



小白学视觉
NoobCV

OpenCV实战 项目20讲

1.0V

作者 | 小白
出品 | 小白学视觉

介绍**20**个基于**OpenCV**实现的经典视觉实战项目，以练代学，轻松掌握**OpenCV**使用方法、精通计算机视觉

提供项目实现的源码，拒绝纸上谈兵。本书目前是第一版，书中内容会逐渐更新和完善，后续会更新更多内容。

2020年8月

近期小白学视觉公众号推出了多篇 OpenCV 实战项目的文章，深受小伙伴们的喜爱。最近有小伙伴推荐，希望可以讲经典的项目整理一下，集成手册，便于小伙伴们在日常的学习中使用。于是小白挑选了# OpenCV 的应用#专栏中的 20 篇经典内容，集结成册，便于小伙伴们阅读和学习。

本手册中主要涉及以下几部分，首先是对 OpenCV 中自带的基本函数进行介绍。其次是 OpenCV 的实战项目，一方面是基于实际项目利用 OpenCV 实现特定对象的检测，例如车道线检测、路面的坑洼检测、等；另一方面是基于 OpenCV 实现图像增强，例如利用 OpenCV 消除运动所引起的图像模糊等。最后是 OpenCV 与深度学习等其他相结合实现图像分割、人脸检测、运动检测等难度较大的问题。

因为本手册是处于实时更新和维护的状态，因此会有一些内容变动，为了使小伙伴们获取准确的信息，因此手册中项目的源码不在书中给出。小伙伴们关注“[小白学视觉](#)”微信公众号，回复【**OpenCV 实战项目 20 讲**】就可以获得最新的源码信息。

手册中的具体内容如下：

1. 使用 OpenCV 进行颜色分割
2. 使用 OpenCV 实现图像覆盖
3. 使用 OpenCV 进行图像全景拼接
4. 使用 OpenCV 实现图像修复
5. 自适应显着性的图像分割
6. 使用 OpenCV 实现海岸线变化检测
7. 使用 OpenCV 为视频中美女添加眼线
8. 使用 OpenCV 实现猜词游戏
9. 使用 OpenCV 进行检测坑洼
10. 使用 OpenCV 实现车道线检测
11. 使用 OpenCV 实现道路车辆计数
12. 使用 C# 和 OpenCV 实现人脸替换
13. 使用 OpenCV 进行实时面部检测
14. 使用 OpenCV 实现口罩检测
15. 使用 OpenCV 预处理神经网络中的面部图像
16. 基于 OpenCV 实现深蹲检测器
17. 利用 OpenCV 实现基于深度学习的超分辨率处理
18. 使用 OpenCV 实现社交距离检测器
19. 使用 OpenCV 实现早期火灾检测系统
20. 使用 OpenCV 构建运动检测器

使用OpenCV进行颜色分割

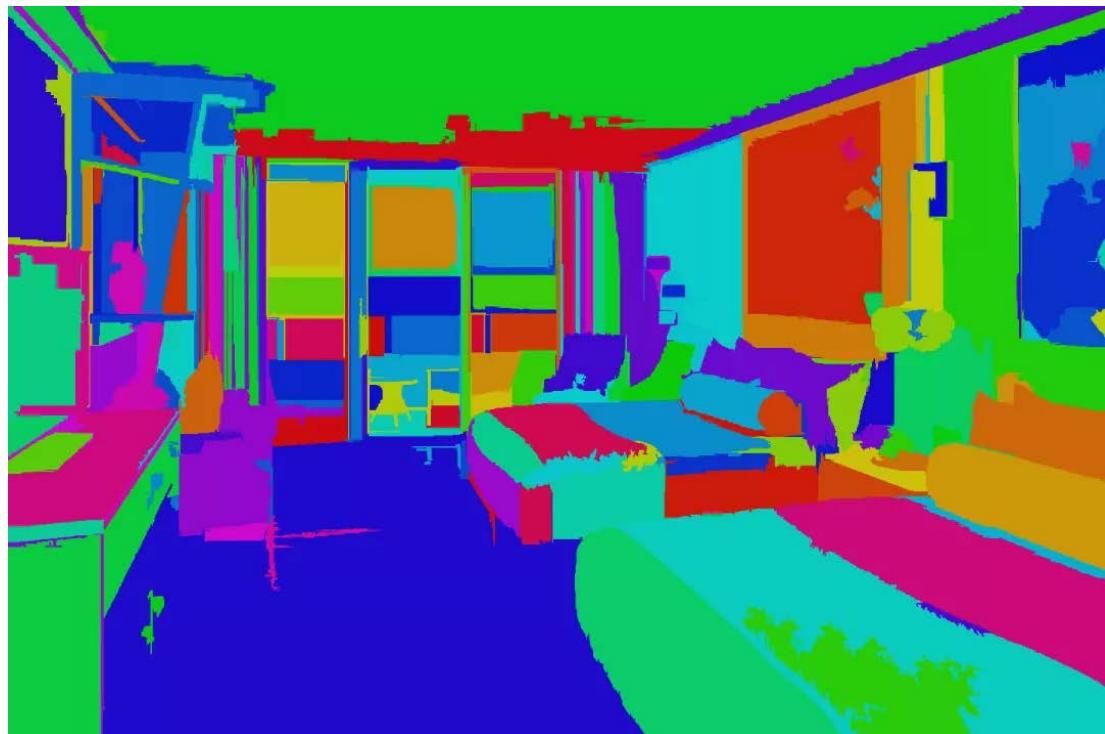
原创 小白 小白学视觉 7月4日

来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达



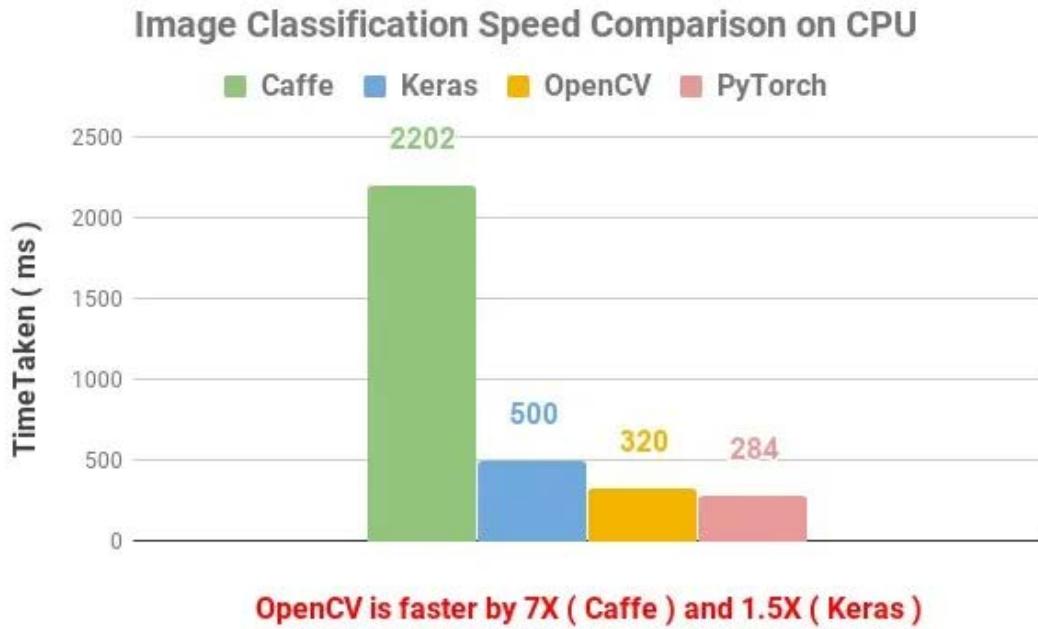
在滤波、变换、缩放等任务中，图像分割具有重要的意义。图像分割是将不同的对象划分为不同的部分，并将这些区域以明显的颜色或者记号标记出来。图像分割是使用轮廓、边界框等概念进行其他高级计算机视觉任务（例如对象分类和对象检测）的基础。良好的图像分割为我们后续的图像分类以及检测奠定了基础。

在计算机视觉中主要有3种不同的图像分割类型：

1. 颜色分割或阈值分割
2. 语义分割
3. 边缘检测

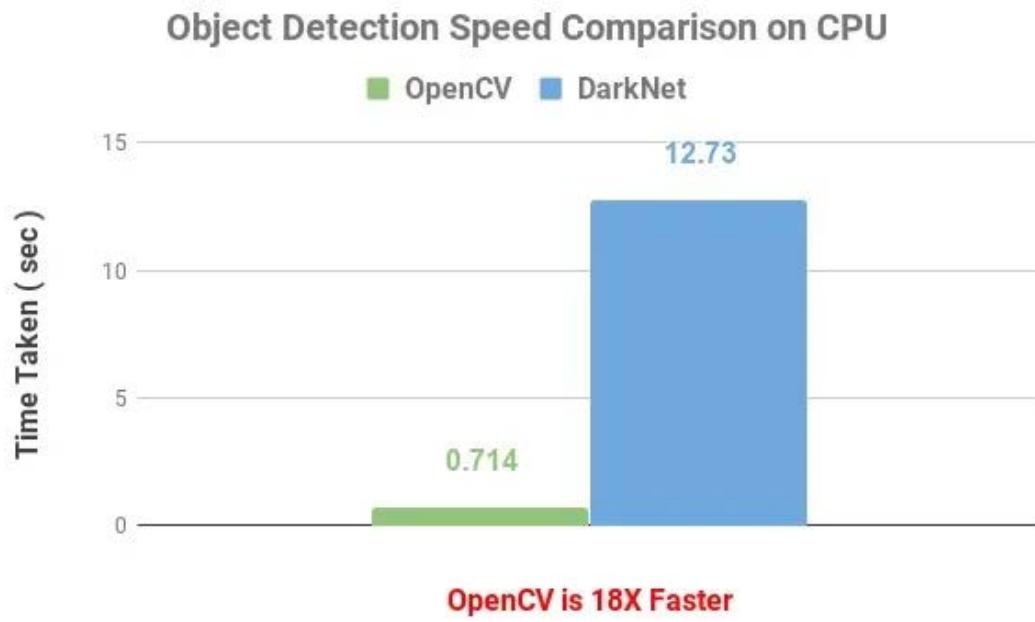
在本文里，我们将介绍基于颜色的图像分割，并通过OpenCV将其实现。小伙伴可能会问，当我们拥有像Caffe和Keras这样的工具时，为什么要使用拥有21年历史的OpenCV库。与Caffe和

Keras等现代SOTA DL方法相比，OpenCV虽然在准确性方面有一些落后，但是运行速度相较于上述方法具有得天独厚的优势。



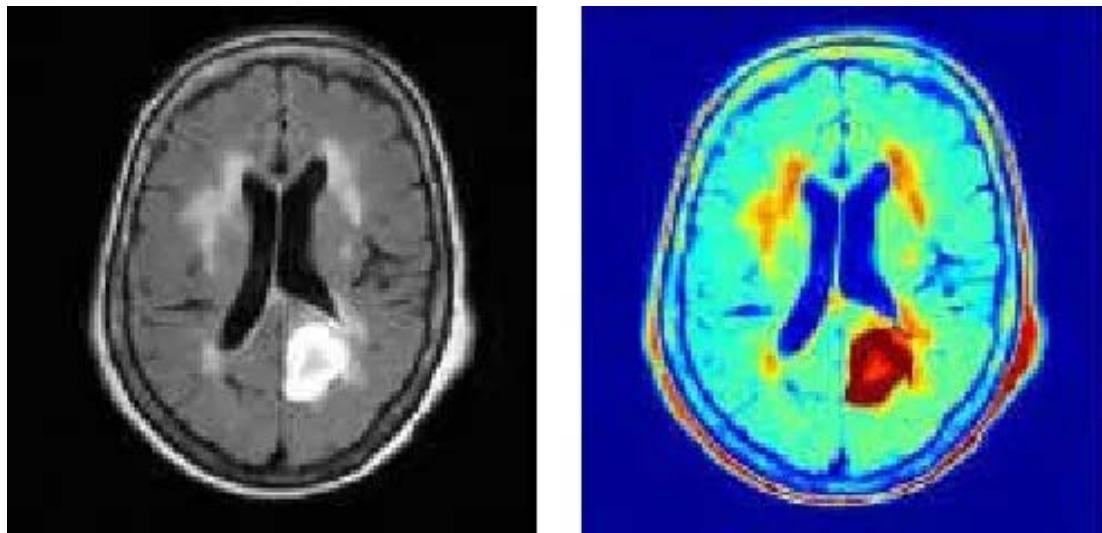
跨框架进行图像分类任务的CPU性能比较

即使使用最著名的神经网络框架之一的YOLOv3进行对象检测时，其运行速度也是不尽如人意的。此外，Darknet使用OpenMP（应用程序编程接口）进行编译的时间几乎是OpenCV的18倍。这更加说明了使用OpenCV的速度是比较快速的。



在OpenCV和Darknet上进行YOLOv3培训时CPU性能

颜色分割可用于检测身体肿瘤、从森林或海洋背景中提取野生动物的图像，或者从单一的背景图像中提取其他彩色物体。下面几幅图是图像分割的几个典型示例。：



医学中的颜色分割



颜色分割示例

从以上示例中可以看出，尽管OpenCV是一种更快的方法，但是它对于图像的分割结果并不是非常的理想，有时会出现分割误差或者错误分割的情况

接下来我们将介绍如何通过OpenCV对图像进行颜色的分割。这里我们有一张含有鸟的图片，我们的目标是通过颜色分割尝试从图片中提取这只鸟。



含鸟的图片

首先我们导入完成该任务所需的所有库和这张图像：

```
1 import cv2 as cv
2 import matplotlib.pyplot as plt from PIL
3 import Image
4 !wget -nv https://static.independent.co.uk/s3fs-public/thumbnails/image/2018/04/10/19
5 img = Image.open('./bird.png')
```

接下来我们使用滤波器对该图像进行预处理，对图像进行模糊操作，以减少图像中的细微差异。在OpenCV中提供了4个内置的滤波器，以满足用户对图像进行不同滤波的需求。这4种滤波器的使用方式在下面的代码中给出。但是，针对于本文中需要分割的图像，我们并不需要将4种滤波器都使用。

```
1 blur = cv.blur(img,(5,5))
2 blur0=cv.medianBlur(blur,5)
3 blur1= cv.GaussianBlur(blur0,(5,5),0)
4 blur2= cv.bilateralFilter(blur1,9,75,75)
```

下图是图像滤波模糊后的结果：



模糊后的图像

如果小伙伴对图像滤波感兴趣，可以在这里进行了解 https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html，这里再做过多的介绍。

接下来我们需要将图像从BGR（蓝绿色红色）转换为HSV（色相饱和度值）。为什么我们要从BGR空间中转到HSV空间中？因为像素B, G和R的取值与落在物体上的光相关，因此这些值也彼此相关，无法准确描述像素。相反，HSV空间中，三者相对独立，可以准确描述像素的亮度，饱和度和色度。

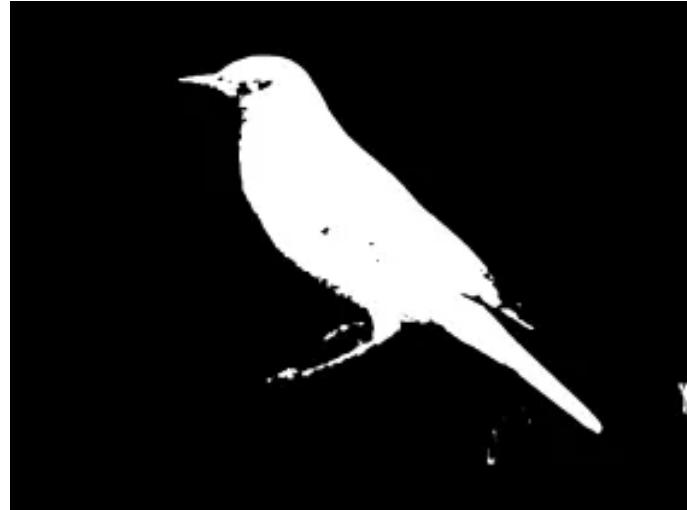
```
1 hsv = cv.cvtColor(blur2, cv.COLOR_BGR2HSV)
```

这个操作看似很小，但当我们尝试找到要提取的阈值或像素范围时，它会使我们的工作变得更加简单。

接下来是“颜色分割”的最重要一步，即“阈值分割”。这里我们将确定要提取的所有像素的阈值。使用OpenCV进行颜色分割中最重要步骤——阈值分割，这可能是一个相当繁琐的任务。即使我们可能想到通过使用颜色选择器工具来了解像素值，但是仍然需要进行不断的尝试，以便在所有像素中获取期望的像素，有些时候这也可能是一项艰巨的任务。具体操作如下：

```
1 low_blue = np.array([55, 0, 0])
2 high_blue = np.array([118, 255, 255])
3 mask = cv.inRange(hsv, low_blue, high_blue)
```

上面代码中最后一行的“Mask”将所有不在描述对象范围内的其他像素进行覆盖。程序运行结果如下图所示：



Mask

接下来，运行最后的代码以显示由Mask作为边界的图像。所使用的代码和程序运行结果在下面给出：

```
1 res = cv.bitwise_and(img, img, mask= mask)
```



从颜色分割中提取图像

那么通过上面的方式，我们就实现了基于颜色的图像分割，感兴趣的小伙伴们可以通过上面的代码和步骤进行尝试，看看能否满足自己的图像分割需求。

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过**。添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





小白学视觉

计算机视觉
论文解读 求职感想
SLAM技术 深度学习 学习感受

距离我们只差一个
长按关注

聚集地
计算机视觉学者

A black and white photograph of a Steinway & Sons grand piano keyboard, angled diagonally. The brand name 'STEINWAY & SONS' is visible on the piano's body.A square QR code with a blue robot icon in the center, located on the right side of the piano image.

