OpenCV实战 | EAST算法实现自然场景文本检测

Adrian Rosebrock CVer 8月24日

来源：pyimagesearch

作者：Adrian Rosebrock

编译：三石、大明

前言

今天要介绍的内容是**利用EAST算法检测自然场景下的文本。**EAST算法是CVPR 2017的paper，很早就开源了，如今移植到OpenCV中，实在太cool了。OpenCV3.4.2包含了很多最新的算法，比如前不久CVer就推送了一篇**利用OpenCV玩转YOLOv3**。

《EAST: An Efficient and Accurate Scene Text Detector》
arXiv：https://arxiv.org/abs/1704.03155
github：https://github.com/argman/EAST

运行环境：

EAST文本检测器需要OpenCV3.4.2或更高版本，有需要的读者可以先安装OpenCV。

主要内容：

- 教程第一部分分析为何在自然场景下进行文本检测的挑战性是如此之高。
- 接下来简要探讨EAST文本检测器，为何使用，算法新在何处，并附上相关论文供读者参考。
- 最后提供 Python + OpenCV文本检测实现方式，供读者在自己的应用中使用。

为何在自然场景下进行文本检测的挑战性是如此之高

由于光照条件、图片质量以及目标非线性排列等因素的限制，自然场景下的文本检测任务难度较大

受约束的受控环境中的文本检测任务通常可以使用基于启发式的方法来完成，比如利用梯度信息或文本通常被分成段落呈现，并且字符一般都是成直线排列等信息。

但自然场景下文本检测则不同，而且更具挑战性。

由于廉价数码相机和智能手机的普及，我们需要高度关注图像拍摄时的条件。Celine Mancas-Thillou和Bernard Gosselin在其2017年发表的优秀论文《自然场景文本理解》中描述了的自然场景文本检测面对的主要挑战：

