            if TestShow:
```

```
            cv2.imshow(self.name + ' Frame', mask)  # 边界

        key = cv2.waitKey(10)
        if key & 0xFF == ord('q'):
            # 按q退出
            break

    writer.release()  # 释放视频输出工具
    cap.release()  # 释放摄像头
    cv2.destroyAllWindows()  # 释放窗口

def WeightsBK(self, cap, KSize=7, delta=0.95,
              iterations=3, threshold=20, BiasNum=1,
              MinArea=360, TestShow=True, nms=True):
    ret, frame = cap.read()  # 读取每一帧图片
    if not ret:
        if self.is_camera:
            raise Exception('Unexpected Error.')
    if self.FrameNum < BiasNum:
        value = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        self.background = value
        self.FrameNum += 1
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    raw = gray.copy()
    gray = np.abs(gray - self.background).astype('uint8')

    self.background = self.background * delta + raw * (1 - delta)
    self.background = self.background.astype('uint8')
    gray = cv2.medianBlur(gray, KSize)
    _, mask = cv2.threshold(
        gray, threshold, 255, cv2.THRESH_BINARY)
    mask = cv2.medianBlur(mask, KSize)
    mask = cv2.dilate(mask, self.es, iterations)
    mask = cv2.erode(mask, self.es, iterations)
    return mask, frame

def NmsCnt(self, cnts, mask, MinArea, nms=True):
    # 对检测到的边界框使用非极大值抑制
    bounds = [cv2.boundingRect(
        c) for c in cnts if cv2.contourArea(c) > MinArea]
    if len(bounds) == 0:
        return []
    if not nms:
        return bounds
```

```
        scores = [self.calculate(b, mask) for b in bounds]
        bounds = np.array(bounds)
        scores = np.expand_dims(np.array(scores), axis=-1)
        keep = NMS(np.hstack([bounds, scores]), self.nms_threshold)
        return bounds[keep]

    def calculate(self, bound, mask):
        x, y, w, h = bound
        area = mask[y:y + h, x:x + w]
        pos = area > 0
        pos = pos.astype(float)
        # 得分应与检测框大小也有关系
        score = np.sum(pos) / (w * h)
        return score


if __name__ == "__main__":
    FilePath = "./test.avi"
    detector = Detector(name='Test')
    detector.VideoProcess(FilePath, BiasNum=2, iterations=2, KSize=5,
TestShow=True, MinArea=360, nms=True,
                          threshold=30, delta=0.9)
```