//
//

使用OpenCV实现图像覆盖

原创 花生 小白学视觉 6月26日

来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

每张图像都包括RGB三个通道，分别代表红色、绿色和蓝色，使用它们来定义图像中任意一点的像素值，红绿蓝的值在0-255之间。

例如：一个像素值[255, 0, 0]代表全部为红色，像素值[255, 255, 0]是红色和绿色的混合，将显示为黄色。

但是，如果使用OpenCV读取图像，它将以BGR格式生成图像，那么[255, 0, 0]将代表蓝色。

使用OpenCV读取一张图像

任何图像都可以通过OpenCV使用cv2.imread()命令读取。不过，OpenCV不支持HEIC格式的图像，所以不得不使用其它类型的库，如Pillow来读取HEIC类型的图像（或者先将它们转换为JPEG格式）

```
1 import cv2
2 image = cv2.imread('image.jpg')
```

当读取图像之后，如果有必要的话可以将其从BGR格式转换为RGB格式，通过使用cv2.cvtColor()命令实现。

```
1 image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
2 image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

覆盖

图像可以看作是一堆像素值以类似矩阵的格式存储。任何像素的值都可以独立于其他像素进行更改。这里有一张图像，使用OpenCV读取图像：



image_1

```
1 image_1 = cv2.imread('image_1.jpg')
2 print(image_1)
```

这里将给出矩阵形式的一系列像素值

```
1 array([[[107, 108, 106], [107, 108, 106], [107, 108, 106], ..., [ 77, 78, 76], [ 77, 78, 76]
```

如果只改变图像某一区域的像素值，比如更改为`[0, 0, 0]`，这部分区域将变成黑色，因为这是颜色为黑色的像素值。同样，如果将像素值更改为`[255, 0, 0]`，则该区域将变为蓝色(OpenCV以BGR格式读取图像)。

```
1 image_1[50: 100, 50:100] = [255, 0, 0]
```



同样，这些像素值可以被另一幅图像替换，只需通过使用该图像的像素值。

为了做到这一点，我们需要将覆盖图像修改为要替换的像素值的大小。可以通过使用`cv2.resize()`函数来实现

```
1 image_2 = cv2.imread('image_2.jpg')
2 resized_image_2 = cv2.resize(image_2, dsize=(100, 100))
```

其中，`dsize` 代表图像要被修改的尺寸。

现在，可以将第二张图像够覆盖在第一张图片的上面

```
1 image_1[50:150, 50:150] = resized_image_2
```



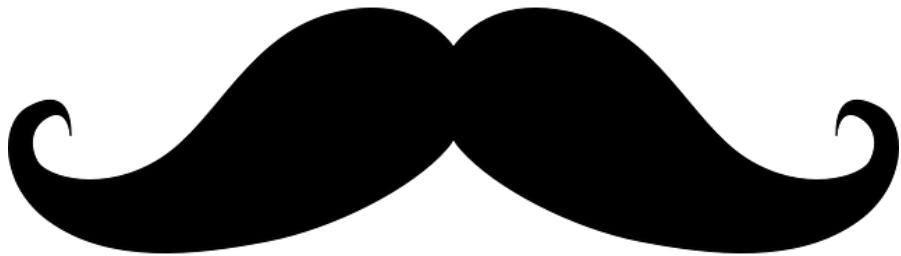
覆盖PNG图像

与JPEG图像不同，PNG图像有第四个通道，它定义了给定像素的ALPHA(不透明度)。

除非另有规定，否则OpenCV以与JPEG图像相同的方式读取PNG图像。

为了读取带有Alpha值的PNG图像，我们需要在读取一张图像时指定标志`cv2.IMREAD_UNCHANGED`。现在，这个图像已经有了四个通道：BGRA。

```
1 image_3 = cv2.imread('image_3.png', cv2.IMREAD_UNCHANGED)
2 print(image_3)
3 array([[[0 0 0 0][0 0 0 0][0 0 0 0]...[0 0 0 0][0 0 0 0][0 0 0 0]]...[[0 0 0 0][0 0 0 0][0 0 0 0]]])
```



然而，这个图像有4个通道，但是我们的JPEG图像只有3个通道，所以这些值不能简单地替换。

我们需要在我们的JPEG图像中添加一个虚拟通道。

为此，我们将使用 numpy。可以使用 pip install numpy 命令安装它。

numpy提供了一个函数numpy.dstack() 来根据深度叠加值。

首先，我们需要一个与图像大小相同的虚拟数组。

为了创建虚拟通道，我们可以使用numpy.ones() 函数创建一个数组。

```
1 import numpy as np
2 ones = np.ones((image_1.shape[0], image_1.shape[1])) * 255
3 image_1 = np.dstack([image_1, ones])
```

我们将其数组与255相乘，因为alpha通道的值也存在于0-255之间。

现在，我们可以用PNG图像替换图像的像素值。

```
1 image_1[150:250, 150:250] = image_3
```

然而，它不会给出期望的结果，因为我们将alpha通道的值改为了零。



我们只需要替换那些具有非零值的像素值。为了做到这一点，我们可以通过检查每个像素值和替换非零值来强行执行，但这很耗时。

这里有一个更好的方法。我们可以获取要覆盖图像的alpha值。

```
1 alpha_image_3 = image_3[:, :, 3] / 255.0
```

我们将像素值除以255.0，以保持值在0-1之间。

`image_1` 和 `image_3` 的 alpha 之和需要等于255。因此，我们可以创建另一个数组，其中包含和等于255的所需alpha值。

```
1 alpha_image = 1 - alpha_image_3
```

现在，我们可以简单的取每个图像的alpha值和每个通道的图像像素值的元素乘积，并取它们的和。

```
1 for c in range(0, 3): image_1[150:250, 150:250, c] = ((alpha_image*image_1[150:250,
```



交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过**。添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~



//
//

使用OpenCV进行图像全景拼接

原创 小白 小白学视觉 昨天

来自专辑

OpenCV应用

点击上方“[AI小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

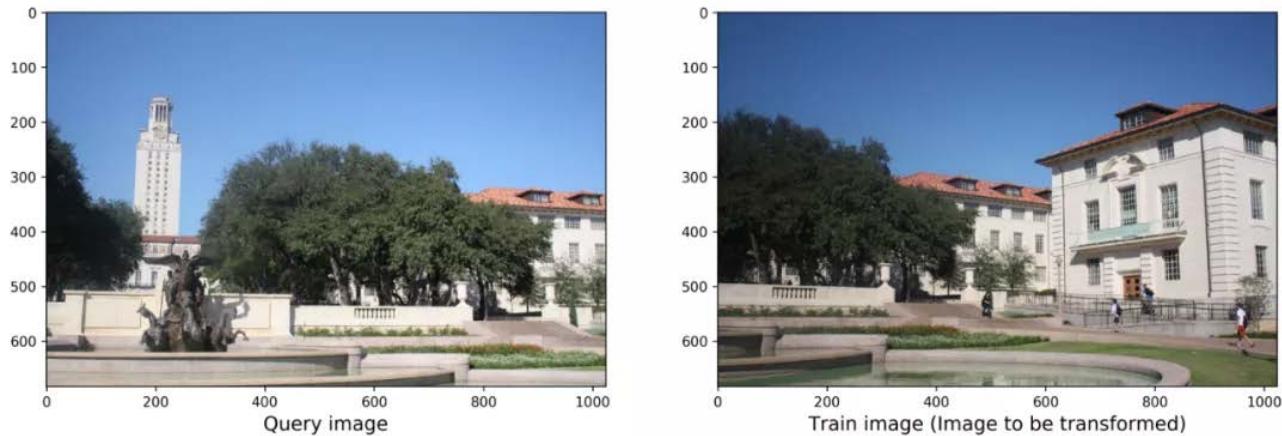


图像拼接是计算机视觉中最成功的应用之一。如今，很难找到不包含此功能的手机或图像处理API。在本文中，我们将讨论如何使用Python和OpenCV进行图像拼接。也就是，给定两张共享某些公共区域的图像，目标是“缝合”它们并创建一个全景图像场景。当然也可以是给定多张图像，但是总会转换成两张共享某些公共区域图像拼接的问题，因此本文以最简单的形式进行介绍。

本文主要的知识点包含一下内容：

- 关键点检测
- 局部不变描述符（SIFT, SURF等）
- 特征匹配
- 使用RANSAC进行单应性估计
- 透视变换

我们需要拼接的两张图像如下：



特征检测与提取

给定上述一对图像，我们希望将它们缝合以创建全景场景。重要的是要注意，两个图像都需要有一些公共区域。当然，我们上面给出的两张图像时比较理想的，有时候两个图像虽然具有公共区域，但是同样还可能存在缩放、旋转、来自不同相机等因素的影响。但是无论哪种情况，我们都需要检测图像中的特征点。

关键点检测

最初的并且可能是幼稚的方法是使用诸如Harris Corners之类的算法来提取关键点。然后，我们可以尝试基于某种相似性度量（例如欧几里得距离）来匹配相应的关键点。众所周知，角点具有一个不错的特性：角点不变。这意味着，一旦检测到角点，即使旋转图像，该角点仍将存在。

但是，如果我们旋转然后缩放图像怎么办？在这种情况下，我们会很困难，因为角点的大小不变。也就是说，如果我们放大图像，先前检测到的角可能会变成一条线！

总而言之，我们需要旋转和缩放不变的特征。那就是更强大的方法（如SIFT，SURF和ORB）。

关键点和描述符

诸如SIFT和SURF之类的方法试图解决角点检测算法的局限性。通常，角点检测器算法使用固定大小的内核来检测图像上的感兴趣区域（角）。不难看出，当我们缩放图像时，该内核可能变得太小或太大。为了解决此限制，诸如SIFT之类的方法使用高斯差分（DoD）。想法是将DoD应用于同一图像的不同缩放版本。它还使用相邻像素信息来查找和完善关键点和相应的描述符。

首先，我们需要加载2个图像，一个查询图像和一个训练图像。最初，我们首先从两者中提取关键点和描述符。通过使用OpenCV `detectAndCompute()`函数，我们可以一步完成它。请注意，为了使用`detectAndCompute()`，我们需要一个关键点检测器和描述符对象的实例。它可以是ORB，SIFT或SURF等。此外，在将图像输入给`detectAndCompute()`之前，我们将其转换为灰度。

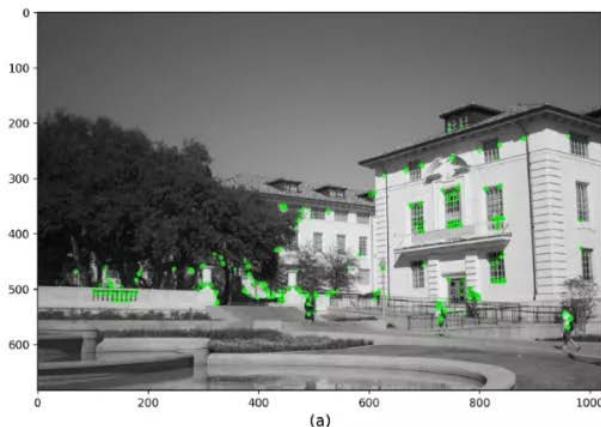
```

1 def detectAndDescribe(image, method=None):
2     """
3         Compute key points and feature descriptors using an specific method
4     """
5
6     assert method is not None, "You need to define a feature detection method. Value
7
8     # detect and extract features from the image
9     if method == 'sift':
10         descriptor = cv2.xfeatures2d.SIFT_create()
11     elif method == 'surf':
12         descriptor = cv2.xfeatures2d.SURF_create()
13     elif method == 'brisk':
14         descriptor = cv2.BRISK_create()
15     elif method == 'orb':
16         descriptor = cv2.ORB_create()
17
18     # get keypoints and descriptors
19     (kps, features) = descriptor.detectAndCompute(image, None)
20
21     return (kps, features)

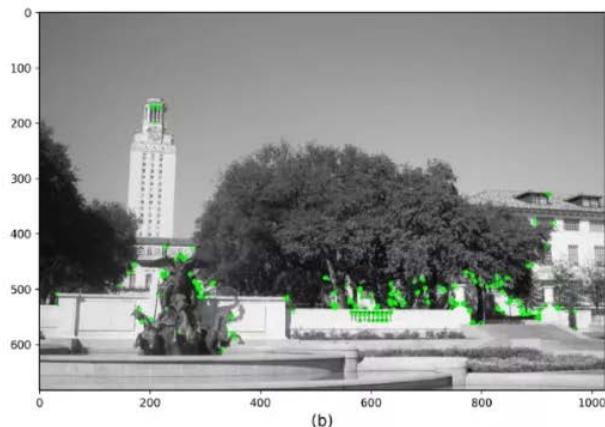
```



我们为两个图像都设置了一组关键点和描述符。如果我们使用SIFT作为特征提取器，它将为每个关键点返回一个128维特征向量。如果选择SURF，我们将获得64维特征向量。下图显示了使用SIFT，SURF，BRISK和ORB得到的结果。

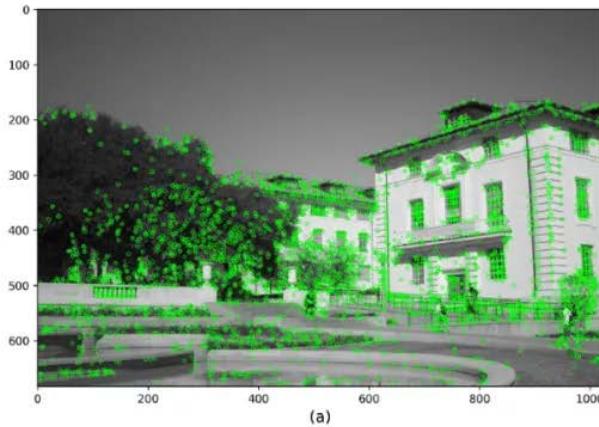


(a)

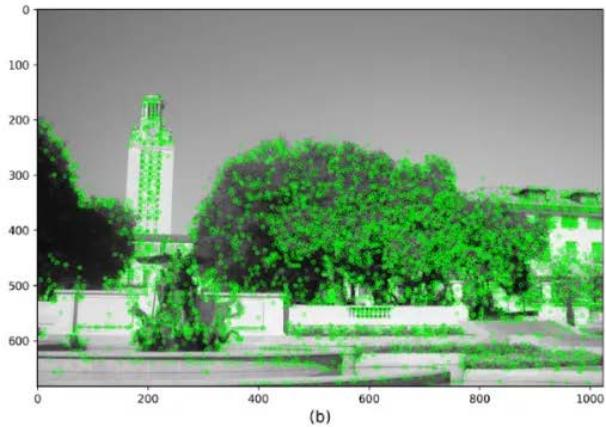


(b)

使用ORB和汉明距离检测关键点和描述符

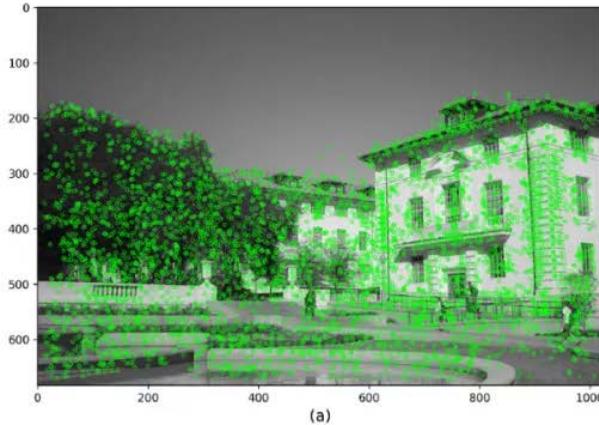


(a)

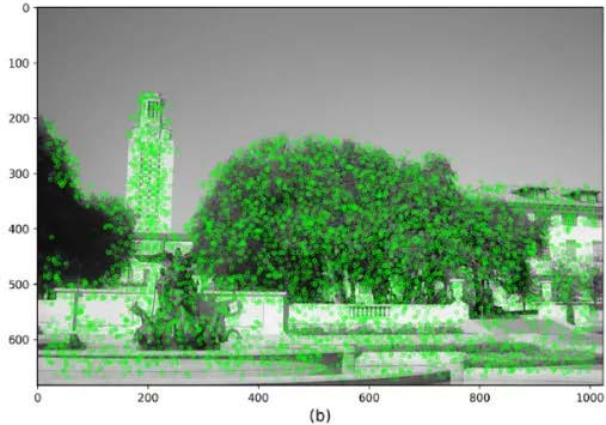


(b)

使用SIFT检测关键点和描述符

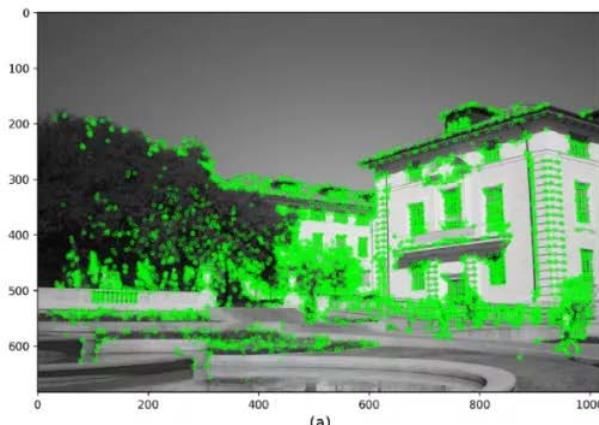


(a)

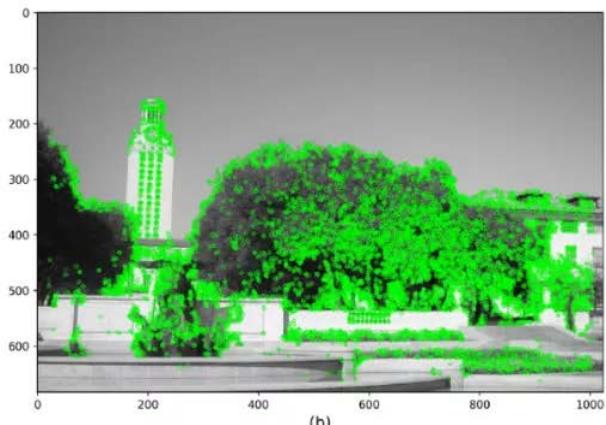


(b)

使用SURF检测关键点和描述符



(a)



(b)

使用BRISK和汉明距离检测关键点和描述符

特征匹配

如我们所见，两个图像都有大量特征点。现在，我们想比较两组特征，并尽可能显示更多相似性的特征点对。使用 OpenCV，特征点匹配需要 Matcher 对象。在这里，我们探索两种方式：暴力匹配器（BruteForce）和 KNN（k 最近邻）。

BruteForce (BF) Matcher的作用恰如其名。给定2组特征（来自图像A和图像B），将A组的每个特征与B组的所有特征进行比较。默认情况下，BF Matcher计算两点之间的欧式距离。因此，对于集合A中的每个特征，它都会返回集合B中最接近的特征。对于SIFT和SURF，OpenCV建议使用欧几里得距离。对于ORB和BRISK等其他特征提取器，建议使用汉明距离。我们要使用OpenCV创建BruteForce Matcher，一般情况下，我们只需要指定2个参数即可。第一个是距离度量。第二个是是否进行交叉检测的布尔参数。具体代码如下：