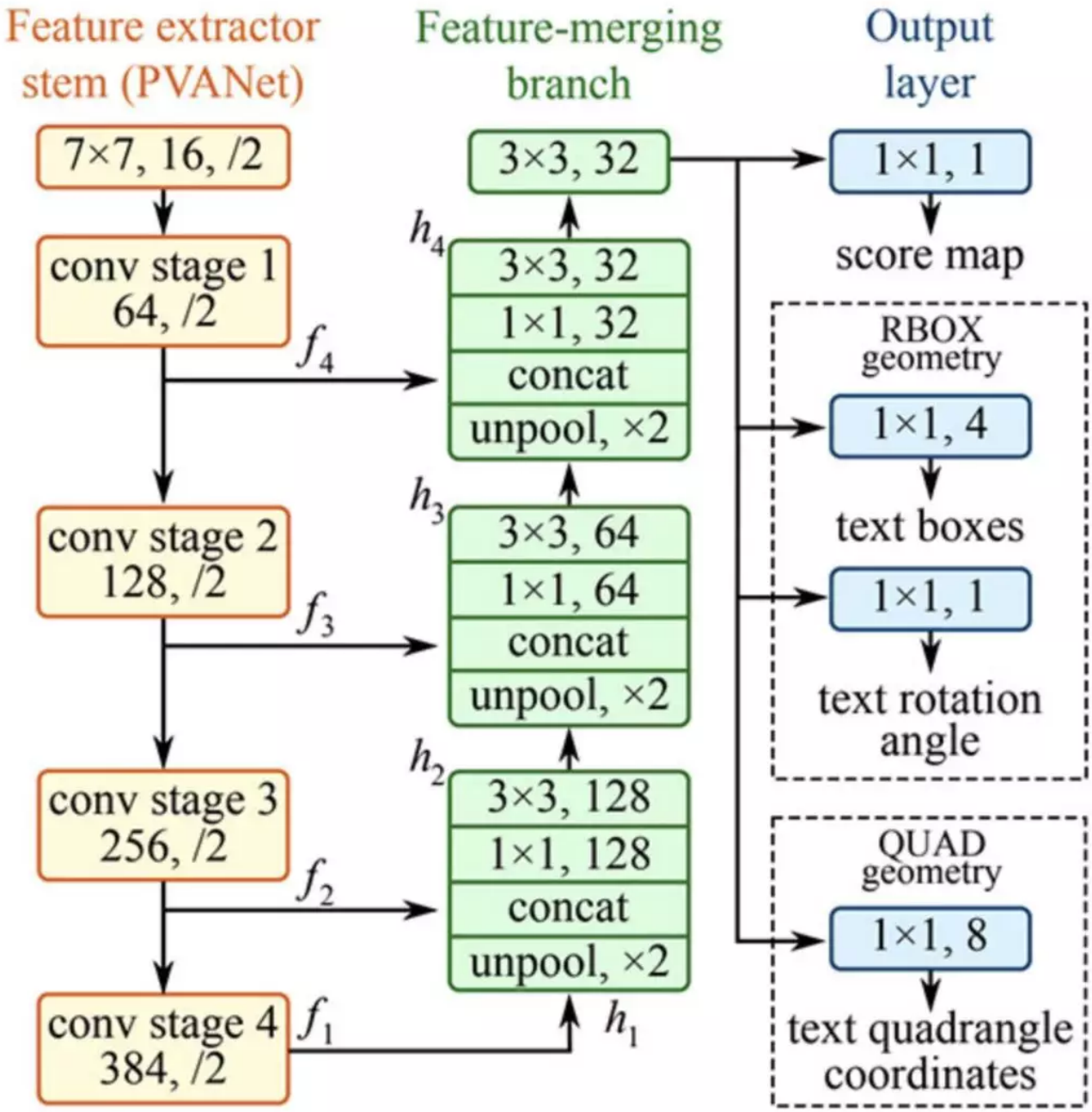
- **图像/传感器噪音**：手持式相机的传感器噪音通常要高于传统扫描仪。此外，廉价相机通常会介入原始传感器的像素以产生真实的颜色。
- **视角**：自然场景中的文本存在不平行的观测角度问题，使文本更难以识别。
- **模糊**：不受控制的环境下，文本往往会变模糊，尤其是如果最终用户使用的智能手机的拍摄稳定性不足时，问题就更明显。
- **照明条件**：我们无法对自然场景图像中的照明条件做出任何假设。可能在接近黑暗的条件下，相机上的闪光灯可能会亮起，也可能在艳阳高照的条件下，使整个图像趋于饱和。
- **分辨率**：每台图像捕捉设备都是不同的，可能存在分辨率过低的摄像机拍出的图像。
- **非纸质对象**：大多数（但不是全部）纸张是不反光的。而自然场景中的文字可能是反光的，比如徽标，标志等。
- **非平面目标**：想象文字印在瓶子上的情况，瓶子表面上的文本会扭曲和变形。虽然我们自己仍可以轻松地“检测”并阅读文本，但算法做起来就会很困难。我们需要能够处理这种情况的用例。
- **处理条件未知**：我们不能使用任何先验信息来为算法提供关于文本所在位置的“线索”。

OpenCV'sEAST文本检测器甚至可以识别模糊图片中的文字

CVer

Welcome to click AD

EAST深度学习文本检测器



EAST文本检测器全卷积网络结构

EAST是一种基于深度学习的文本探测器，即高效、准确的场景文本检测（Efficient and Accurate Scene Text detectionpipeline）。更重要的是，深度学习模型是端对端的，因此可能绕开一般文本识别器用的计算成本高昂的子算法，比如候选对象聚合和词汇分割等。

项目结构

首先使用Tree终端命令来浏览项目结构：

```
$ tree --dirsfirst
.
├── images
```

```
4   │   ├── car_wash.png
5   │   ├── lebron_james.jpg
6   │   └── sign.jpg
7   ├── frozen_east_text_detection.pb
8   ├── text_detection.py
9   └── text_detection_video.py
10
11  1 directory, 6 files
```

在images/ 目录下已有三张样图，读者可以自己添加更多图片。

我们使用两个.py 文件:

- text_detection.py：检测静态图像中的文本
- text_detection_video.py：检测网络摄像头或输入图像文件中的文本

两个脚本都使用EAST模型（frozen_east_text_detection.pb）

注意事项

本文中介绍的实例基于OpenCV的官方C++实例，在转换为Python的过程中可能会遇见一些问题。

比如，Python中没有Point2f 和 RotatedRect函数，所以不能完全再现C++环境下的实现。

其次，NMSBoxes函数不返回Python绑定的任何值，最终导致OpenCV报错。 NMSBoxes函数可以在OpenCV3.4.2中使用，但我无法对其进行详尽的测试。

使用OpenCV实现文本检测器的构建

在开始之前，我想再次指出，您至少需要在系统上安装OpenCV 3.4.2（或OpenCV 4）才能使用OpenCV的EAST文本检测器，因此如果您还没有安装OpenCV 3.4.2或更高版本，请参阅后文的OpenCV安装指南。

接下来，安装或升级你的系统中的imutils 。

```
1   $ pip install --upgrade imutils
```

此时，系统设置已经完成，打开 text_detection.py ，输入以下代码:

```
# import the necessary packages
from imutils.object_detection import non_max_suppression
import numpy as np
import argparse
import time
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str,
    help="path to input image")
ap.add_argument("-east", "--east", type=str,
    help="path to input EAST text detector")
ap.add_argument("-c", "--min-confidence", type=float, default=0.5,
    help="minimum probability required to inspect a region")
ap.add_argument("-w", "--width", type=int, default=320,
    help="resized image width (should be multiple of 32)")
ap.add_argument("-e", "--height", type=int, default=320,
    help="resized image height (should be multiple of 32)")
args = vars(ap.parse_args())
```

首先，我们在第2-6行导入所需的包和模块。注意，我们从imutils.object_detection导入NumPy，OpenCV和non_max_suppression实现。

然后我们继续解析第9-20行的五个命令行参数：

- --image：输入图像的路径。
- --east：EAST场景文本检测器模型文件路径。
- --min-confidence：确定文本的概率阈值。可选，默认值= 0.5。
- --width：调整后的图像宽度 - 必须是32的倍数。可选，默认值= 320。
- --height：调整后的图像高度 - 必须是32的倍数。可选，默认值= 320。

重要提示：EAST文本要求输入图像尺寸为32的倍数，因此如果您选择调整图像的宽度和高度值，请确保这两个值是32的倍数！

然后加载图像并调整大小：

```
# load the input image and grab the image dimensions
image = cv2.imread(args["image"])
orig = image.copy()
(H, W) = image.shape[:2]

# set the new width and height and then determine the ratio in change
# for both the width and height
(newW, newH) = (args["width"], args["height"])
rW = W / float(newW)
rH = H / float(newH)
```

```
32
33  # resize the image and grab the new image dimensions
34  image = cv2.resize(image, (newW, newH))
35  (H, W) = image.shape[:2]
```

第23和24行加载并复制输入图像。

第30行和第31行确定原始图像尺寸与新图像尺寸的比率（基于为--width和--height提供的命令行参数）。

然后我们调整图像大小，忽略纵横比（第34行）。

为了使用OpenCV和EAST深度学习模型执行文本检测，我们需要提取两层的输出特征映射：

```
37  # define the two output layer names for the EAST detector model that
38  # we are interested -- the first is the output probabilities and the
39  # second can be used to derive the bounding box coordinates of text
40  layerNames = [
41      "feature_fusion/Conv_7/Sigmoid",
42      "feature_fusion/concat_3"]
```

我们在40-42行构建了layerNames的表：

- 第一层是我们的输出sigmoid激活，它给出了包含文本或不包含文本的区域的概率。
- 第二层是表示图像“几何”的输出要素图。我们使用它来导出输入图像中文本的边界框坐标。

