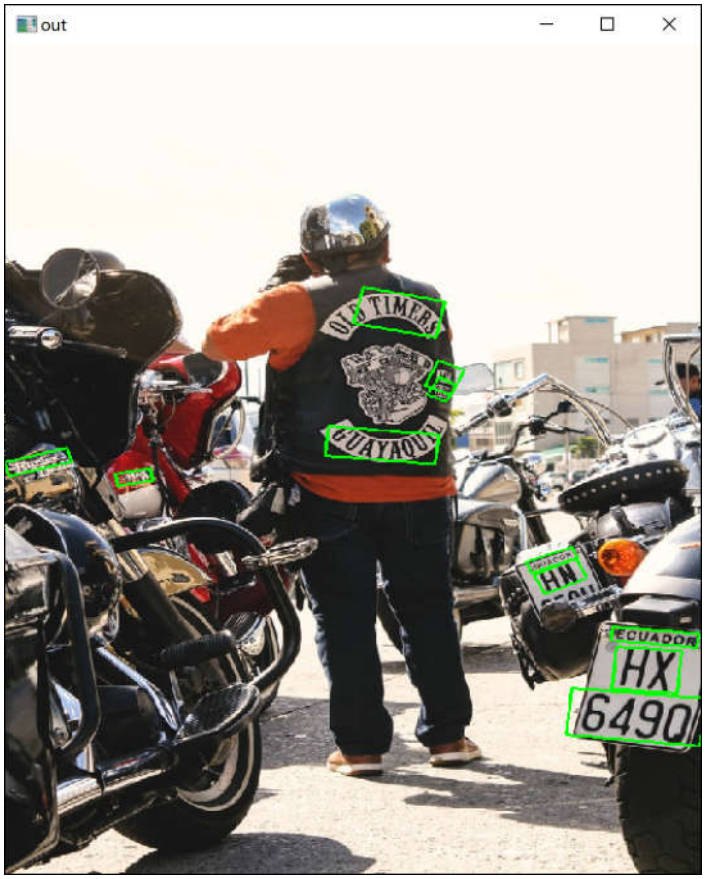
Python编程与人工智能实践



应用篇：自然场景文字检测
Text Detection (EAST)