```
1 def createMatcher(method,crossCheck):
2     "Create and return a Matcher Object"
3
4     if method == 'sift' or method == 'surf':
5         bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=crossCheck)
6     elif method == 'orb' or method == 'brisk':
7         bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=crossCheck)
8
9     return bf
```

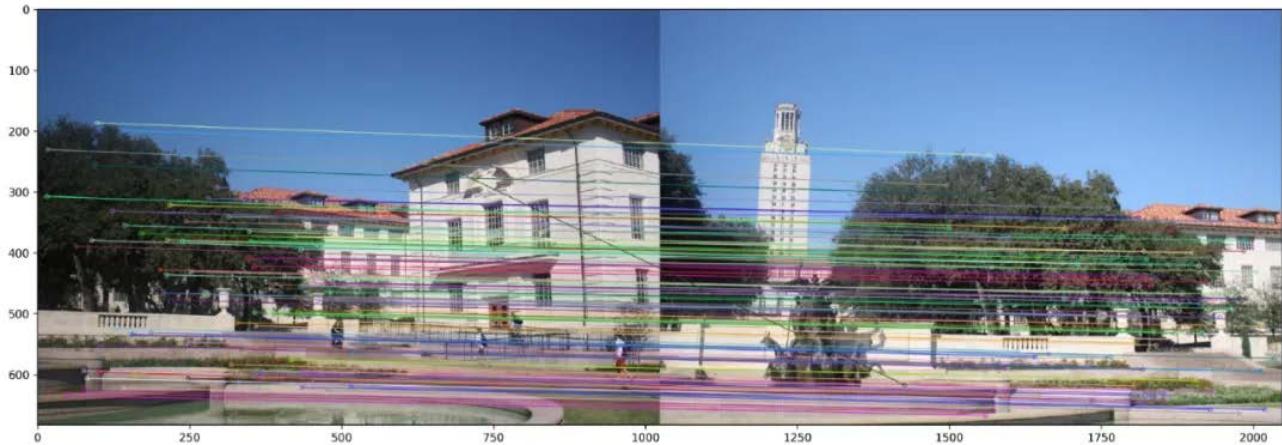
交叉检查布尔参数表示这两个特征是否具有相互匹配才视为有效。换句话说，对于被认为有效的一对特征(f_1, f_2)， f_1 需要匹配 f_2 ， f_2 也必须匹配 f_1 作为最接近的匹配。此过程可确保提供更强大的匹配功能集，这在原始SIFT论文中进行了描述。

但是，对于要考虑多个候选匹配的情况，可以使用基于KNN的匹配过程。KNN不会返回给定特征的单个最佳匹配，而是返回k个最佳匹配。需要注意的是，k的值必须由用户预先定义。如我们所料，KNN提供了更多的候选功能。但是，在进一步操作之前，我们需要确保所有这些匹配对都具有鲁棒性。

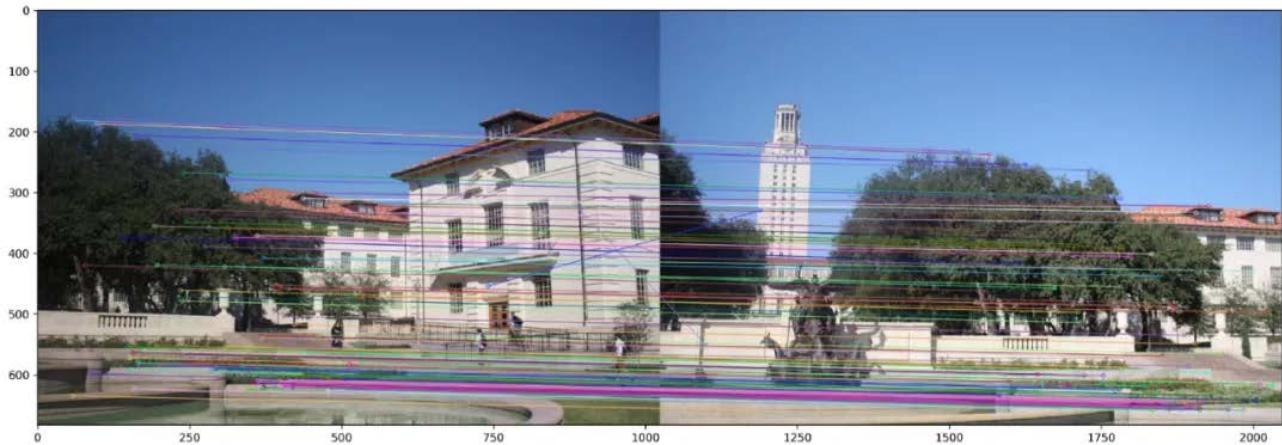
比率测试

为了确保KNN返回的特征具有很好的可比性，SIFT论文的作者提出了一种称为比率测试的技术。一般情况下，我们遍历KNN得到匹配对，之后再执行距离测试。对于每对特征(f_1, f_2)，如果 f_1 和 f_2 之间的距离在一定比例之内，则将其保留，否则将其丢弃。同样，必须手动选择比率值。

本质上，比率测试与BruteForce Matcher的交叉检查选项具有相同的作用。两者都确保一对检测到的特征确实足够接近以至于被认为是相似的。下面2个图显示了BF和KNN Matcher在SIFT特征上的匹配结果。我们选择仅显示100个匹配点以清晰显示。



使用KNN和SIFT的定量测试进行功能匹配



在SIFT特征上使用暴力匹配器进行特征匹配

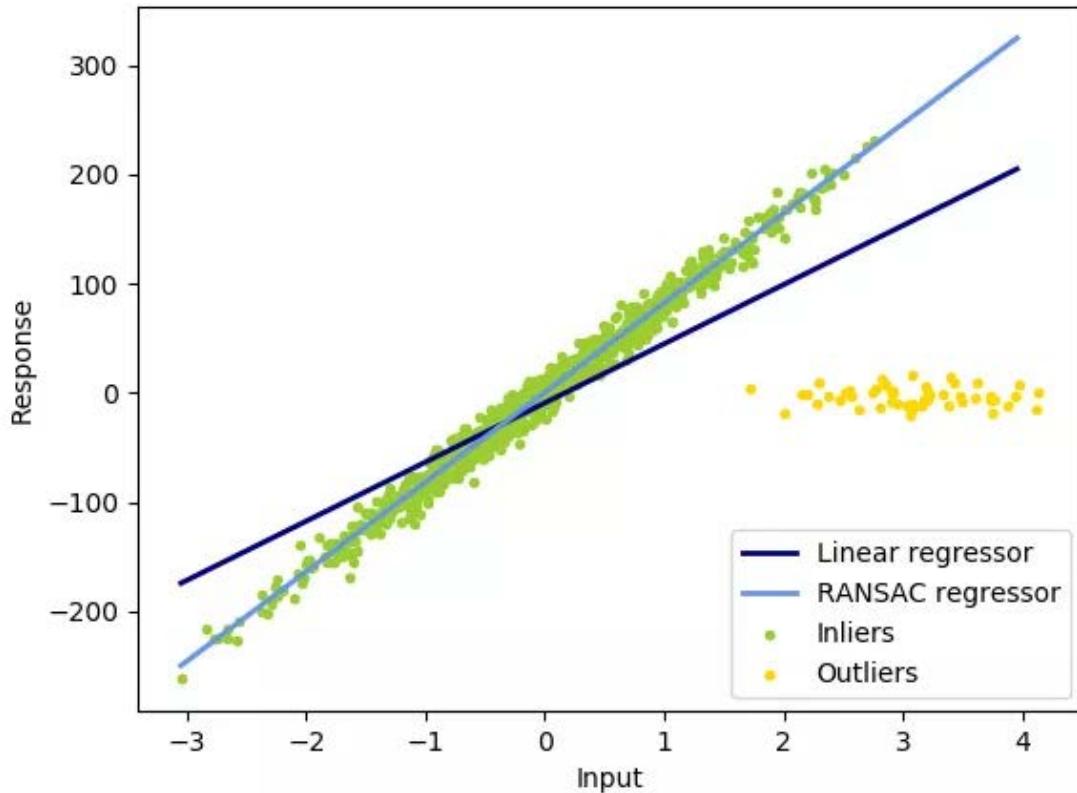
需要注意的是，即使做了多种筛选来保证匹配的正确性，也无法完全保证特征点完全正确匹配。尽管如此，Matcher算法仍将为我们提供两幅图像中最佳（更相似）的特征集。接下来，我们利用这些点来计算将两个图像的匹配点拼接在一起的变换矩阵。

这种变换称为单应矩阵。简而言之，单应性是一个 3×3 矩阵，可用于许多应用中，例如相机姿态估计，透视校正和图像拼接。它将点从一个平面（图像）映射到另一平面。

估计算法

随机采样一致性（RANSAC）是用于拟合线性模型的迭代算法。与其他线性回归器不同，RANSAC被设计为对异常值具有鲁棒性。

像线性回归这样的模型使用最小二乘估计将最佳模型拟合到数据。但是，普通最小二乘法对异常值非常敏感。如果异常值数量很大，则可能会失败。RANSAC通过仅使用数据中的一组数据估计参数来解决此问题。下图显示了线性回归和RANSAC之间的比较。需要注意数据集包含相当多的离群值。



我们可以看到线性回归模型很容易受到异常值的影响。那是因为它试图减少平均误差。因此，它倾向于支持使所有数据点到模型本身的总距离最小的模型。包括异常值。相反，RANSAC仅将模型拟合为被识别为点的点的子集。

这个特性对我们的用例非常重要。在这里，我们将使用RANSAC来估计单应矩阵。事实证明，单应矩阵对我们传递给它的数据质量非常敏感。因此，重要的是要有一种算法（RANSAC），该算法可以从不属于数据分布的点中筛选出明显属于数据分布的点。

估计了单应矩阵后，我们需要将其中一张图像变换到一个公共平面上。在这里，我们将对其中一张图像应用透视变换。透视变换可以组合一个或多个操作，例如旋转，缩放，平移或剪切。我们可以使用OpenCV warpPerspective()函数。它以图像和单应矩阵作为输入。

```

1 # Apply panorama correction
2 width = trainImg.shape[1] + queryImg.shape[1]
3 height = trainImg.shape[0] + queryImg.shape[0]
4
5 result = cv2.warpPerspective(trainImg, H, (width, height))
6 result[0:queryImg.shape[0], 0:queryImg.shape[1]] = queryImg
7
8 plt.figure(figsize=(20,10))
9 plt.imshow(result)
10
11 plt.axis('off')

```

```
12 plt.show()
```

生成的全景图像如下所示。如我们所见，结果中包含了两个图像中的内容。另外，我们可以看到一些与照明条件和图像边界边缘效应有关的问题。理想情况下，我们可以执行一些处理技术来标准化亮度，例如直方图匹配，这会使结果看起来更真实和自然一些。







交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM、三维视觉、传感器、自动驾驶、计算摄影、检测、分割、识别、医学影像、GAN、算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过。**添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





//
//

利用OpenCV实现图像修复（含源码链接）

原创 小白 小白学视觉 2019-04-22



前一段时间小白分享过关于图像修复技术介绍的推文（点击可以跳转），有小伙伴后台咨询能不能分享一下关于图像修复的项目或者程序。今天小白带着满满的诚意，带来了通过OpenCV实现图像修复的C++代码与Python代码。

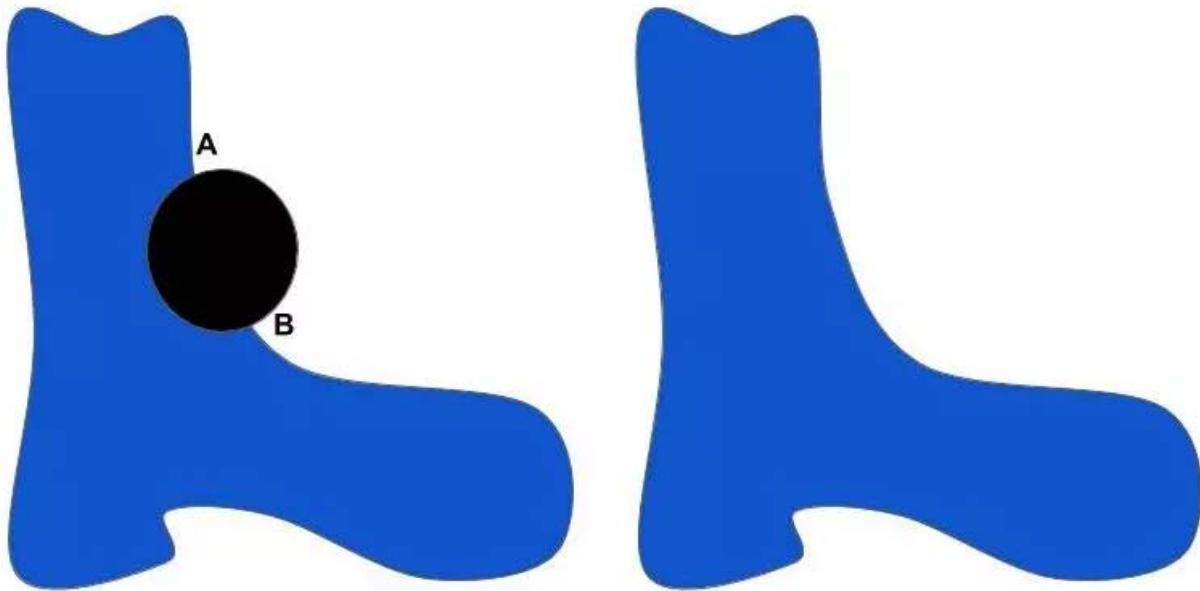
图像修复技术应用在什么地方呢？

想想一下，我们有一张非常棒的相片，但是由于时间比较久远，没有电子版留底，而纸质版的又十分不利于保存。因此长采用扫描的方式获得电子版。但是非常不幸，扫描过程中落入了一根头发，或者是机器出现故障，对相片造成了影响，这个时候就可以通过图像修复技术解决这个问题。

强大的**OpenCV库**里集成了**两种**用与图像修复的方法

INPAINT_NS: 基于Navier-Stokes的图像修复

该方法在2001年提出，其神奇之处竟然是基于流体力学理论提出的方法。根据其作者提出，我们需要解决的问题可以抽象成在一个鞋子图片上有一个黑色的区域，通过填充黑色区域，使得最佳的恢复鞋子的样子。



对于如何填补这个黑色区域，可以抽象成存在一条曲线，使得由A到B将黑色区域分开，并且保证在曲线的一侧是蓝色，另一侧是白色。这个曲线应具有如下的约束：

1. 保持边缘特征
2. 在平滑区域中保持颜色信息

通过构建一个偏微分方程来更新具有上诉约束的区域内的图像强度，同时利用拉普拉斯算子估计图像平滑度信息，并沿着等照度传播。

由于这些方程与Navier-Stokes方程（流体力学中的方程，感兴趣的小伙伴可以自行百度）相关且类似，因此可以通过流体力学中的方法进行求解。

由于小白对流体力学不是很了解，具体就不详细解释了，感兴趣的小伙伴可以阅读该论文了解详情。

[论文地址：http://www.math.ucla.edu/~bertozzi/papers/cvpr01.pdf](http://www.math.ucla.edu/~bertozzi/papers/cvpr01.pdf)

INPAINT_TELEA：基于快速行进方法的图像修复

该方法中没有使用拉普拉斯算子作为平滑度的估计，而是使用像素的已知图像邻域上的加权平均值来描述。同时利用邻域像素和梯度恢复填充区域像素的颜色。

当像素被修复之后，通过快速行进方法更新边界。

[论文地址：](#)

<https://pdfs.semanticscholar.org/622d/5f432e515da69f8f220fb92b17c8426d0427.pdf>

相关API介绍

C++：

```
void inpaint(
    Mat& src,
    Mat& inpaintMask,
    Mat& dst,
    double inpaintRadius,
    int flags)
```

Python:

```
dst = cv2.inpaint(
    src,
    inpaintMask,
    inpaintRadius,
    flags)
```

其中各参数的含义如下：

src =源图像

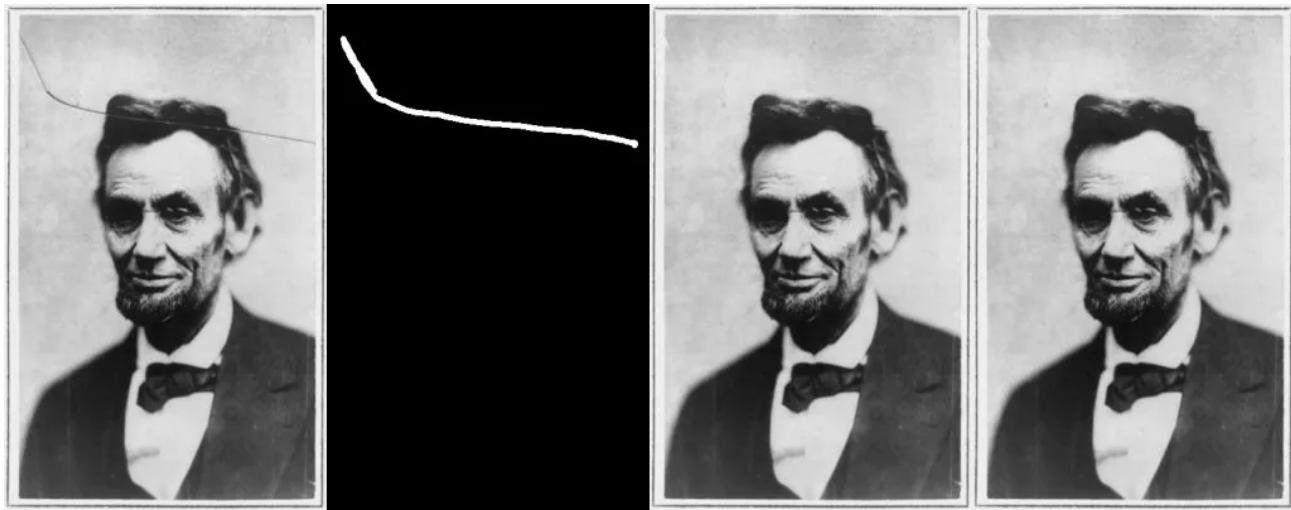
inpaintMask =二进制掩码，指示要修复的像素。

DST =目标图像

inpaintRadius =像素周围的邻域补绘。通常，如果要修复的区域很薄，使用较小的值会产生较少的模糊。

flags: INPAINT_NS (基于Navier-Stokes的方法) 或INPAINT_TELEA (基于快速行进的方法)

示例



左边的第一个图像是输入图像，第二个图像是掩模，第三个图像是INPAINT_TELEA的结果，最终结果是INPAINT_NS

关于这个图片有一个小小的故事，1865年2月5日星期日，在华盛顿特区的加德纳画廊，亚历山大·加德纳拍摄了几张总统的多镜头照片。在本届会议结束之前，加德纳请求为总统拍摄最后一个姿势。他把相机拉得更近，拍了一张林肯头部、肩膀和胸部的照片。但是玻璃板在这个时候突然破裂，对拍摄图像产生影响。加德纳小心翼翼地将它带到了他的黑暗房间，制作一张相片，发现在林肯的脸上有一个不祥的裂缝。这张相片，即O-118，至今仍然存在。多年来，许多人认为这一裂缝是10周后林肯中弹的预言。

让我们看一个更复杂的例子，在图片上写上英文单词，之后通过opencv函数去修复该单词。



左：带有Scribbles的原始图像。中：使用快速行进方法修复，右：使用Navier-Stokes方法修复。

该程序的源码和使用的图片链接为：

<https://github.com/spmallick/learnopencv/tree/master/Image-Inpainting>

往期文章一览



- 1、[【OpenCV入门之十五】随心所欲绘制想要图形](#)
- 2、[【OpenCV入门之十四】揭开mask](#)
- 3、[【OpenCV入门之十三】如何在ROI中添加Logo](#)
- 4、[如何让黑白相片恢复生机](#)
- 5、[我竟然用OpenCV实现了卡尔曼滤波](#)
- 6、[【走进OpenCV】滤波代码原来这么写](#)
- 7、[【走进OpenCV】这样腐蚀下来让我膨胀](#)
- 8、[小心！你看到的图像可能隐藏了重大机密](#)



//
//

基于自适应显着性的图像分割（源码开放）

小白 小白学视觉 4月13日

点击上方“**小白学视觉**”，选择“**星标**”公众号

重磅干货，第一时间送达

本文介绍算法的源码在github上给出

<https://github.com/TimChinenov/GraspPicture>

前言

成产品及系统平台的现场演示，编写技术应用服务方案等，编写投标类方案文件及标书的制作；通常，当我们看到一张图片时，会在图片中聚焦一个焦点。这个可能是一个人，一座建筑物甚至是一个桶。其他没有聚焦区域虽然很清晰，但是却由于颜色单调或者纹理较为平滑而很少引起关注。当遇到此类图象时，我们希望从图像中分割感兴趣的对像。下面给出了显着图像的示例，本文探讨了此类显着图像的分割方法，也称为显着性的图像分割。



显着图像的示例。桶（左）和人（右）是感兴趣的对像

这种分割方式最开始起源于希望能够自主寻找图像中的Trimap。Trimap是图像掩码（mask），当与掩码算法配合使用时，可用于分割图像，同时能够提示前景和背景之间的细节。Trimap通常包含定义前景的白色区域，定义背景的黑色区域以及代表不确定区域的灰色区域。具体形式如下图所示。



Trimap示例

大部分抠图算法问题在于，他们希望Trimap由用户提供，这是一项非常耗时的任务。这里面介绍两个试图解决自主trimap生成问题的相关论文，这两篇论文在文末给出。在第一篇论文中使用了一种相当简单且易于实现的方法。不幸的是，他们的方法并不是完全自主的，因为它要求用户为Grabcut算法提供一个矩形区

域。第二篇论文中，使用显着性方法预测感兴趣的区域。但是，它们的显着性方法非常复杂，将三种不同的显着性算法的结果结合在一起。这三种算法中有一种利用卷积神经网络，为了易于实现，应该尽量避免这种技术。

如果忽略需要人为给出矩形区域，第一篇论文中能够产生较好的分割结果。通过第二篇论文的原理去自动给出一个Grabcut算法的矩形区域，那么将完美的解决自主分割的问题。

方法

对于大多数形式的图像分割，目标都是将图像二值化为感兴趣的区域。这个本文介绍方法的目标也是这样的。首先，大致确定感兴趣的对象在哪里。将高斯模糊应用于图像，之后在模糊图像中生成平均15像素大小的超像素。超像素算法旨在根据像素区域中值的颜色和距离来分解图像。具体来说，使用了简单的线性迭代聚类（SLIC）算法。具体形式如下图所示。



一个桶和一个人的超像素划分结果

超像素将图像分解为大致相同的区域。这样的一个优点是，超像素允许区域的泛化。我们可以假设超像素内的大多数像素具有相似的属性。

在确定图像中的超像素的同时，计算图像的显着性图。使用了两种不同的显着性技术。第一种方法使用OpenCV内置的方法，即所谓的细颗粒显着性。第二种方法涉及获取细颗粒显着性图像的平均值，然后从图像的高斯模糊版本中减去平均值，然后是新图像的绝对值。

下方的图像均突出显示了感兴趣的区域。细颗粒显着性产生的图像较为柔和。此外，细颗粒显着性图像主要勾勒出突出图像的边界。而另一种方法虽然也捕获了突出图像的内部，但是与细颗粒方法相比，该方法会产生更多的噪音。之后需要对噪声进行去除。



第一种显着性结果



第二种显着性结果

为了将图像二值化，对从彩色图像生成的每个超级像素进行迭代。如果显着图像内该超像素区域的中值像素值大于阈值T1，则整个超像素将被二值化为白色。否则，整个超像素将保留为黑色。T1由用户选择，一般情况下，将T1设置为显着图像中最大像素值的25%-30%。

在对图像进行二值化之后，基于所使用的显着性技术对图像进行扩张。在第一种方法中，将图像放大为平均超像素尺寸的两倍。在第二种方法中没有进行扩大，因为图像中存在的较大噪声使扩张风险增大。处理的结果在下面给出。



最后一步操作取决于使用的是哪种显着性。在这两种方法的结果中，都提取最大的白色像素区域。通过查找图像中的轮廓并选择面积最大的轮廓来执行此操作，之后将边界框拟合到所选区域。

根据一般性结果，第一种显着性方法通常会导致区域碎片化。生成边界框后，将落入该框的不属于最大区域的所有其他白色区域添加到该框。框的边界增加到包括这些区域。第二种显着性方法不需要这样做。通常，

最大获取的区域会超出期望的数量。

最后一步是将最终找到的边界框提供给Grabcut算法。Grabcut是用于分割图像的常用方法，该方法会将绝对是背景和前景的内容分开。这里面我们直接使用OpenCV的内置Grabcut函数。处理的结果如下所示。

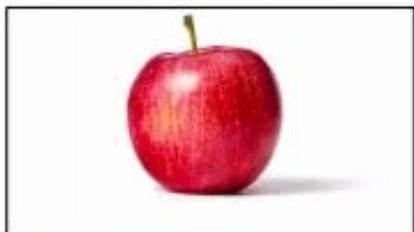


结果

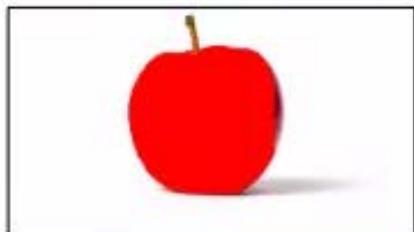
两种显着性计算方法对于结果会有一些影响。第一种显着性方法更加适用于含有噪声的图像中，在含有噪声的图像中不会像第二种显着性方法造成分割结果的溢出。但是如果图像太长或有卷须，则这些部分通常会与图像的其余部分断开连接。

下面是这两种方法分割更多图像的示例结果。

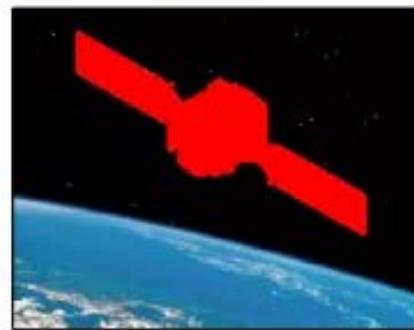
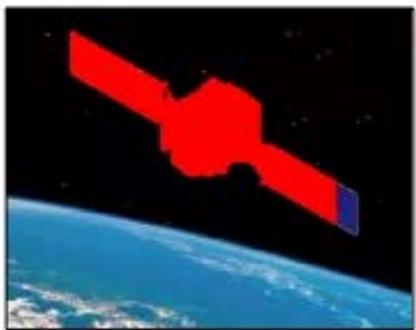
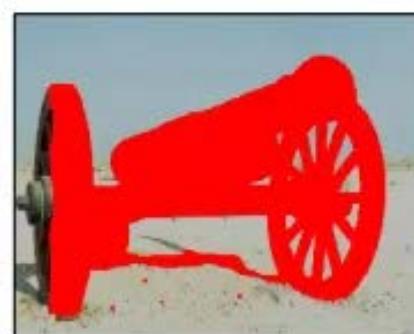
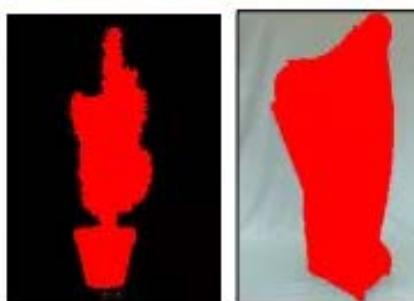
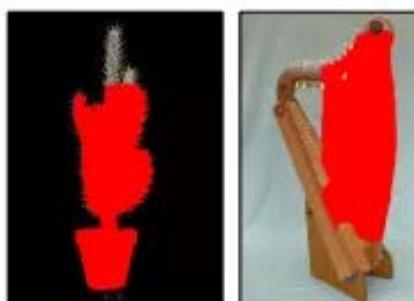
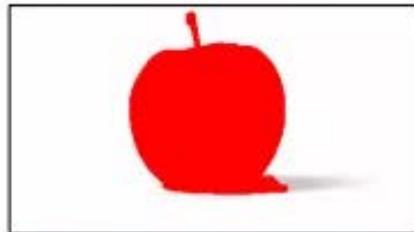
Original Image



Fine Grain Approach



Aggressive Approach



参考文献：

- [1] C. Hsieh and M. Lee, "Automatic trimap generation for digital image matting," 2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, Kaohsiung, 2013, pp. 1–5.
- [2] Gupta, Vikas & Raman, Shanmuganathan. (2017). Automatic Trimap Generation for Image Matting.

原文地址：<https://towardsdatascience.com/saliency-based-image-segmentation-473b4cb31774>

作者：Tim Chin



基于OpenCV实现海岸线变化检测

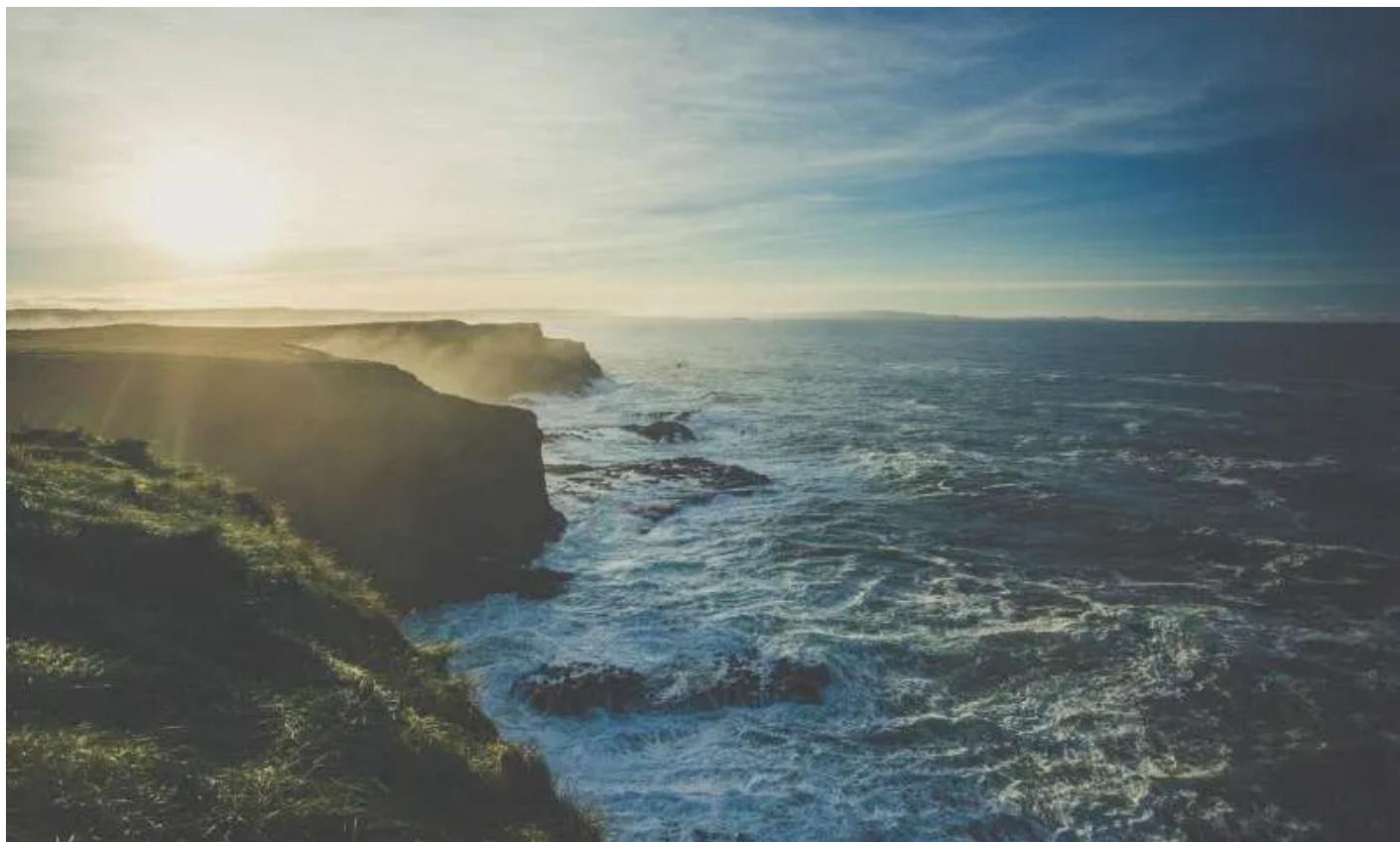
原创 努比 小白学视觉 前天

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”
重磅干货，第一时间送达

介绍

海岸是一个动态系统，其中因侵蚀现象导致的海岸线后退、或是由众多因素如气象，地质，生物和人类活动所导致线前进的是常见现象。

在海洋磨损作用大于沉积物的情况下，有明显的海岸侵蚀，我们称之为**地球表面的崩解和破坏**。



资料来源：弗林德斯大学 (CC0)

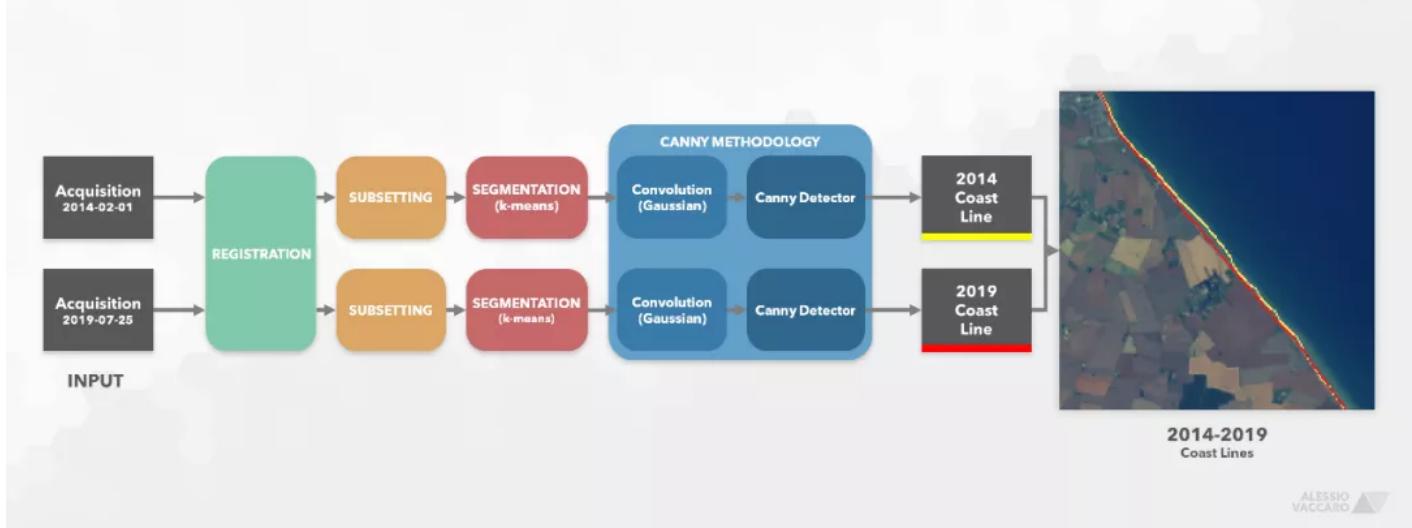
本文的目标

在本文中，我们将对Landsat 8平台上的**OLI (陆地成像仪) 传感器**获取的卫星图像使用**Canny Edge Detection**算法。

通过这种方法，**我们将能够可视化的估计**特定欧洲地区遭受强腐蚀作用的**海岸线随时间的推移**：霍德内斯海岸。



一下是处理流程:



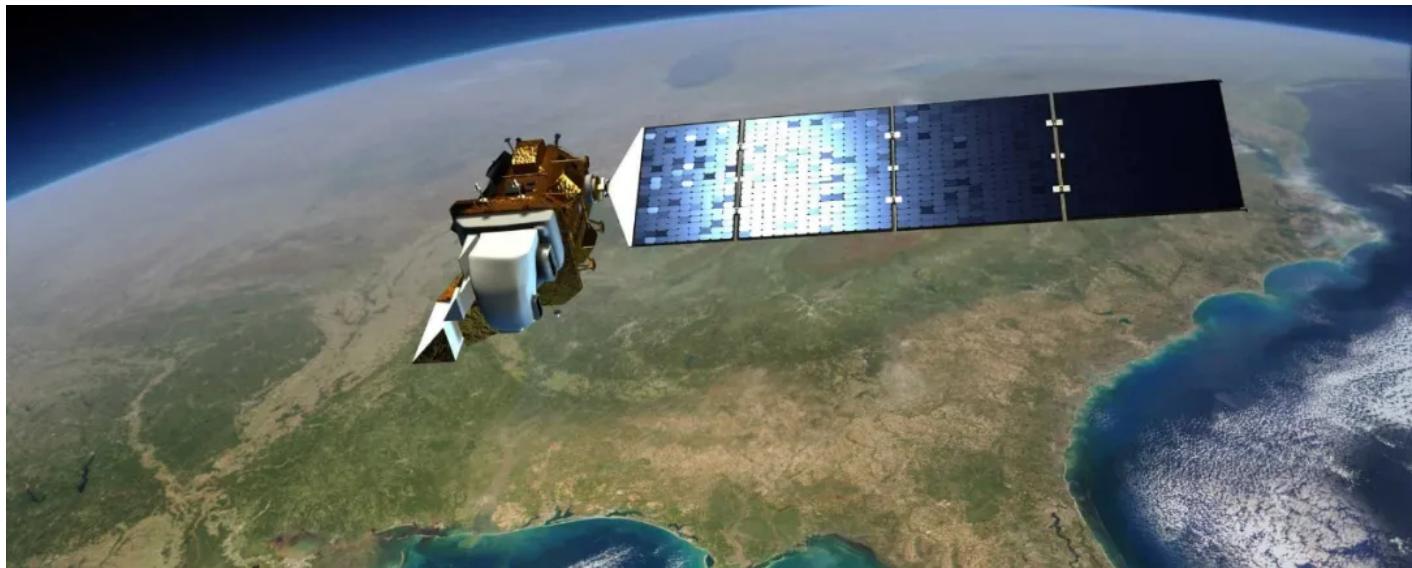
处理流程

在开始之前让我们先介绍一下OLI数据...

0.关于Landsat OLI数据的简要介绍

Landsat 8是一个轨道平台，安装在称为OLI（陆地成像仪）的**11波段多光谱传感器上**。

具体来说，在本文中，我们将仅使用分辨率为30米（即前7个）的波段。



美国地质调查局陆地卫星8号

该数据可以免费下载，注册后，获得USGS: [https://earthexplorer.usgs.gov/。](https://earthexplorer.usgs.gov/)

而且，通常我摸并不使用入射太阳光作为原始数据，而是使用反射率，即从地球表面反射的太阳光量[0-1]。

1.包导入

在各种常见的包，我们将使用*rasterio*处理图像，利用*OpenCV*中的Canny 算法和*Scikit-Learn*分割图像。

```

1 from glob import glob
2 import numpy as np
3
4 import rasterio
5 import json, re, itertools, os
6
7 import matplotlib.pyplot as plt
8
9 import cv2 as cv
10 from sklearn import preprocessing
11 from sklearn.cluster import KMeans

```

2.数据导入

让我们定义一个变量，该变量告诉我们要保留的波段数以及在JSON中输入的辅助数据：

```

1 N_OPTICS_BANDS = 7
2
3 with open("bands.json","r") as bandsJson:
4     bandsCharacteristics = json.load(bandsJson)

```

这个Json是Landsat OLI成像仪的信息集合。类似于一种说明手册：

```

1 # bands.json
2
3 [{"id": "1", "name": "Coastal aerosol", "span": "0.43-0.45", "resolution": "30"},  