加载OpenCV的EAST文本检测器：

```
44  # load the pre-trained EAST text detector
45  print("[INFO] loading EAST text detector...")
46  net = cv2.dnn.readNet(args["east"])
47
48  # construct a blob from the image and then perform a forward pass of
49  # the model to obtain the two output layer sets
50  blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
51      (123.68, 116.78, 103.94), swapRB=True, crop=False)
52  start = time.time()
53  net.setInput(blob)
54  (scores, geometry) = net.forward(layerNames)
55  end = time.time()
56
57  # show timing information on text prediction
58  print("[INFO] text detection took {:.6f} seconds".format(end - start))
```

我们使用cv2.dnn.readNet将神经网络加载到内存中，方法是将路径传递给EAST检测器作为第46行的参数。

然后我们通过将其转换为第50行和第51行的blob来准备我们的图像。要了解有关此步骤的更多信息，请参阅深度学习：OpenCV的blobFromImage如何工作。

要预测文本，我们可以简单地将blob设置为输入并调用net.forward（第53和54行）。这些行被抓取时间戳包围，以便我们可以在第58行打印经过的时间。

通过将layerNames作为参数提供给net.forward，我们指示OpenCV返回我们感兴趣的两个特征映射：

- 输出几何图用于导出输入图像中文本的边界框坐标
- 类似地，分数图包含文本的给定区域的概率:

我们需要逐一循环这些值：

```
60  # grab the number of rows and columns from the scores volume, then
61  # initialize our set of bounding box rectangles and corresponding
62  # confidence scores
63  (numRows, numCols) = scores.shape[2:4]
64  rects = []
65  confidences = []
66
67  # loop over the number of rows
68  for y in range(0, numRows):
69      # extract the scores (probabilities), followed by the geometrical
70      # data used to derive potential bounding box coordinates that
71      # surround text
72      scoresData = scores[0, 0, y]
73      xData0 = geometry[0, 0, y]
74      xData1 = geometry[0, 1, y]
75      xData2 = geometry[0, 2, y]
76      xData3 = geometry[0, 3, y]
77      anglesData = geometry[0, 4, y]
```

我们首先抓取score的维度（第63行），然后初始化两个列表：

- rects：存储文本区域的边界框（x，y）坐标
- 置信度：存储与每个边界框相关的概率

我们稍后将对这些区域使用non-maximasuppression。

在第68行开始循环。

第72-77行提取当前行的分数和几何数据y。

接下来，我们遍历当前所选行的每个列索引：

```
 79   # loop over the number of columns
 80       for x in range(0, numCols):
 81           # if our score does not have sufficient probability, ignore it
 82           if scoresData[x] < args["min_confidence"]:
 83               continue
 84
 85           # compute the offset factor as our resulting feature maps will
 86           # be 4x smaller than the input image
 87           (offsetX, offsetY) = (x * 4.0, y * 4.0)
 88
 89           # extract the rotation angle for the prediction and then
 90           # compute the sin and cosine
 91           angle = anglesData[x]
 92           cos = np.cos(angle)
 93           sin = np.sin(angle)
 94
 95           # use the geometry volume to derive the width and height of
 96           # the bounding box
 97           h = xData0[x] + xData2[x]
 98           w = xData1[x] + xData3[x]
 99
100           # compute both the starting and ending (x, y)-coordinates for
101           # the text prediction bounding box
102           endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
103           endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
104           startX = int(endX - w)
105           startY = int(endY - h)
106
107           # add the bounding box coordinates and probability score to
108           # our respective lists
109           rects.append((startX, startY, endX, endY))
110           confidences.append(scoresData[x])
```

对于每一行，我们开始循环第80行的列。

我们需要通过忽略概率不高的区域来过滤弱文本检测(第82行和第83行)。

当图像通过网络时，EAST文本检测器自然地减少了体积大小——我们的体积实际上比输入图像小4倍，所以我们乘以4，使坐标回到原始图像。

我已经包含了如何在第91-93行提取角度数据；然而，正如我在前一节中提到的，不能像在C++中那样构造一个旋转的边界框——如果你想要处理这个任务，那么从第91行角度开始将是你的第一