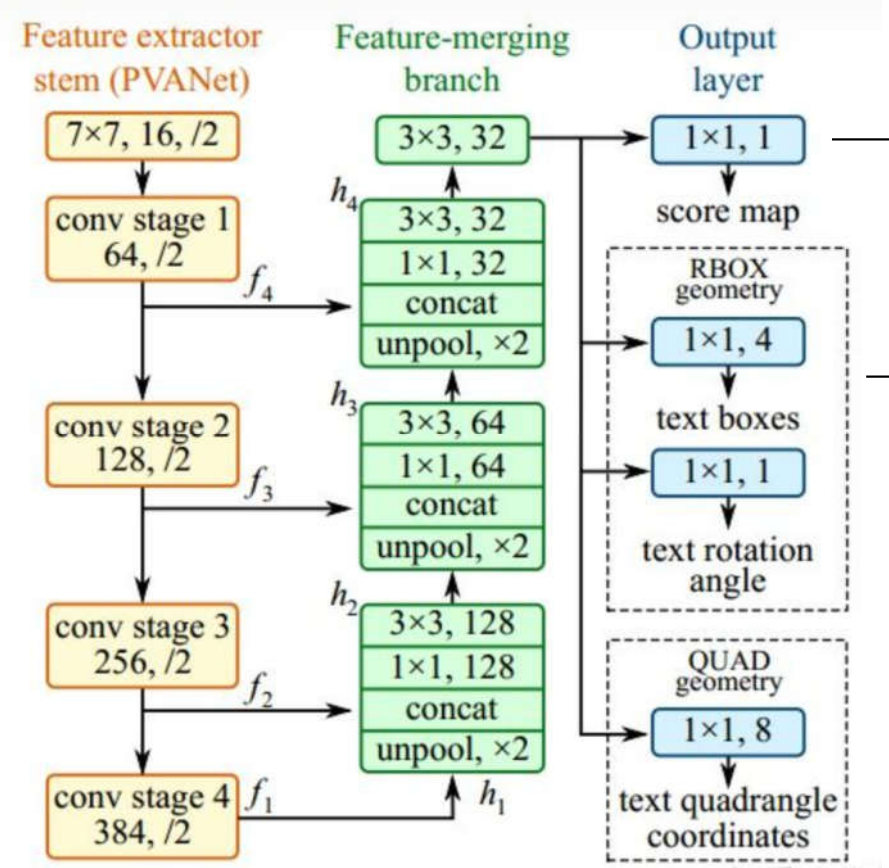
于泓
鲁东大学
信息与电气工程学院
2022.2.12

自然场景下的文字检测

文字检测是很多计算机视觉任务的前置步骤，比如文字识别，身份认证等，因此，文字的精准定位既十分重要又具备挑战。

EAST：预测 收缩的文字区域，并对区域内的**每个像素**预测它到文字区域边界，上下左右的四个距离和一个旋转角度，Pipeline 十分简单，同时速度较快。

W，H

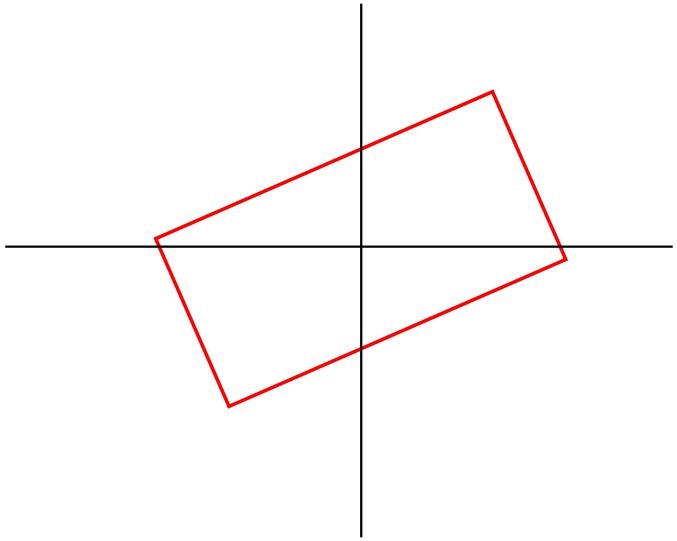


分数

W/4,H/4,1

W/4,H/4,5

几何形状



中心点，到上、右、下、左四个边的偏移量
旋转角度（逆时针）

提取过程
（1） 加载模型
（2）调整图像大小 **W、H**可以被**32**整除
（3）将图像输入网络，计算每个像素的几何信息
（4）根据几何信息获取文字框的位置
（5）采用非最大抑制获取最终的文本框

代码实现：

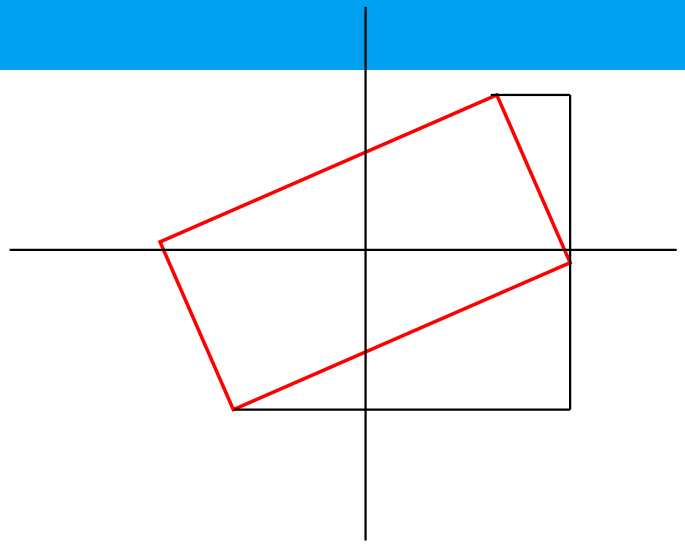
```
import cv2
import numpy as np
import time
if __name__ == "__main__":

    confThreshold = 0.5
    nmsThreshold = 0.3

    # 读取图像
    image = cv2.imread("bikes.jpg")
    orig = image.copy()

    # 获取图像大小
    H,W = image.shape[:2]

    # 定义输入神经网络图像大小
    # newW = 320
    # newH = 320
    newW = int(W/32)*32
    newH = int(H/32)*32

    # 计算缩放比例
    rW = W/float(newW)
    rH = H/float(newH)

    # 载入模型
    file_east = "frozen_east_text_detection.pb"
    net = cv2.dnn.readNet(file_east)
```

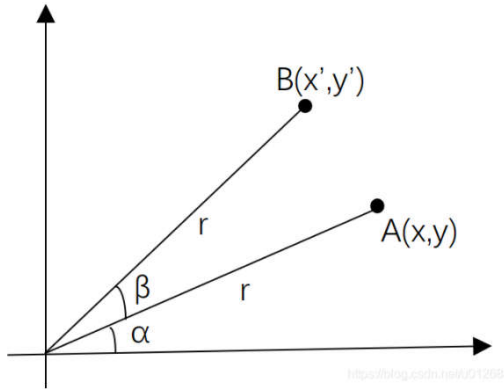
设置输出层

```
# 定义模型输出端口
outputLayers = []
layerNames = ["feature_fusion/Conv_7/Sigmoid","feature_fusion/concat_3"]

# 将输入图像转换为适合神经网络模型的格式
blob = cv2.dnn.blobFromImage(image, 1.0, (newW, newH),(123.68, 116.78, 103.94), swapRB=True, crop=False)

# 送入模型进行处理
net.setInput(blob)

# 得到密集采样的结果
# scores :  1,1,h,w
# geometry: 1,5,h,w
(scores, geometry) = net.forward(layerNames)

# 获取密集采样的结果
(numRows, numCols) = scores.shape[2:4]

 # 存储检测区域的list
 detections = []

 # 存储检测区域的置信度
 confidences = []
```