4 {"id": "2", "name": "Blue", "span": "0.45-0.51", "resolution": "30"},  

5 {"id": "3", "name": "Green", "span": "0.53-0.59", "resolution": "30"},  

6 {"id": "4", "name": "Red", "span": "0.64-0.67", "resolution": "30"},  

7 {"id": "5", "name": "NIR", "span": "0.85-0.88", "resolution": "30"},  

8 {"id": "6", "name": "SWIR1", "span": "0.64-0.67", "resolution": "30"},  

9 {"id": "7", "name": "SWIR2", "span": "0.85-0.88", "resolution": "30"}]

```

```

8  {'id': '6', 'name': 'SWIR 1', 'span': '1.57-1.65', 'resolution': '30'},
9  {'id': '7', 'name': 'SWIR 2', 'span': '2.11-2.29', 'resolution': '30'},
10 {'id': '8', 'name': 'Panchromatic', 'span': '0.50-0.68', 'resolution': '15'},
11 {'id': '9', 'name': 'Cirrus', 'span': '1-36-1.38', 'resolution': '30'},
12 {'id': '10', 'name': 'TIRS 1', 'span': '10.6-11.9', 'resolution': '100'},
13 {'id': '11', 'name': 'TIRS 2', 'span': '11.50-12.51', 'resolution': '100'}]

```

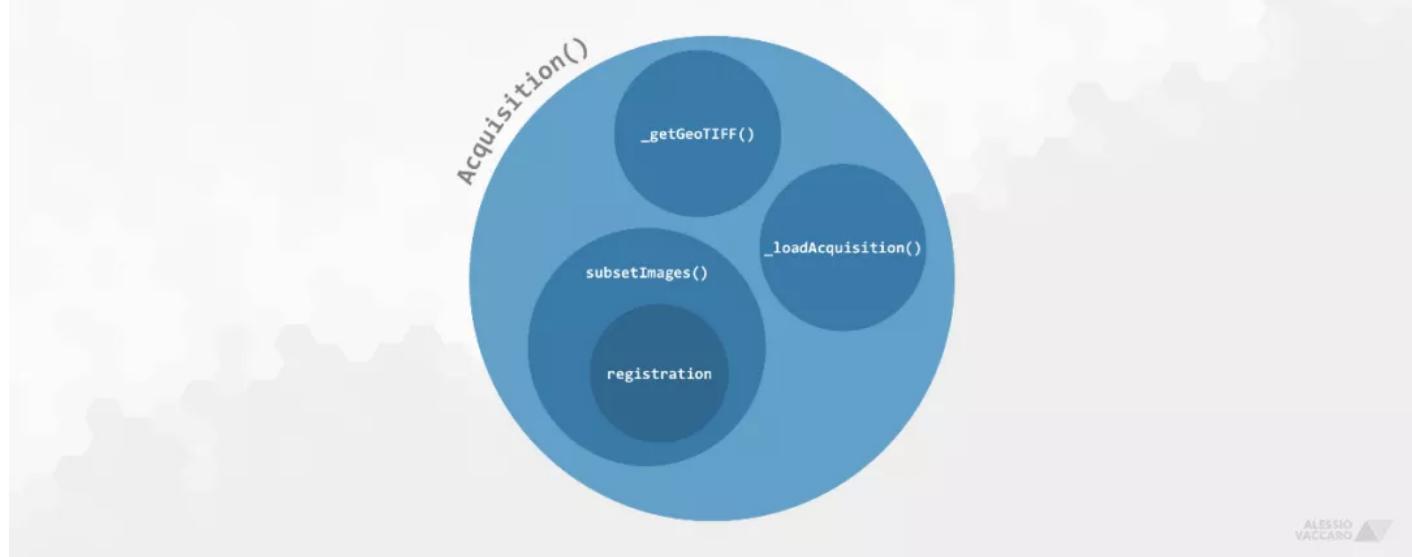
`bands.json`文件包含有关我们将要使用的频段的所有有用信息。

注意，我们将仅使用分辨率为30 m的频段，因此仅使用前7个频段。如果您愿意使用较低的分辨率（100m），则也可以嵌入**TIRS 1**和**TIRS 2**频段。

正如上面几行已经提到的那样，我们将使用从*Landsat-8 OLI*上获取两组不同的数据：

- 2014/02/01
- 2019/07/25

为了简化两次采集的所需操作，**我们将定义一个Acquisition () 类**，其中将封装所有必要的函数。



在执行代码期间，我们能够执行一些基础支持性的功能，例如：

- 在指定路径中搜索GeoTIFF；
- **加载采购**；
- **购置登记**（调整）；
- **收购子集**

```

1 class Acquisition:
2
3     def __init__(self, path, ext, nOpticsBands):
4         self.nOpticsBands = nOpticsBands
5         self._getGeoTIFFs(path, ext)
6         self.images = self._loadAcquisition()
7
8     def _getGeoTIFFs(self, path, ext):
9         # It searches for GeoTIFF files within the folder.
10        print("Searching for '%s' files in %s" % (ext, path))
11        self.fileList = glob(os.path.join(path, "*." + ext))
12        self.opticsFileList = [      [list(filter(re.compile(r"band%a\.%a").search, se
13        print("Found %d 'tif' files" % len(self.opticsFileList))
14
15    def _loadAcquisition(self):
16        # It finally reads and loads selected images into arrays.
17        print("Loading images")
18        self.loads = [rasterio.open(bandPath) for bandPath in self.opticsFileList]
19        images = [load.read()[0] for load in self.loads]
20        print("Done")
21
22        return images
23
24    def subsetImages(self, w1, w2, h1, h2, leftBound):
25        # This function subsets images according the defined sizes.
26        print("Subsetting images (%s:%s, %s:%s)" % (w1, w2, h1, h2))
27        cols = (self.loads[0].bounds.left - leftBound)/30
28        registered = [np.insert(band, np.repeat(0, cols), 0, axis=1) for band in self.im
29        subset = [band[w1:w2, h1:h2] for band in registered]
30        print("Done")
31
32        return subset

```

好的，让我们现在开始启动整个代码：

```

1 DATES = ["2014-02-01", "2019-07-25"]
2 acquisitionsObjects = []
3
4 for date in DATES:
5     singleAcquisitionObject = Acquisition("Data/" + date, "tif", N_OPTICS_BANDS)
6     acquisitionsObjects.append(singleAcquisitionObject )

```

运行结果如下：

Searching for 'tif' files in Data/2014-02-01

Found 7 'tif' files

Loading images

Done

Searching for 'tif' files in Data/2019-07-25

Found 7 'tif' files

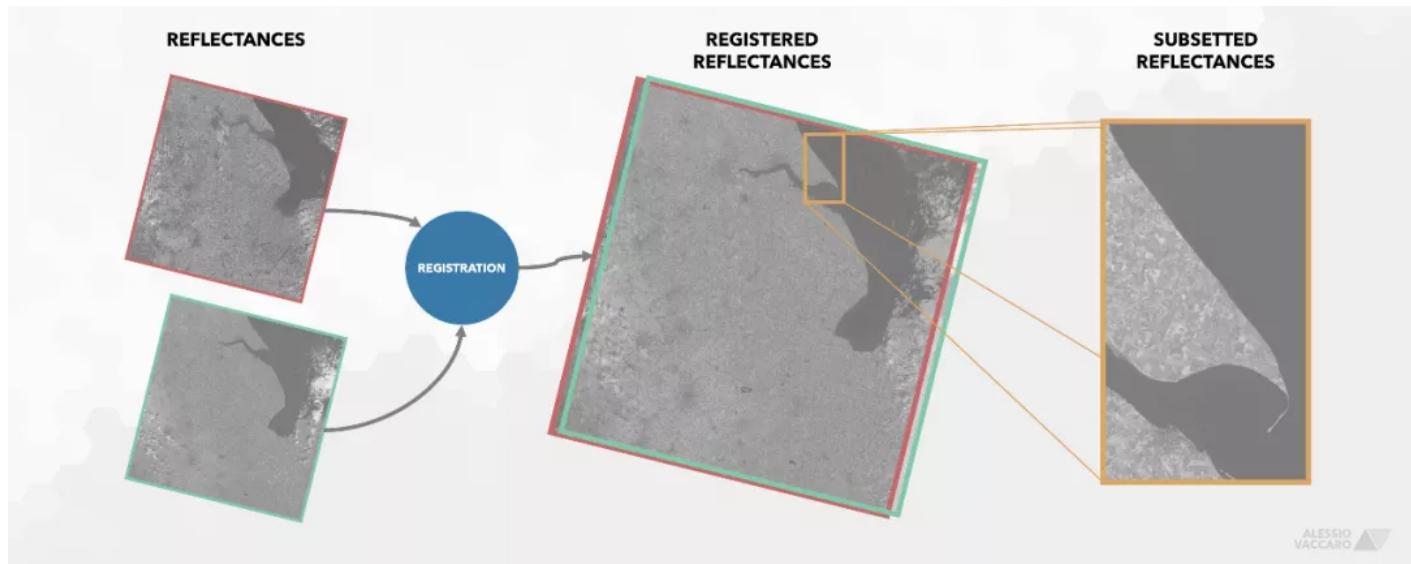
Loading images

Done

现在我们已加载了14张OLI图像（在7个波段中各采集2个）。

2.1 子集多光谱立方体

在这个阶段中，先对**两个多光谱立方体**进行“对齐”（或正式注册），再切出不感兴趣的部分。



我们可以使用***ImageImages ()*** 函数“剪切”不需要的数据。

因此，我们**定义AOI**（感兴趣的区域），并使用***Acquisition ()*** 类中的***subsetImages ()*** 函数进行设置：

```

1 W1, W2 = 950, 2300
2 H1, H2 = 4500, 5300
3
4 subAcquisitions = [acquisition.subsetImages(W1, W2, H1, H2, 552285.0) for acquisition

```

完成!

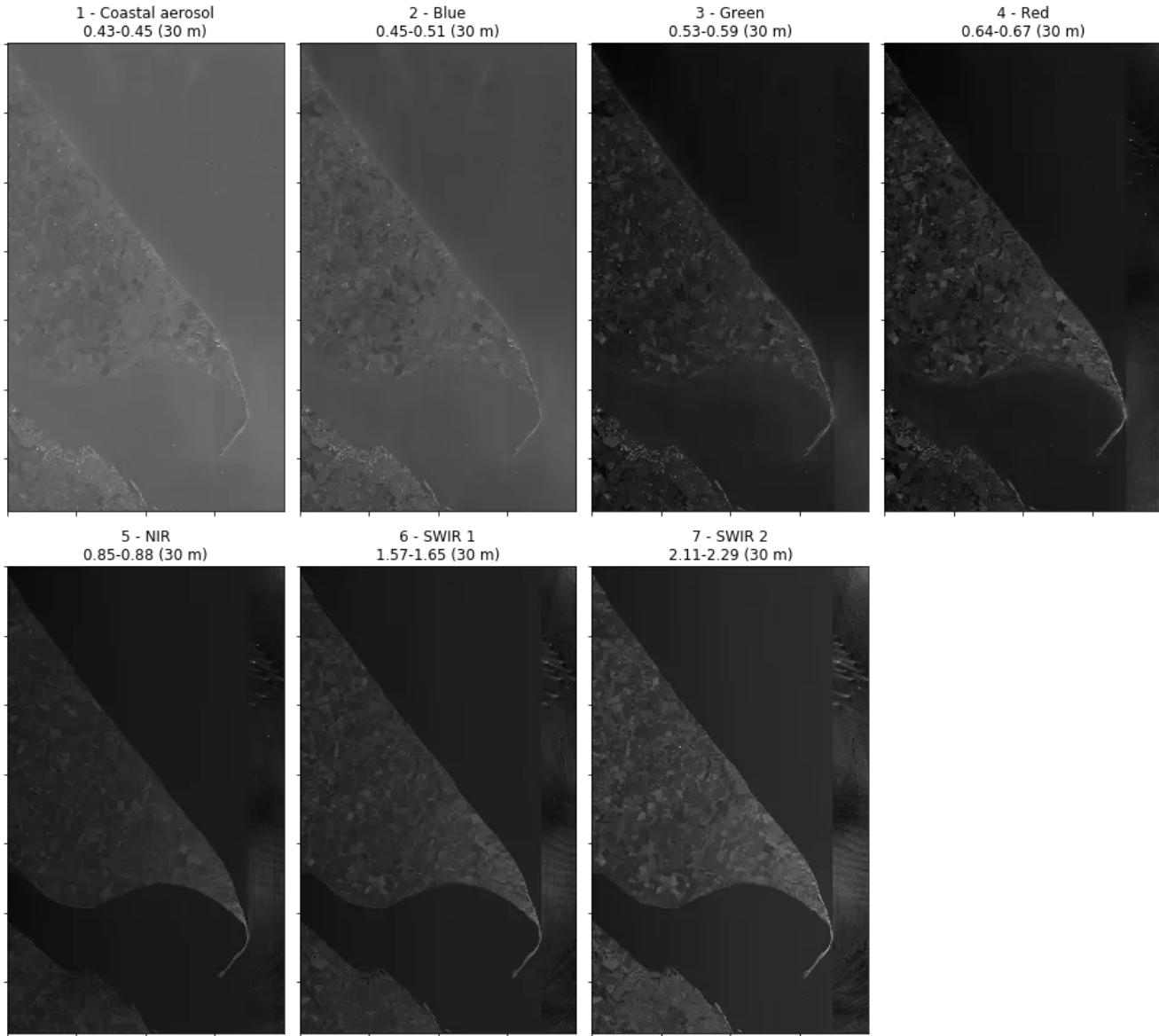
3.数据探索

3.1 可视化多光谱立方体

让我们尝试查看2019/07/25收购的所有范围。**出于纯粹的美学原因**, 在绘制图像之前, 让我们使用 **StandardScaler ()** 对图像进行**标准化**。

```
1  axs = range(N_OPTICS_BANDS)
2  fig, axs = plt.subplots(2, 4, figsize=(15,12))
3  axs = list(itertools.chain.from_iterable(axs))
4
5  for b in range(N_OPTICS_BANDS):
6      id_ = bandsCharacteristics[b]["id"]
7      name_ = bandsCharacteristics[b]["name"]
8      span_ = bandsCharacteristics[b]["span"]
9      resolution_ = bandsCharacteristics[b]["resolution"]
10     title = "%s - %s\n%s (%s m)" % (id_, name_, span_, resolution_)
11     axs[b].imshow(preprocessing.StandardScaler().fit_transform(subAcquisitions[1][b])
12     axs[b].set_title(title); axs[b].set_xticklabels([]); axs[b].set_yticklabels([])
13
14 plt.axis("off"); plt.tight_layout(w_pad=-10); plt.show()
```

以下是运行结果。



这些图中，有些波段比其他波段更亮。这很正常。

3.2 可视化复合RGB中的多光谱立方体

现在，让我们尝试可视化使用波段4（红色），3（绿色）和2（蓝色）获得的**RGB复合图像**中的两次采集。

定义BIAS和GAIN **仅是为了获得更好的效果。**

```

1 BIAS = 1.5
2 GAIN = [2.3,2.4,1.4]
3
4 r1 = (subAcquisitions[0][3] - subAcquisitions[0][3].min()) / (subAcquisitions[0][3].
5 g1 = (subAcquisitions[0][2] - subAcquisitions[0][2].min()) / (subAcquisitions[0][2].
6 b1 = (subAcquisitions[0][1] - subAcquisitions[0][1].min()) / (subAcquisitions[0][1].
7

```