步。

第97-105行派生出文本区域的边框坐标。

然后我们分别更新rects和confi数据库列表(第109行和第110行)。

最后一步是将非最大值抑制应用于我们的边界框以抑制弱重叠边界框，然后显示结果文本预测：

```
112  # apply non-maxima suppression to suppress weak, overlapping bounding
113  # boxes
114  boxes = non_max_suppression(np.array(rects), probs=confidences)
115
116  # loop over the bounding boxes
117  for (startX, startY, endX, endY) in boxes:
118      # scale the bounding box coordinates based on the respective
119      # ratios
120      startX = int(startX * rW)
121      startY = int(startY * rH)
122      endX = int(endX * rW)
123      endY = int(endY * rH)
124
125      # draw the bounding box on the image
126      cv2.rectangle(orig, (startX, startY), (endX, endY), (0, 255, 0), 2)
127
128  # show the output image
129  cv2.imshow("Text Detection", orig)
130  cv2.waitKey(0)
```

正如我在上一节中提到的，我无法在我的OpenCV 4安装（cv2.dnn.NMSBoxes）中使用非最大值抑制，因为Python绑定没有返回值，最终导致OpenCV出错。我无法完全在OpenCV 3.4.2中进行测试，因此它可以在v3.4.2中运行。

相反，我使用了imutils包中提供的非最大值抑制实现（第114行）。结果仍然很好;但是，我无法将我的输出与NMSBoxes函数进行比较，看它们是否相同。

第117-126行循环遍历边界框，将坐标缩放到原始图像尺寸，并将输出绘制到orig图像。直到按下一个按键为止，原始图像将一直显示（129-130行）。

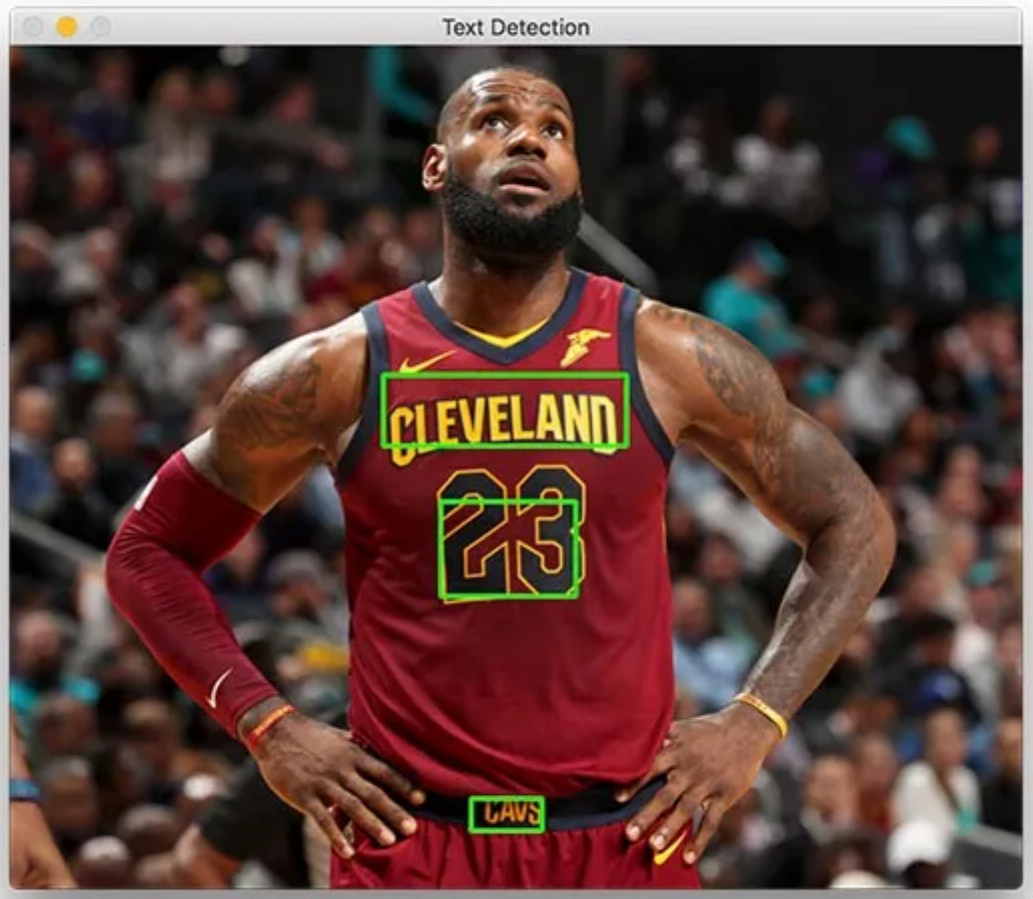
最后一个实验需要注意的是，我们的两个嵌套for循环用于循环第68-110行上的分数和几何体（geometry volume），这是一个很好的例子，说明你可以利用Cython极大地加快pipeline的速度。我已经用OpenCV和Python演示了Cython在快速优化“for”像素循环中的强大功能。