对输入图像进行处理

```
for y in range(numRows):
    for x in range(numCols):
        score = scores[0,0,y,x]
        # 置信度过低，不处理
        if(score < confThreshold):
            continue
        # print(y,x,score)
        # 获取区域信息，
        up = geometry[0,0,y,x]   # 上 偏移
        right = geometry[0,1,y,x]   # 右 偏移
        down = geometry[0,2,y,x]   # 下 偏移
        left = geometry[0,3,y,x]   # 左 偏移
        angle = geometry[0,4,y,x] # 角度逆时针

        cosA = np.cos(angle)
        sinA = np.sin(angle)

        # 计算区域的宽和高
        h = up + down
        w = left + right

        # 计算中心点偏移量 × 4
        (offsetX, offsetY) = (x * 4.0, y * 4.0)

        # 计算右下角坐标
        pt_right_down = (offsetX + cosA * right + sinA * down, offsetY - sinA * right + cosA * down)

        # 计算算右上角坐标
        pt_right_up = (-sinA * h + pt_right_down[0], -cosA * h + pt_right_down[1])

        # 计算左下坐标
        pt_left_down = (-cosA * w + pt_right_down[0],  sinA * w + pt_right_down[1])
```





x'=rcos(α+β) = r(cosαcosβ-sinαsinβ) = xcosβ-ysinβ

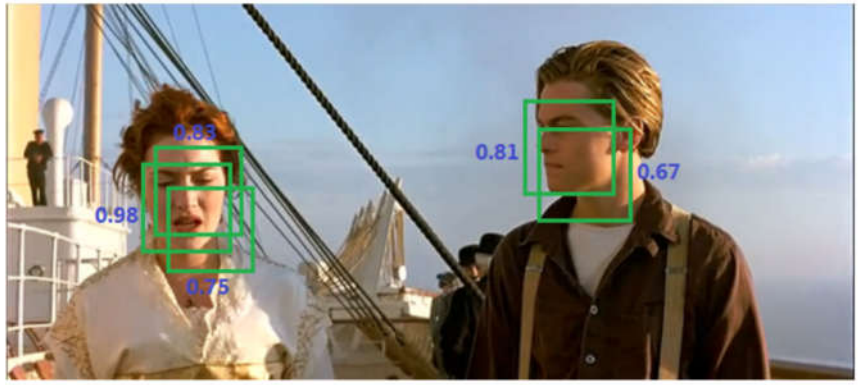
y'=rsin(α+β) = r(sinαcosβ+cosαsinβ) = xsinβ+ycosβ

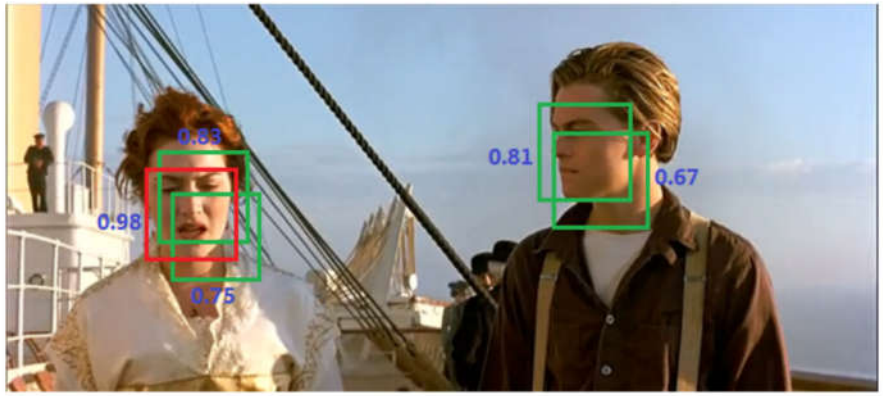
```
# 根据 右上 左下计算中心点坐标
center = (0.5*(pt_right_up[0]+pt_left_down[0]), 0.5*(pt_right_up[1]+pt_left_down[1]))

# 信息存储 满足boxPoints输入的格式
# （中心坐标），（宽，高），顺时针角度
detections.append((center, (w,h), -1*angle * 180.0 / np.pi))

confidences.append(score)
```

计算中心点的坐标

```
# 进行非最大抑制
# 重叠面面积>nmsThreshold的框去除
indices = cv2.dnn.NMSBoxesRotated(detections, confidences, confThreshold,nmsThreshold)
print(indices)
# 绘图
for i in indices:

    # 得到四个顶点
    vertices = cv2.boxPoints(detections[i])

    # 进行坐标缩放
    vertices = np.array(vertices) * np.array([[rW,rH]])
    vertices = vertices.astype("int")

    # 画矩形
    cv2.polylines(image, [vertices], True, (0, 255, 0), 2)

# 结果显示
cv2.namedWindow("out", cv2.WINDOW_NORMAL or cv2.WINDOW_KEEPRATIO or cv2.WINDOW_GUI_NORMAL )
cv2.imshow("out",image)

cv2.waitKey(0)
```

非最大抑制





找到分数最大区域（红色），将和红框重叠超过某个阈值的框，都去除

再找下一个分数最大的框，重复上述动作