```

8 r2 = (subAcquisitions[1][3] - subAcquisitions[1][3].min()) / (subAcquisitions[1][3].
9 g2 = (subAcquisitions[1][2] - subAcquisitions[1][2].min()) / (subAcquisitions[1][2].
10 b2 = (subAcquisitions[1][1] - subAcquisitions[1][1].min()) / (subAcquisitions[1][1].
11
12 rgbImage1, rgbImage2 = np.zeros((W2-W1,H2-H1,3)), np.zeros((W2-W1,H2-H1,3))
13 rgbImage1[:, :, 0], rgbImage2[:, :, 0] = r1, r2
14 rgbImage1[:, :, 1], rgbImage2[:, :, 1] = g1, g2
15 rgbImage1[:, :, 2], rgbImage2[:, :, 2] = b1, b2
16
17 fig, (ax1,ax2) = plt.subplots(1,2,figsize=(16,12))
18 ax1.imshow(rgbImage1); ax2.imshow(rgbImage2)
19 ax1.set_title("RGB\n(Bands 4-3-2)\n2014-02-01"); ax2.set_title("RGB\n(Bands 4-3-2)\n2019-07-25")
20 plt.show()

```



结果如下图所示！有趣的是，这两次获取的反射率完全不同。



好的，继续进行**海岸线检测**。

4. 自动化海岸线检测

在本段中，我们将使用*Canny的算法*执行边缘检测。

在进行实际检测之前，有必要准备数据集，尝试通过聚类算法对数据集进行分割以区分海洋和陆地。



4.1 数据准备

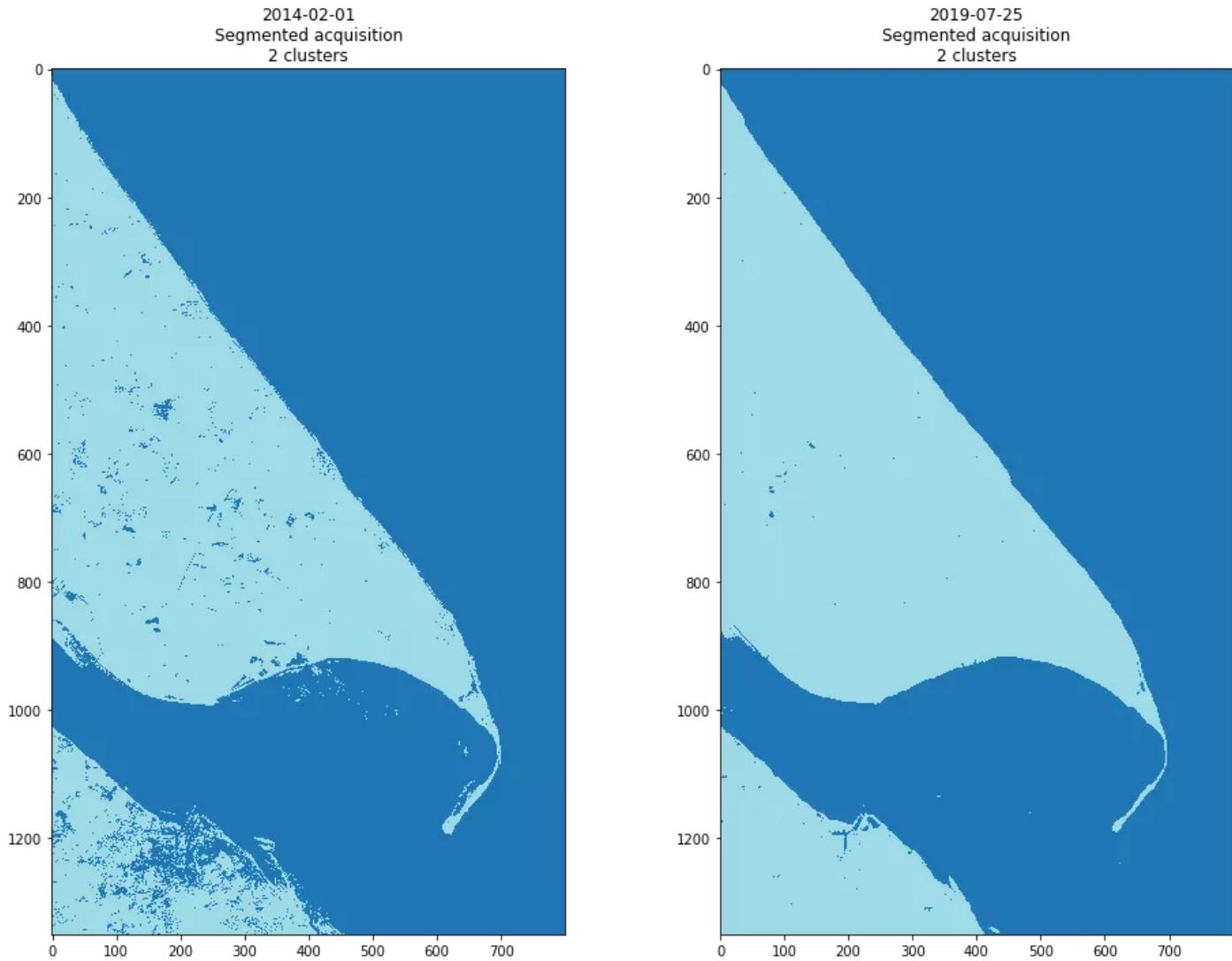
在此阶段，我们将重塑两个多光谱立方体以进行聚类操作。

4.2 用K均值进行图像分割

我们通过k均值对这两次采集进行细分（使用自己喜欢的模型即可）。

4.3 细分结果

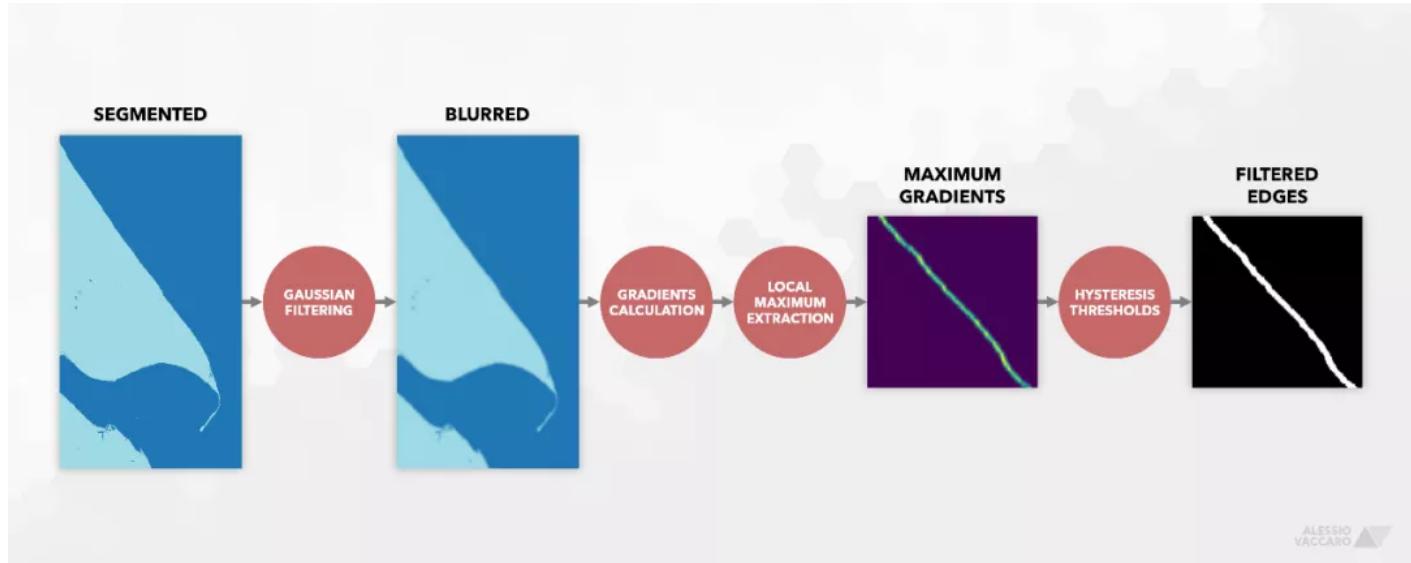
这是确定的代表新兴土地和水体的两个集群。



4.4Canny边缘检测算法

Canny的传统键技术分为以下几个阶段：

1. 高斯滤波器通过卷积**降低噪声**；
2. 四个方向（水平，垂直和2个倾斜）的图像**梯度计算**；
3. **梯度局部最大值的提取**；
4. **带有滞后的阈值**，用于边缘提取。



让我们开始，将聚类结果转换为图像，然后通过具有 15×15 内核的高斯滤波器降低噪声：

```

1 clusteredImages = [clusterLabels.reshape(subAcquisitions[0][0].shape).astype("uint8")
2 blurredImages = [cv.GaussianBlur(clusteredImage, (15,15), 0) for clusteredImage in c
3
4 fig, (ax1, ax2) = plt.subplots(1,2,figsize=(16,13))
5
6 ax1.imshow(blurredImages[0])
7 ax1.set_title("2014-02-01\nGaussian Blurred Image")
8 ax2.imshow(blurredImages[1])
9 ax2.set_title("2019-07-25\nGaussian Blurred Image")
10
11 plt.show()

```

在图像稍微模糊之后，我们可以使用**OpenCV Canny ()** 模块：

```

1 rawEdges = [cv.Canny(blurredImage, 2, 5).astype("float").reshape(clusteredImages[0].s
2
3 edges = []
4 for edge in rawEdges:
5     edge[edge == 0] = np.nan
6     edges.append(edge)

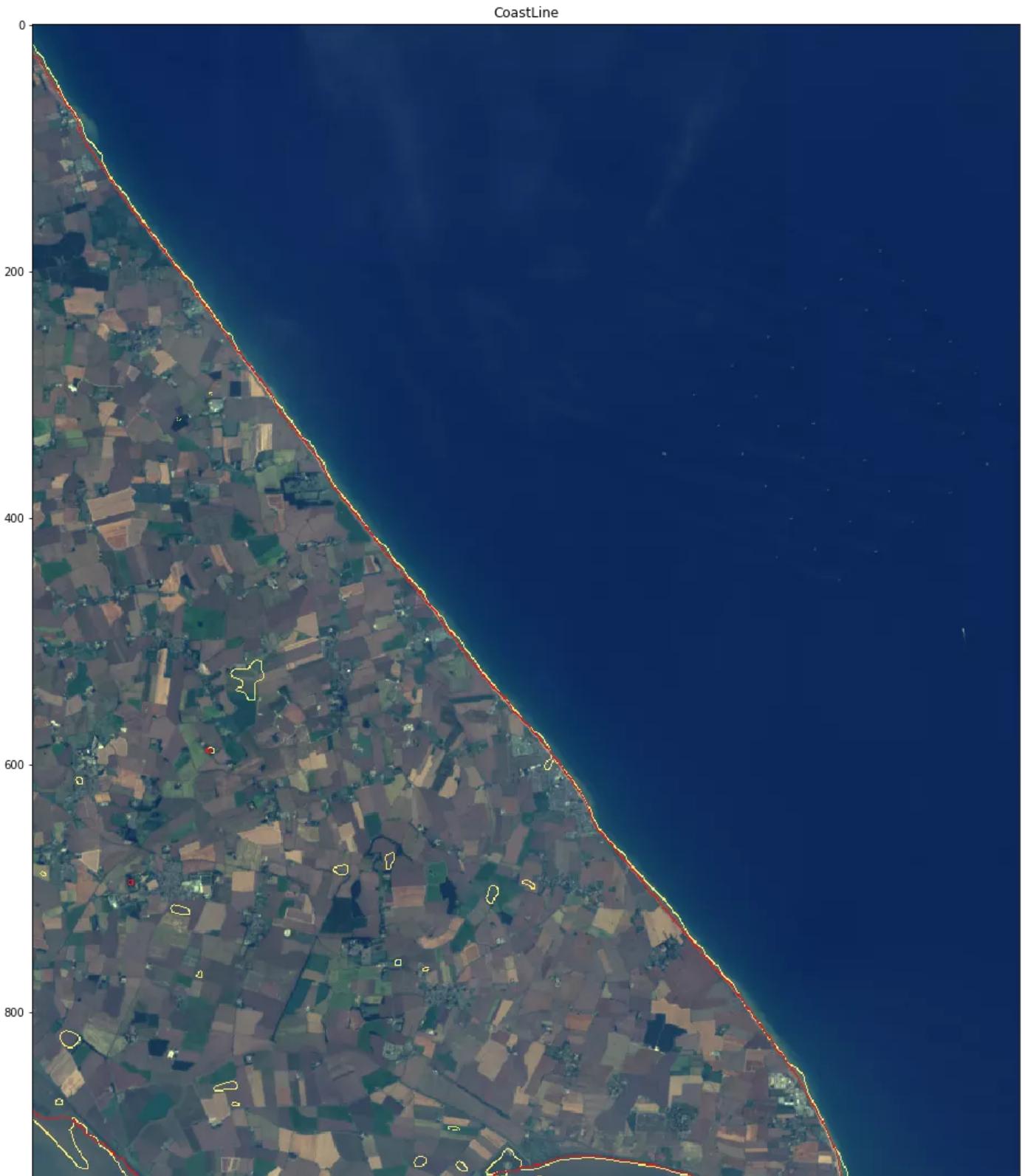
```

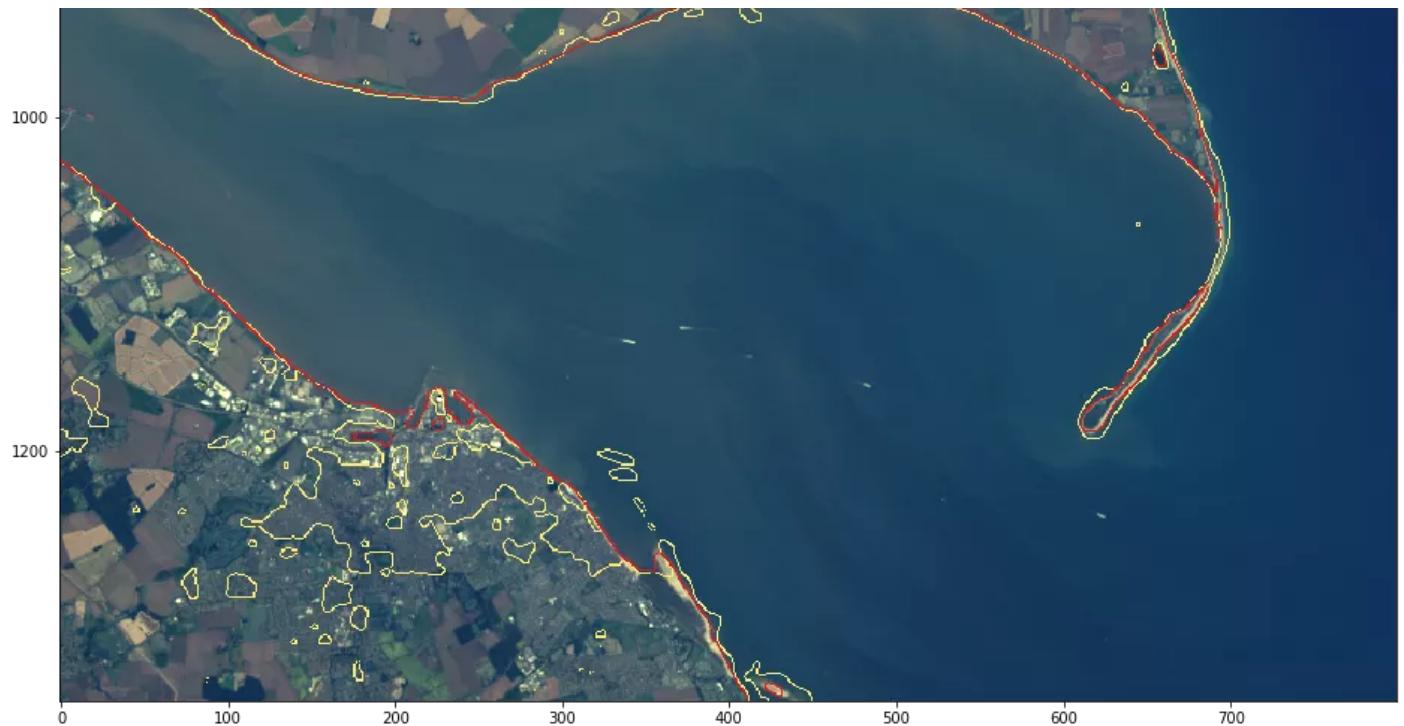
在单行代码中，我们获得了梯度，提取了局部最大值，然后对每次采集都应用了带有滞后的阈值。

注意：我们可以使用不同参数Canny () 来探索处理结果。

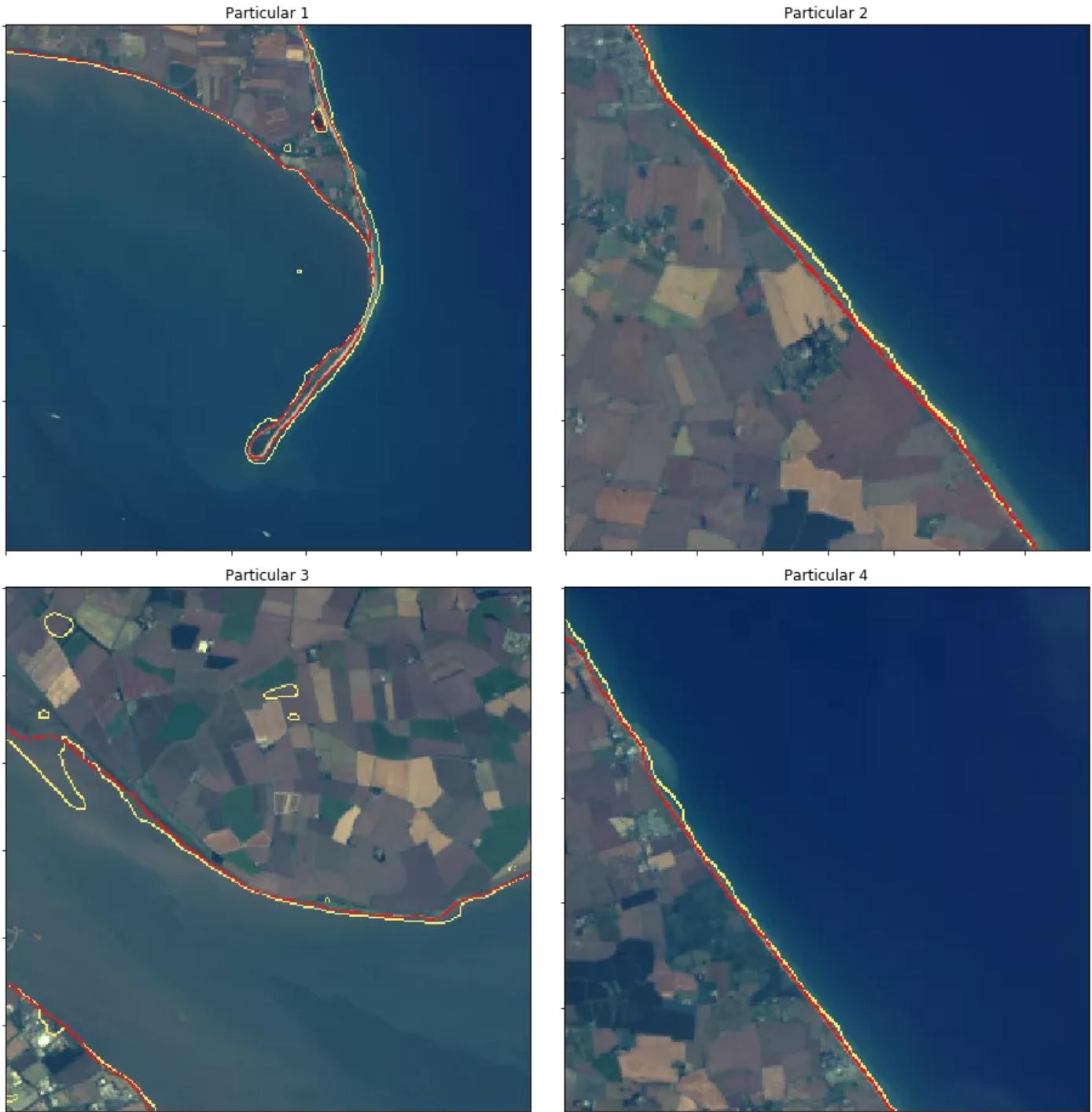
4.5结果