OpenCV文本检测器结果

在终端可以执行一下命令（注意两个命令行参数）：

```
$ python text_detection.py --image images/lebron_james.jpg \
    --east frozen_east_text_detection.pb
[INFO] loading EAST text detector...
[INFO] text detection took 0.142082 seconds
```

结果应该如下图所示：



文本检测器成功识别出篮球巨星勒布朗·詹姆斯球衣上的文字

詹姆斯身上有三个文本区域。

现在让我们尝试检测业务标志的文本：

```
$ python text_detection.py --image images/car_wash.png \
    --east frozen_east_text_detection.pb
[INFO] loading EAST text detector...
[INFO] text detection took 0.142295 seconds
```

使用EAST文本检测器很容易识别出路边洗车店的招牌文字

最后，我们将尝试一个路标:

```
$ python text_detection.py --image images/sign.jpg \
    --east frozen_east_text_detection.pb
[INFO] loading EAST text detector...
[INFO] text detection took 0.141675 seconds
```

基于Python和OpenCV的场景文本检测器和EAST文本检测器成功检测出西班牙语的停车指示路牌

该场景中包含一个西班牙的停车标志。“ALTO”可以准确的被OpenCV和EAST识别出来。

如你所知，EAST非常精确，且相对较快，平均每张图片耗时约0.14秒。

OpenCV在视频中进行文本检测

我们可以基于上述工作，进一步使用OpenCV在视频中进行文本检测。

开启text_detection_video.py，然后插入如下代码：

```
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
from imutils.object_detection import non_max_suppression
import numpy as np
import argparse
import imutils
```

```
import time
import cv2
```

首先，我们导入一些包。我们将使用VideoStream访问网络摄像头并用FPS来为这个脚本测试每秒帧数。其他内容与前一节相同。

为方便起见，定义一个新函数来为我们的预测函数进行解码 - 它将被重用于每个帧并使循环更清晰：

```
def decode_predictions(scores, geometry):
    # grab the number of rows and columns from the scores volume, then
    # initialize our set of bounding box rectangles and corresponding
    # confidence scores
    (numRows, numCols) = scores.shape[2:4]
    rects = []
    confidences = []

    # loop over the number of rows
    for y in range(0, numRows):
        # extract the scores (probabilities), followed by the
        # geometrical data used to derive potential bounding box
        # coordinates that surround text
        scoresData = scores[0, 0, y]
        xData0 = geometry[0, 0, y]
        xData1 = geometry[0, 1, y]
        xData2 = geometry[0, 2, y]
        xData3 = geometry[0, 3, y]
        anglesData = geometry[0, 4, y]

        # loop over the number of columns
        for x in range(0, numCols):
            # if our score does not have sufficient probability,
            # ignore it
            if scoresData[x] < args["min_confidence"]:
                continue

            # compute the offset factor as our resulting feature
            # maps will be 4x smaller than the input image
            (offsetX, offsetY) = (x * 4.0, y * 4.0)

            # extract the rotation angle for the prediction and
            # then compute the sin and cosine
            angle = anglesData[x]
            cos = np.cos(angle)
            sin = np.sin(angle)

            # use the geometry volume to derive the width and height
            # of the bounding box
            h = xData0[x] + xData2[x]
            w = xData1[x] + xData3[x]

            # compute both the starting and ending (x, y)-coordinates
            # for the text prediction bounding box
```

```
55              endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
56              endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
57              startX = int(endX - w)
58              startY = int(endY - h)
59
60              # add the bounding box coordinates and probability score
61              # to our respective lists
62              rects.append((startX, startY, endX, endY))
63              confidences.append(scoresData[x])
64
65      # return a tuple of the bounding boxes and associated confidences
66      return (rects, confidences)
```