```
1 plt.figure(figsize=(16,30))
2 plt.imshow(rgbImage2)
3 plt.imshow(edges[0], cmap = 'Set3_r')
4 plt.imshow(edges[1], cmap = 'Set1')
5 plt.title('CoastLine')
6 plt.show()
```





以下是一些详细信息：



5 结论

从结果中可以看到，Canny的算法在其原始管道中运行良好，但其性能通常取决于所涉及的数据。

实际上，所使用的聚类算法使我们能够对多光谱立方体进行细分。并行使用多个聚类模型可以总体上改善结果。

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。请按照格式备注，否则不予通过。添加成功后会根据研究方向邀请进入相关微信群。请勿在群内发送广告，否则会请出群，谢谢理解~



 小白学视觉

计算机视觉
论文解读 求职感想
SLAM技术 深度学习 学习感受

距离我们只差一个
长按关注

聚集地
计算机视觉学者



使用OpenCV为视频中美女加上眼线

原创 小白 小白学视觉 昨天

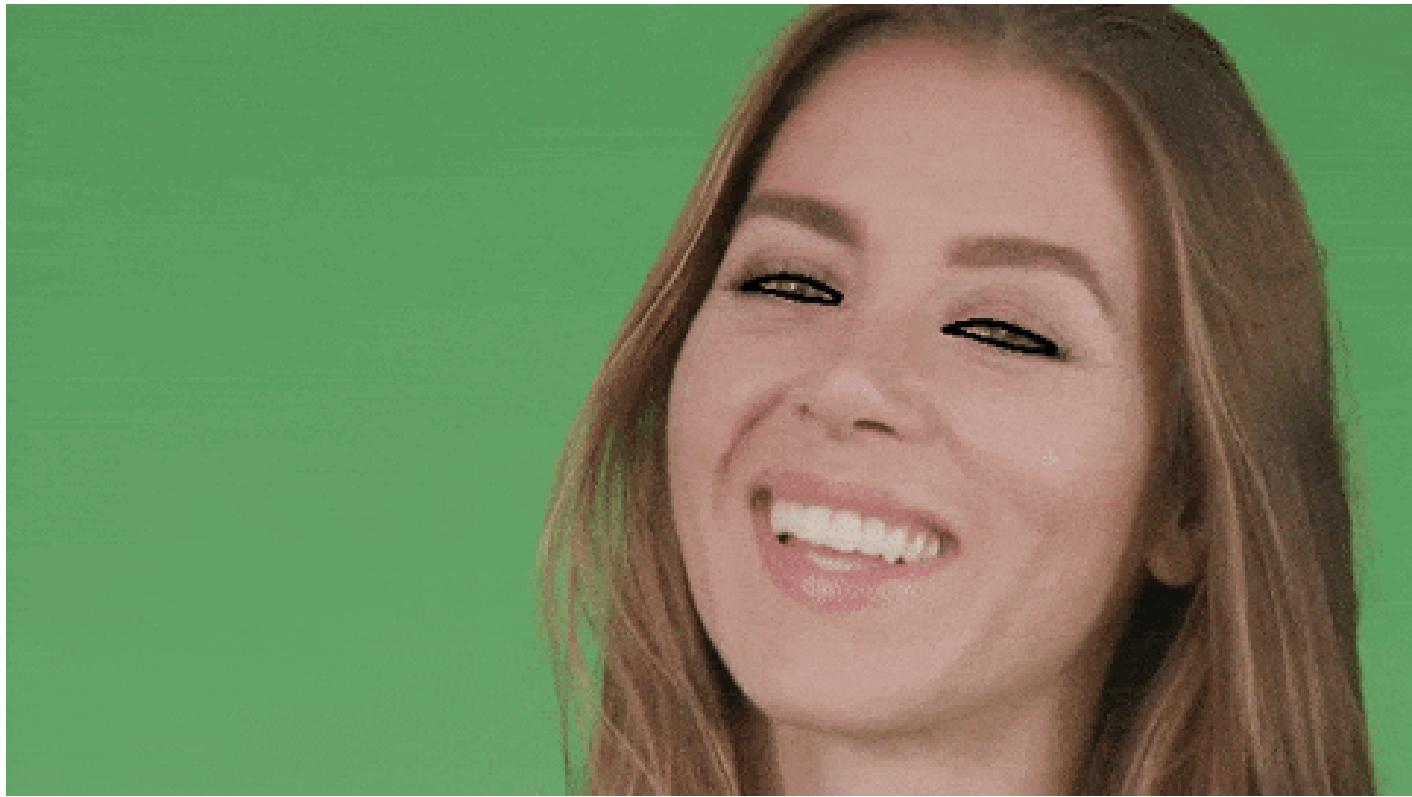
来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

计算机视觉是最令人兴奋的领域之一，其应用范围非常广泛。从医学成像到创建最有趣的面部滤镜等各个领域都充分见证了计算机视觉技术的强大。在本文中，我们将尝试创建一个人造眼线笔来模仿 Snapchat 或 Instagram 滤波器，为视频中的美女添加上美丽的眼线。最终的结果可以通过下面的动图观察到。



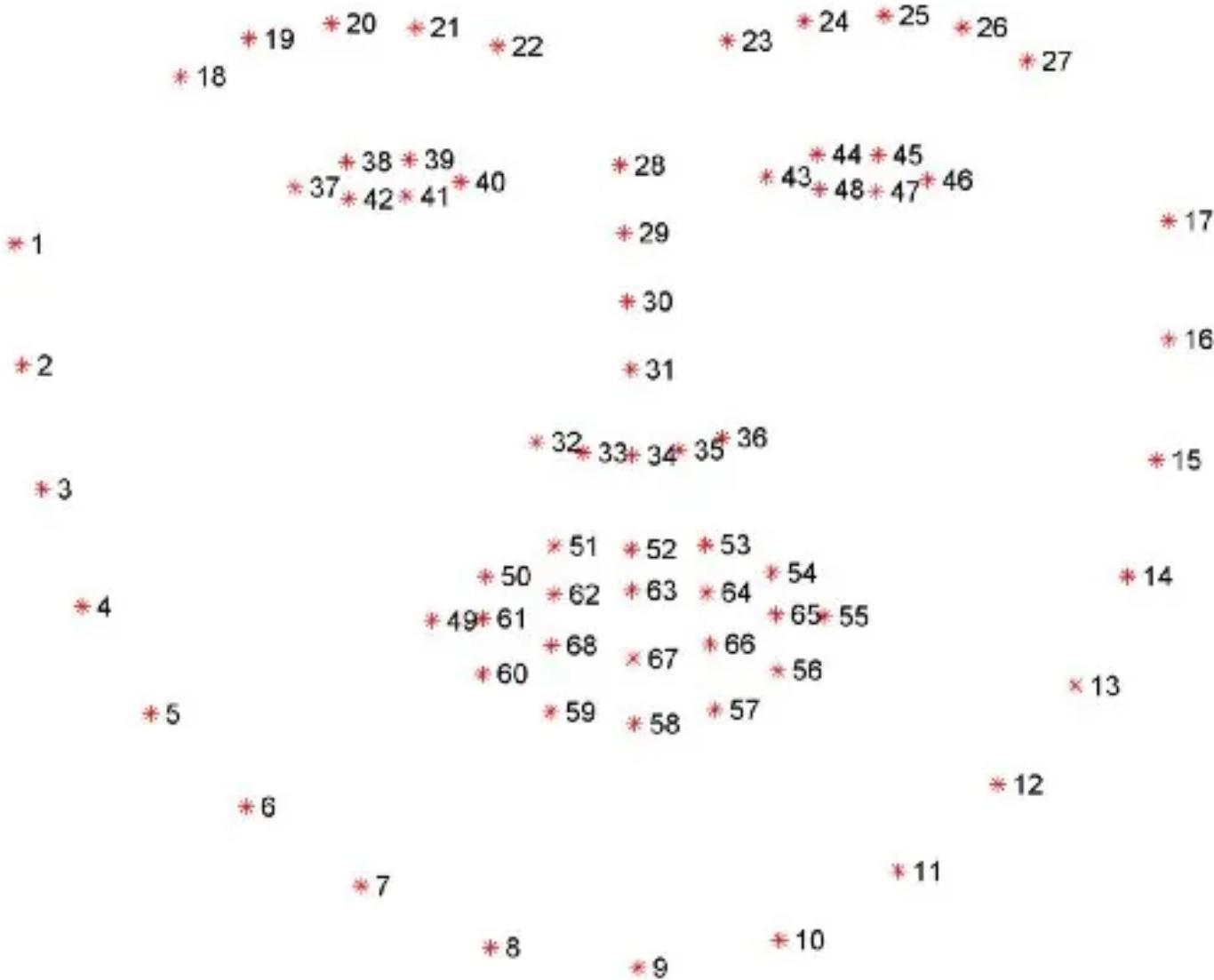
本文介绍的内容适合想要通过计算机视觉来实现一个具有一定展示性功能的计算机视觉初学者。因此，在本文重我们会尽量简化说明，如果您对完整的程序感兴趣，可以在Github上找到完整的代码。Github的链接在本文的文末给出。

在实现本文功能之前，我们需要设置一个新的虚拟环境并安装所有必需的依赖项。这个过程比较简单，我们也在Github里面给出了如何配置环境的具体过程。在本项目中，我们需要使用的工具有 OpenCV, NumPy, imutils, SciPy 和 Dlib。有些小伙伴可能对这些工具和库比较陌生，接下来我们简单介绍一下每个模块的作用。

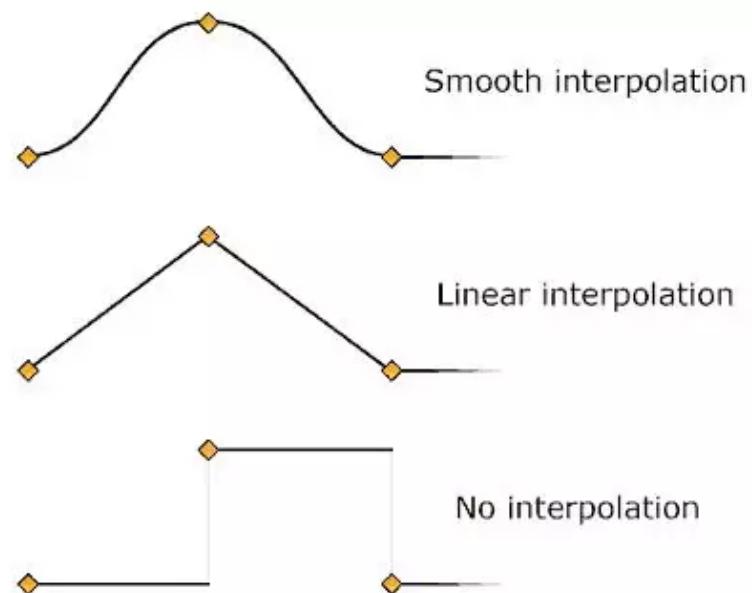
- **OpenCV**: 用于图像处理的最受欢迎的模块之一。我们将使用OpenCV读取，写入和绘制图像。
- **NumPy**: 在处理OpenCV项目时经常使用NumPy。图像本质上是一个像素数组，OpenCV使用以NumPy数组形式存储的这些数组，并对图像执行操作。
- **Imutils**: Imutils附带了自定义功能，使我们的计算机视觉工作变得更加轻松。在这里，我们将使用它来将dlib对象转换为非常灵活且广泛接受的numpy数组。
- **Scipy**: 顾名思义，SciPy用于python上的科学计算。我们将使用它来创建插值（如果现在没有意义，可以的）。
- **Dlib**: Dlib是一个包含各种ML算法的C ++库。我们将使用dlib提取面部界标点。

项目简介介绍

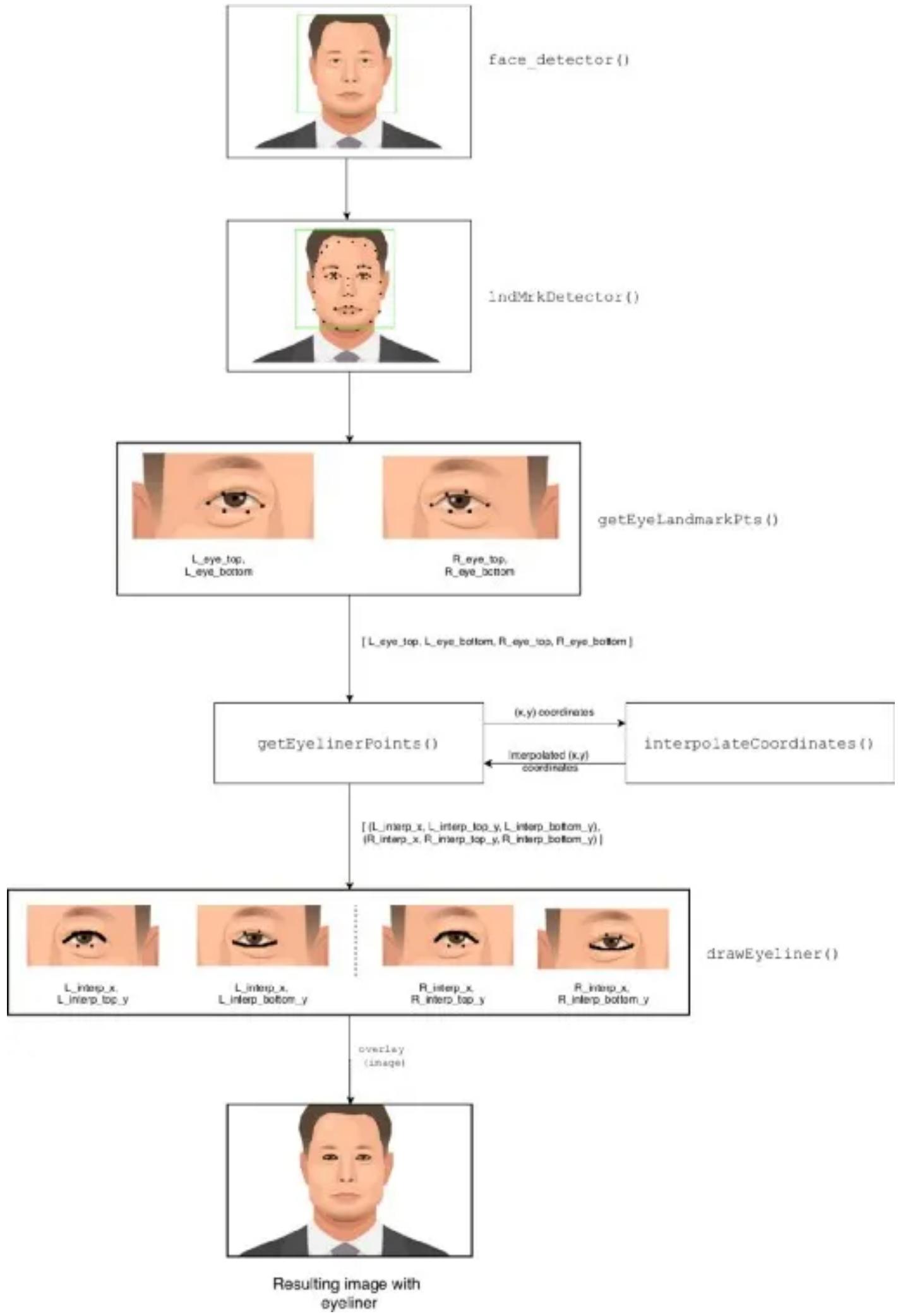
该程序首先从每个面孔中提取68个界标点。在这68个点中，点37–42属于左眼，点43–48属于右眼，具体形式如下图所示。



因为我们的目标是给面部添加眼线，所以我们只对37-48点感兴趣，因此我们提取了这些点。我们将对这些提取的点进行插值。插值意味着我们尝试在两个给定点之间插入点。我们可以使用的插值方式如下图所示。



眼线算法的流程图如下所示



接下来，我们将进一步详细描述该算法。如果小伙伴只对运行代码感兴趣，可以跳至最后一部分。

算法介绍

我们首先需要提取脸部周围边界框的坐标。

OpenCV将图像转换为NumPy数组。`numpy.array`（即图像的矩阵表示形式）存储在名为`frame`的变量中。我们使用一个名为`face_detector()`的函数，该函数返回围绕框架中所有脸部的包围框的坐标。这些边界框坐标存储在一个名为`bounding_boxes`的变量中。遍历循环`bounding_boxes`以将眼线应用于帧中检测到的每个脸部。`face_landmark_points`存储68个坐标点。`eye_landmark_points`是从`getEyeLandmarkPts()`函数中得到。

`getEyeLandmarkPts()`函数使用68个坐标点作为输入并返回具有左上眼睑的坐标4个矩阵，左上眼线（`L_eye_top`），左下眼线（`L_eye_bottom`）和相同的右眼（`R_eye_top & R_eye_bottom`）。这可以通过简单的NumPy索引完成的。我们将端点（pt号37、40、43和46。请参见68个界标点图）向外移动5px，以使外观更逼真。

现在，我们需要对这些点进行插值以获得平滑的曲线，进而可以画出眼线。我们需要对每个曲线进行不同的处理（即`L_eye_top`, `L_eye_bottom`, `R_eye_top`, `R_eye_bottom`）。因此，我们为每个曲线使用单独的变量名称。`interpolateCoordinates()`用于在每条曲线上生成插值。重复使用该函数，为每个曲线生成插值坐标。这个函数为每个曲线返回一个插值点数组。

`drawEyeLiner()`函数将生成的插值点作为参数，并在两个连续点之间画一条线。在两个循环中为每个曲线完成此操作，一个循环用于左眼，另一个循环用于右眼。

调用项目

该项目的用发非常简单，首先从Github上克隆到本地

```
1 git clone https://github.com/kaushil24/Artificial-Eyeliner/
```

接下来，打开命令提示符并键入以下代码以运行示例测试

```
1 python3 eyeliner.py -v "Media/Sample Video.mp4"
```

我们也可以通过将视频路径放在参数中来使用自己的视频。完整的CLI命令如下：

```
1 python eyeliner.py [-i image] [-v video] [-d dat] [-t thickness] [-c color] [-s save]
```

每个参数的具体含义如下：

- i : 要在其上绘制眼线的图像的路径
- v : 要在其上绘制眼线的视频的路径。
- v : 也可以通过网络摄像头获取视频。例如：python3 -v webcam -s "Webcam output"
- t : 整数（整数）以设置眼线的厚度。默认值= 2。推荐的数值介于1-5之间
- d : shape_predictor_68_face_landmarks.dat文件的路径。默认路径在根目录中。除非将shape_predictor_68_face_landmarks.dat文件存储在其他位置，否则不需要使用此参数。
- c : 更改眼线的颜色。语法-c 255 255 255。默认值= 0 0 0。其中每个数字代表其RGB值。
- s : 要将输出保存到的位置和文件名。注意程序在保存文件时会自动添加扩展名。如果已经存在同名文件，它将覆盖该文件。

好了，对这个项目感兴趣的小伙伴可以按照上面的说明来进行尝试，通过对程序的修改以达到自己的需求。如果小伙伴觉得这个项目比较有趣，文末给小白六留个“好看”哦。

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过**。添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





小白学视觉

计算机视觉
论文解读 求职感想
SLAM技术 深度学习 学习感受

距离我们只差一个
长按关注

聚集地
计算机视觉学者



用OpenCV实现猜词游戏

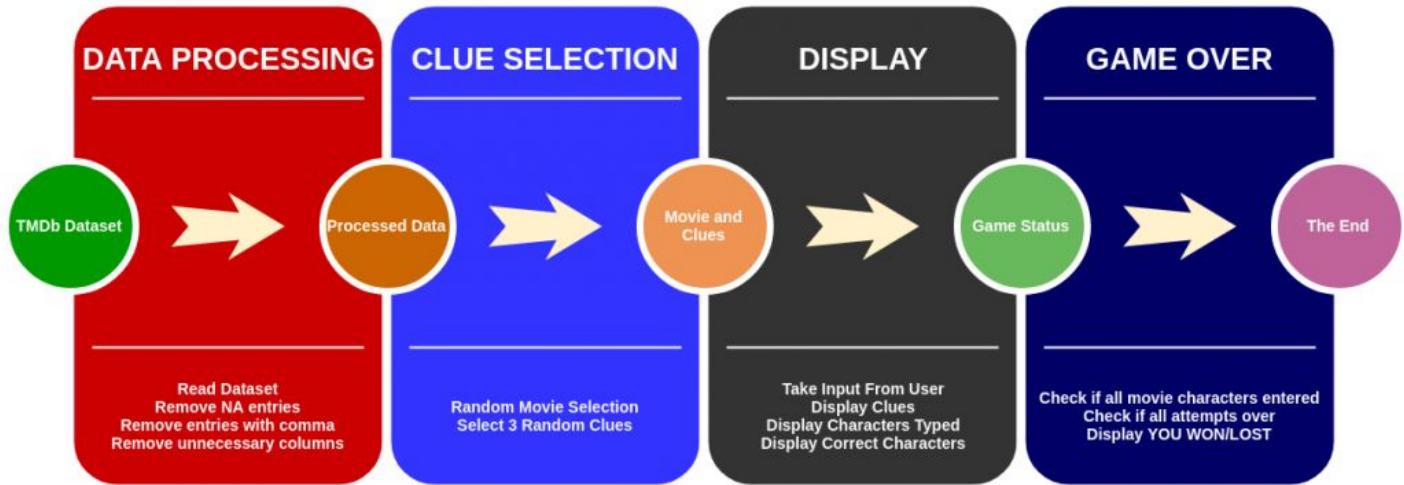
原创 小白 小白学视觉 2019-07-01

小伙伴们是不是在用OpenCV来处理图像处理的相关任务，从来没有想过还可以通过OpenCV设计一款游戏，今天小白将为各位小伙伴们介绍如何通过OpenCV创建一个猜词的小游戏。为了增加趣味性，我们给小游戏起了一个比较具有故事性的名字“刽子手游戏（Hangman）”，我们先来看一下该游戏的视频。

00:34

这是一个猜电影名字的游戏，会在屏幕下方显示电影的单词数目以及每个单词的字母个数，我们需要猜电影名字中含有的字母，如果猜测错误，右侧的刽子手处就会依次出现人头、身体、手和脚等，当猜错6次之后，刽子手就会行动，游戏就输了。但是为了增加获胜几率，在出现错误的时候会给出关于电影的部分提示，当把电影名字全部猜出后，我们就取得了胜利。

接下来将介绍如何实现这个有趣的小游戏，我们将本小游戏实现分成4个子功能模块，其构成如下图所示。



数据处理

为了确保游戏不会很快耗尽电影和线索，我们选择TMDb数据集（电影数据库），数据处理步骤包括删除却是值，仅保留所需的内容-发行年份、演员表、导演、电影关键字和标语。同时为了减少数据集大小，只保留标题不超过20个字符且标语不超过30的电影。

电影和线索选择

首先我们从CSV文件加载数据集并以字典格式存储它