在第11行，我们定义decode_prediction函数。该函数用于提取:

- 文本区域的边界框坐标;
- 文本区域检测的概率。

这个专用函数将使代码更易于阅读和管理。

让我们来解析命令行参数:

```
68  # construct the argument parser and parse the arguments
69  ap = argparse.ArgumentParser()
70  ap.add_argument("-east", "--east", type=str, required=True,
71      help="path to input EAST text detector")
72  ap.add_argument("-v", "--video", type=str,
73      help="path to optinal input video file")
74  ap.add_argument("-c", "--min-confidence", type=float, default=0.5,
75      help="minimum probability required to inspect a region")
76  ap.add_argument("-w", "--width", type=int, default=320,
77      help="resized image width (should be multiple of 32)")
78  ap.add_argument("-e", "--height", type=int, default=320,
79      help="resized image height (should be multiple of 32)")
80  args = vars(ap.parse_args())
```

69-80行代码中命令行参数解析:

- --east：EAST场景文本检测器模型文件路径。
- --video：输入视频的路径（可选）。如果提供了视频路径，那么网络摄像头将不会被使用。
- --Min-confidence：确定文本的概率阈值（可选）。default=0.5。
- --width：调整图像宽度(必须是32的倍数，可选)。default=320。
- --Height：调整图像高度(必须是32的倍数，可选)。default=320。

与上一节中仅使用图像的脚本(就命令行参数而言)的不同之处在于，用视频替换了图像参数。

接下里，我们将进行重要的初始化工作：

```
82  # initialize the original frame dimensions, new frame dimensions,
83  # and ratio between the dimensions
84  (W, H) = (None, None)
85  (newW, newH) = (args["width"], args["height"])
86  (rW, rH) = (None, None)
87
88  # define the two output layer names for the EAST detector model that
89  # we are interested -- the first is the output probabilities and the
90  # second can be used to derive the bounding box coordinates of text
91  layerNames = [
92      "feature_fusion/Conv_7/Sigmoid",
93      "feature_fusion/concat_3"]
94
95  # load the pre-trained EAST text detector
96  print("[INFO] loading EAST text detector...")
97  net = cv2.dnn.readNet(args["east"])
```

第84-86行上的高度、宽度和比率初始化将允许我们稍后适当地缩放边界框。

我们定义了输出层的名称，并在第91-97行加载了预先训练好的EAST文本检测器。

下面的代码设置了我们的视频流和每秒帧数计数器：

```
99  # if a video path was not supplied, grab the reference to the web cam
100  if not args.get("video", False):
101      print("[INFO] starting video stream...")
102      vs = VideoStream(src=0).start()
103      time.sleep(1.0)
104
105  # otherwise, grab a reference to the video file
106  else:
107      vs = cv2.VideoCapture(args["video"])
108
109  # start the FPS throughput estimator
110  fps = FPS().start()
```

我们的视频流设置为：

- 一个摄像头（100-103行）
- 或一个视频文件（106-107行）

我们在第110行初始化每秒帧计数器，并开始循环传入帧：

```
112 # loop over frames from the video stream
113 while True:
114     # grab the current frame, then handle if we are using a
115     # VideoStream or VideoCapture object
116     frame = vs.read()
117     frame = frame[1] if args.get("video", False) else frame
118 
119     # check to see if we have reached the end of the stream
120     if frame is None:
121         break
122 
123     # resize the frame, maintaining the aspect ratio
124     frame = imutils.resize(frame, width=1000)
125     orig = frame.copy()
126 
127     # if our frame dimensions are None, we still need to compute the
128     # ratio of old frame dimensions to new frame dimensions
129     if W is None or H is None:
130         (H, W) = frame.shape[:2]
131         rW = W / float(newW)
132         rH = H / float(newH)
133 
134     # resize the frame, this time ignoring aspect ratio
135     frame = cv2.resize(frame, (newW, newH))
```

我们从113行开始在视频/摄像头框架上进行循环。

我们的框架调整了大小，保持了纵横比(第124行)。从129-132行中获取维度并计算比例。然后我们再次调整帧的大小(必须是32的倍数)，这一次忽略了长宽比，因为我们已经存储了用于安全维护（safe keeping）的比率(第135行)。

推理和绘制文本区域边框发生在以下几行:

```
137 # construct a blob from the frame and then perform a forward pass
138     # of the model to obtain the two output layer sets
139     blob = cv2.dnn.blobFromImage(frame, 1.0, (newW, newH),
140         (123.68, 116.78, 103.94), swapRB=True, crop=False)
141     net.setInput(blob)
142     (scores, geometry) = net.forward(layerNames)
143 
144     # decode the predictions, then  apply non-maxima suppression to
145     # suppress weak, overlapping bounding boxes
146     (rects, confidences) = decode_predictions(scores, geometry)
147     boxes = non_max_suppression(np.array(rects), probs=confidences)
148 
149     # loop over the bounding boxes
150     for (startX, startY, endX, endY) in boxes:
151         # scale the bounding box coordinates based on the respective
152         # ratios
153         startX = int(startX * rW)
154         startY = int(startY * rH)
155         endX = int(endX * rW)
```

```
156             endY = int(endY * rH)
157
158             # draw the bounding box on the frame
159             cv2.rectangle(orig, (startX, startY), (endX, endY), (0, 255, 0), 2)
```

在这一代码块中：

创建一个blob并通过网络传递文本区域(第139-142行)；
解码预测并应用NMS(第146行和第147行)。使用之前在这个脚本中定义的decode_forecasts函数和imutils non_max_suppression函数。
循环包围框并在框架上绘制它们(150-159行)。这涉及到按前面收集的比率缩放方框。

而后我们将关闭框架处理循环以及脚本本身：

```
161 # update the FPS counter
162     fps.update()
163
164     # show the output frame
165     cv2.imshow("Text Detection", orig)
166     key = cv2.waitKey(1) & 0xFF
167
168     # if the `q` key was pressed, break from the loop
169     if key == ord("q"):
170         break
171
172 # stop the timer and display FPS information
173 fps.stop()
174 print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
175 print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
176
177 # if we are using a webcam, release the pointer
178 if not args.get("video", False):
179     vs.stop()
180
181 # otherwise, release the file pointer
182 else:
183     vs.release()
184
185 # close all windows
186 cv2.destroyAllWindows()
```

我们在循环的每次迭代中更新fps计数器(第162行)，以便当我们跳出循环时可以计算和显示计时(第173-175行)。

我们在第165行显示了EAST文本检测的输出，并处理按键(第166-170行)。如果“q”键代表“退出”，并被按下，我们将跳出循环，继续清理和释放指针。

视频文本检测结果

要使用OpenCV对视频进行文本检测，请务必点击本文底部“阅读原文”链接获取相应资源。

而后，打开终端并执行以下命令（将会开启摄像头，因为通过命令行参数不提供- -video）：

```
$ python text_detection_video.py --east frozen_east_text_detection.pb
[INFO] loading EAST text detector...
[INFO] starting video stream...
[INFO] elasped time: 59.76
[INFO] approx. FPS: 8.85
```

代码下载及原文链接：
https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/

欢迎给CVer点赞和转发

▲长按关注我们

阅读原文