```

def read_from_csv(csv_f):
    with open(csv_f, 'r') as f:
        movie_data = {}
        for line in f.readlines():
            line_split = line.strip().split(',')
            year = line_split[-1].split('|')
            keywords = line_split[-2].split('|')
            tagline = line_split[-3].split('|')
            director = line_split[-4].split('|')
            cast = line_split[-5].split('|')
            movie = line_split[0].upper()
            movie_data[movie] = [year, keywords, tagline, director, cast]
    return movie_data
    
```

请注意，我们使用以下字典格式存储电影和电影详细信息。

```
movie_title:[year, list of keywords, tagline, director, list of cast]
```

接下来，我们将从电影列表中获取一部随机电影并获取该电影的信息（年份，关键字，标语，导演和演员）

```

def get_movie_info(movies_data):
    movies_list = list(movies_data.keys())
    movie = np.random.choice(movies_list, 1)[0].upper()
    movie_info = movies_data[movie]
    return movie, movie_info
    
```

我们将使用另一个函数select_hints来选择任意3个随机提示。如果关键字或强制转换作为提示存在，我们将从它们中随机选择一个元素

```
def select_hints(movie_info):
    # We will randomly select 3 types of
    # hints to display
    hints_index = list(np.random.choice(5, 3, replace=False))
    hints = []
    hints_labels = ["Release Year", "Keyword", "Tagline", "Director", "Cast"]
    labels = []
    for hint_index in hints_index:
        hint = np.random.choice(movie_info[hint_index], 1)[0].upper()
        hints.append(hint)
        labels.append(hints_labels[hint_index].upper())
    return hints, labels
```

编程显示

加载Hangman Canvas

首先，我们显示下面的hangman模板



```
def get_canvas(canvas_file):
    img = cv2.imread(canvas_file, 1)
    return img
```

获得角色尺寸

接下来，这里有一个棘手的部分。我们想要显示空白框，以便用户知道电影标题的长度。考虑到角色的宽度和高度，我们将使它更有趣和更通用。我们将使用该标题中所有字符的最大高度和宽度。我们将使用函数get_char_coords来计算这些框的坐标。

```
def get_char_coords(movie):
    x_coord = 100
    y_coord = 400

    char_ws = []
    char_hs = []

    for i in movie:
        char_width, char_height = cv2.getTextSize(i, \
            cv2.FONT_HERSHEY_SIMPLEX, 1, 2)[0]
        char_ws.append(char_width)
        char_hs.append(char_height)

    max_char_h = max(char_hs)
    max_char_w = max(char_ws)

    char_rects = []

    for i in range(len(char_ws)):
        rect_coord = [(x_coord, y_coord-max_char_h), \
            (x_coord+max_char_w, y_coord)]
        char_rects.append(rect_coord)
        x_coord = x_coord + max_char_w
    return char_rects
```

画空白框

一旦我们有这些坐标，我们将使用它们来绘制框。

```
def draw_blank_rects(movie, char_rects, img):
    for i in range(len(char_rects)):
        top_left, bottom_right = char_rects[i]
        if not movie[i].isalpha() or \
            ord(movie[i]) < 65 or \
            ord(movie[i]) > 122 or \
            (ord(movie[i]) > 90 and \
            ord(movie[i]) < 97):
            cv2.putText(img, movie[i], (top_left[0], bottom_right[1]), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
            continue
        cv2.rectangle(img, top_left, bottom_right, (0, 0, 255), thickness=1, lineType = cv2.LINE_8)
    return img
```

显示提示

要显示的提示取决于不正确尝试的次数。如果没有不正确的尝试，我们将不会显示任何提示。如果一次尝试不正确，我们将显示第一个提示。类似地，对于少于4次不正确的尝试，我们将显示第二个提示，最后，对于少于7个不正确的尝试，我们将显示第三个提示。

```
def draw_hint(img, hints, labels, incorrect_attempts):
    x, y = 20, 30
    if incorrect_attempts == 0:
        return img
    elif incorrect_attempts <= 1:
        index = 0
    elif incorrect_attempts <= 3:
        index = 1
    elif incorrect_attempts <= 6:
        index = 2
    cv2.putText(img, "HINT: {}".format(labels[index]), (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 255))
    cv2.putText(img, "{}".format(hints[index]), (x, y+30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 255))
    return img
```

显示正确或不正确的尝试

如果电影标题中出现猜测的字母，需要进行提示。

```
def draw_wrong(img, incorrect_attempts):
    cv2.putText(img, "WRONG {}/6".format(incorrect_attempts+1), (380, 40), \
    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
    return img

def draw_right(img):
    cv2.putText(img, "RIGHT", (380, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
    return img
```

显示游戏输赢

如果游戏赢了或输了，同样需要进行提示

```
def draw_lost(img):
    cv2.putText(img, "YOU LOST", (380, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    return img

def draw_won(img):
    cv2.putText(img, "YOU WON", (380, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
    return img
```

显示无效的移动和字符重用

现在，我们还有两个案例需要考虑。如果输入了无效字符怎么办？这可以是数字或非字母数字字符。

第二，如果用户输入之前已输入的角色，该怎么办

```

def draw_invalid(img):
    cv2.putText(img, "INVALID INPUT", (300, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    return img

def draw_reuse(img):
    cv2.putText(img, "ALREADY USED", (300, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    return img

```

显示使用的字符

现在，为了让用户更容易知道已经使用了哪些字符，我们也将显示它们。我们将在这里再添一个。我们将显示以红色输入的有效字符，以便用户可以看到他们输入的字符。

```

def draw_used_chars(img, chars_entered, letter):
    cv2.putText(img, "Letters used:", (300, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    y = 120
    x = 350
    count = 0
    for i in chars_entered:
        if count == 10:
            x += 50
            y = 120
        if i==letter:
            cv2.putText(img, i, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
        else:
            cv2.putText(img, i, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
        y += 20
        count += 1
    return img

```

显示刽子手身体

接下来轮到编写函数来绘制刽子手的不同身体部位 - 面部，背部，左右手臂，左右腿。将绘制什么身体部位取决于不正确尝试的次数

```

def draw_hangman(img, num_tries):
    if num_tries==1:
        return draw_circle(img)
    elif num_tries==2:
        return draw_back(img)
    elif num_tries==3:
        return draw_left_hand(img)
    elif num_tries==4:
        return draw_right_hand(img)
    elif num_tries==5:
        return draw_left_leg(img)
    elif num_tries==6:
        return draw_right_leg(img)

```

```

else:
    return img

def draw_circle(img):
    cv2.circle(img, (190, 160), 40, (0, 0, 0), thickness=2, lineType=cv2.LINE_AA)
    return img

def draw_back(img):
    cv2.line(img, (190, 200), (190, 320), (0, 0, 0), thickness=2, lineType=cv2.LINE_AA)
    return img

def draw_left_hand(img):
    cv2.line(img, (190, 240), (130, 200), (0, 0, 0), thickness=2, lineType=cv2.LINE_AA)
    return img

def draw_right_hand(img):
    cv2.line(img, (190, 240), (250, 200), (0, 0, 0), thickness=2, lineType=cv2.LINE_AA)
    return img

def draw_left_leg(img):
    cv2.line(img, (190, 320), (130, 360), (0, 0, 0), thickness=2, lineType=cv2.LINE_AA)
    return img

def draw_right_leg(img):
    cv2.line(img, (190, 320), (250, 360), (0, 0, 0), thickness=2, lineType=cv2.LINE_AA)
    return img

```

显示字符出现次数

我们还希望显示在电影标题中输入的所有字母（如果它出现在电影标题中）。

```

def displayLetter(img, letter, movie, char_rects):
    for i in range(len(movie)):
        if movie[i]==letter:
            top_left, bottom_right = char_rects[i]
            cv2.putText(img, movie[i], (top_left[0],bottom_right[1]), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))
    return img

```

显示电影标题

最后，我们想在游戏结束后揭示正确的电影片名。

```

def revealMovie(movie, img, char_rects):
#img = cv2.imread(canvas_file, 1)
    for i in range(len(movie)):
        top_left, bottom_right = char_rects[i]
        cv2.putText(img, movie[i], (top_left[0],bottom_right[1]), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0))
    return img

```

编程游戏

现在我们已经涵盖了所有功能，让我们看看我们将如何在游戏中使用它们。我们将从读取CSV文件中的数据并获取随机电影开始。

```
import cv2
import numpy as np
from utils import *

movie_csv = "movies-list-short.csv"
canvas = "blank-canvas.png"

movies_data = read_from_csv(movie_csv)

movie, movie_info = get_movie_info(movies_data)
```

我们还将选择之前讨论的3个随机提示。

```
hints, labels = select_hints(movie_info)
```

现在，让我们从空白的Hangman画布开始，零尝试不正确。

```
char_rects = get_char_coords(movie)

img = draw_blank_rects(movie, char_rects, img)

cv2.namedWindow("Hangman", cv2.WND_PROP_FULLSCREEN)
cv2.setWindowProperty("Hangman", cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)

cv2.imshow("Hangman", img)

chars_entered = []

incorrect_attempts = 0

img_copy = img.copy()
```

接下来，我们将对该部分进行编码，以便从用户那里获取输入并显示输入的字母。我们还需要显示尝试是正确还是错误，或者是否无效或已经使用过。如果用户用完了尝试，循环将中断。

我们通过以下方式实现上述目标。

1. 创建当前图像的副本。这是为了确保我们不会覆盖诸如错误，正确等字样或提示。
2. 接下来，根据不正确的尝试次数，我们将在图像上显示提示。
3. 如果用户已经用完了所有的生命，我们将显示您丢失并且循环将中断。

4. 如果用户设法猜出电影的所有字符，我们将显示你WON并打破循环。
5. 要检查用户输入的字符是否有效，我们将检查字符是否位于az或AZ之间。如果移动无效，我们将显示相应的消息 - INVALID MOVE，游戏将继续。
6. 将检查用户输入的有效字符以查看它之前是否已被使用过，在这种情况下将显示相应的消息并且游戏将继续。
7. 请注意，在最后两个步骤中，不会更改不正确的尝试次数。
8. 如果输入的字符是新字符，我们将首先将其附加到所用字符列表中，然后检查它是否出现在电影标题中，在这种情况下，我们将显示CORRECT并显示电影中所有出现的字符。
9. 如果在电影标题中找不到该字符，我们将显示错误并增加错误尝试次数。
10. 最后，一旦游戏获胜或失败，我们将揭示正确的电影标题。

```

while 1:
    img = img_copy.copy()
    img = draw_hint(img, hints, labels, incorrect_attempts)
    if incorrect_attempts >= 6:
        img = draw_lost(img)
        break
    elif check_all_chars_found(movie, chars_entered):
        img = draw_won(img)
        break
    else:
        letter = cv2.waitKey(0) & 0xFF
        if letter < 65 or letter > 122 or (letter > 90 and letter < 97):
            img = draw_invalid(img)
            cv2.imshow("Hangman", img)
            continue
        else:
            letter = chr(letter).upper()
            if letter in chars_entered:
                img = draw_reuse(img)
                img = draw_used_chars(img, chars_entered, letter)
                cv2.imshow("Hangman", img)
                continue
            else:
                chars_entered.append(letter)
                if letter in movie:
                    img = draw_right(img)
                    img = displayLetter(img, letter, movie, char_rects)
                    img_copy = displayLetter(img_copy, letter, movie, char_rects)
                else:
                    img = draw_wrong(img, incorrect_attempts)
                    incorrect_attempts += 1
                img = draw_used_chars(img, chars_entered, letter)
                img = draw_hangman(img, incorrect_attempts)
                img_copy = draw_used_chars(img_copy, chars_entered, letter)
                img_copy = draw_hangman(img_copy, incorrect_attempts)

```

```

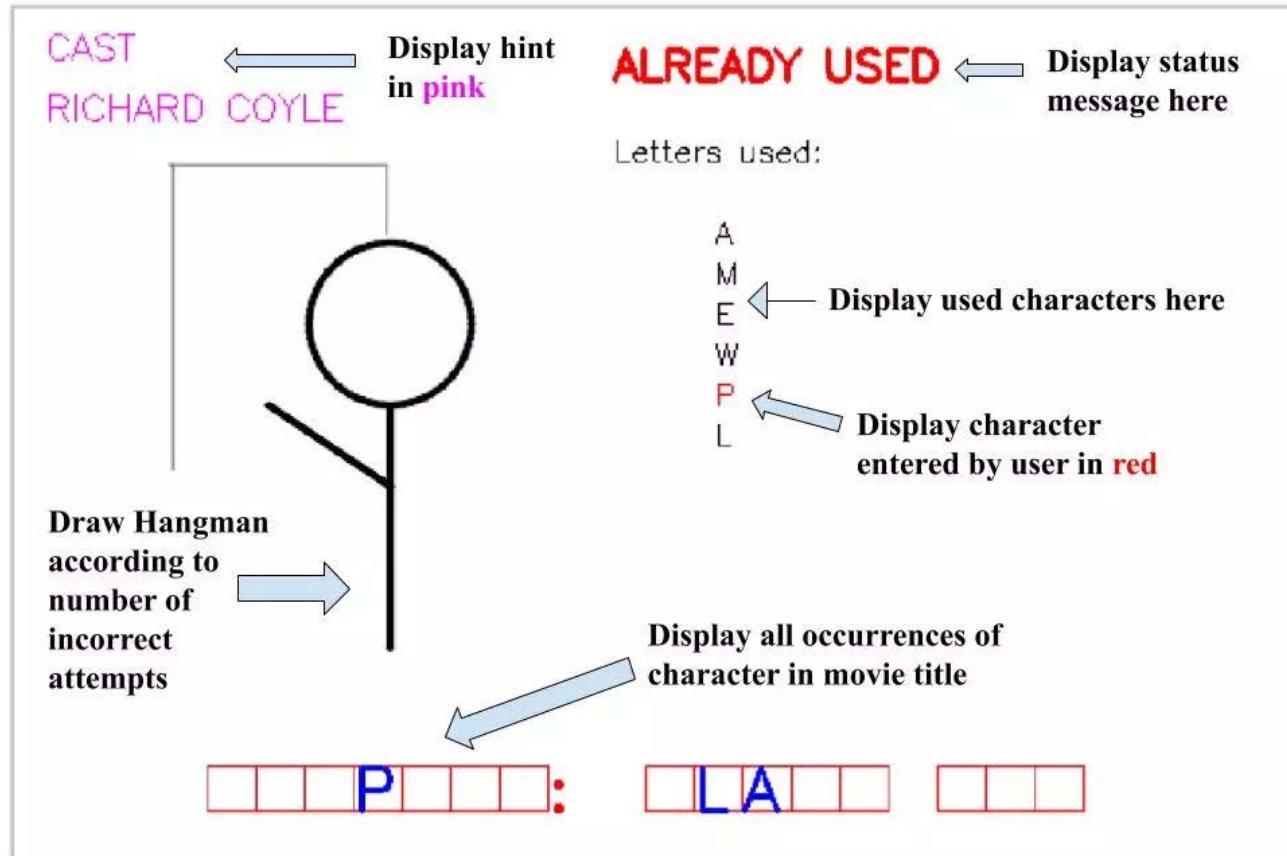
cv2.imshow("Hangman", img)

img = revealMovie(movie, img, char_rects)
cv2.imshow("Hangman", img)
cv2.waitKey(0)

cv2.destroyAllWindows()

```

最后，让我们借助下面显示的图像快速总结一下Hangman游戏的不同部分。



往期文章一览

- 1、人脸识别中的活体检测算法综述
- 2、CVPR2019：PizzaGAN通过深度学习制作披萨
- 3、漫话：如何给女朋友解释为什么计算机只认识0和1？
- 4、10个不得不知道的Python图像处理工具，非常全了！
- 5、OpenCV4.0实现人脸识别
- 6、基于内容的图像检索技术综述-传统经典方法
- 7、为什么不建议你入门计算机视觉
- 8、机器视觉检测系统中这些参数你都知道么？

加群交流

扫码添加助手，可申请加入**OpenCV与数字图像处理交流群**。**一定要备注：图像处理+地点+学校/公司+昵称（如目标检测+上海+上交+卡卡西）**，不根据格式申请，一律不通过。



//
//

使用OpenCV检测坑洼

原创 花生 小白学视觉 6月14日

来自专辑

OpenCV应用

点击上方“**小白学视觉**”，选择“**星标**
重磅干货，第一时间送达

本文将向大家介绍如何使用OpenCV库进行坑洼检测。

为什么要检测坑洼？

坑洼是道路的结构性指标，事先发现坑洼地可以延长高速公路的使用寿命，防止事故的发生，同时降低死亡率。

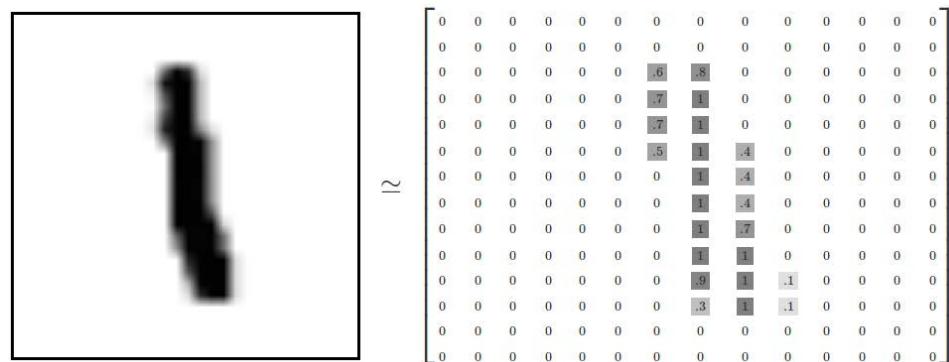
一种可行的解决方案是构建自动坑洞检测系统，该系统可通过云服务发送实时信息以提醒管理结构，来杜绝每天人工检查所产生的不必要花费。

OpenCV是一个帮助研究人员处理图像问题的库，该库提供了大量处理图像的方法。OpenCV的使用将有助于坑洼检测。

图像的基础知识

在了解代码之前，必须先了解图像的工作原理。

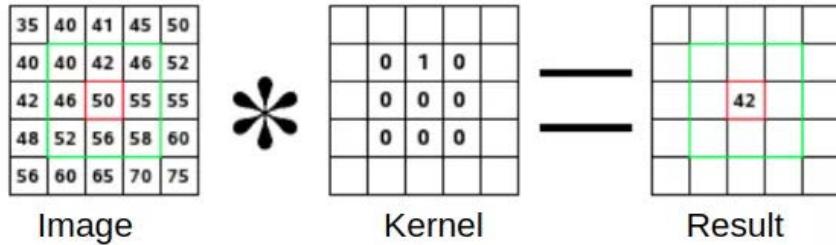
图像一般被划分为很多像素，每个像素的值范围介于 0 和 255 之间。转换为灰度时，范围从 0 到 1。



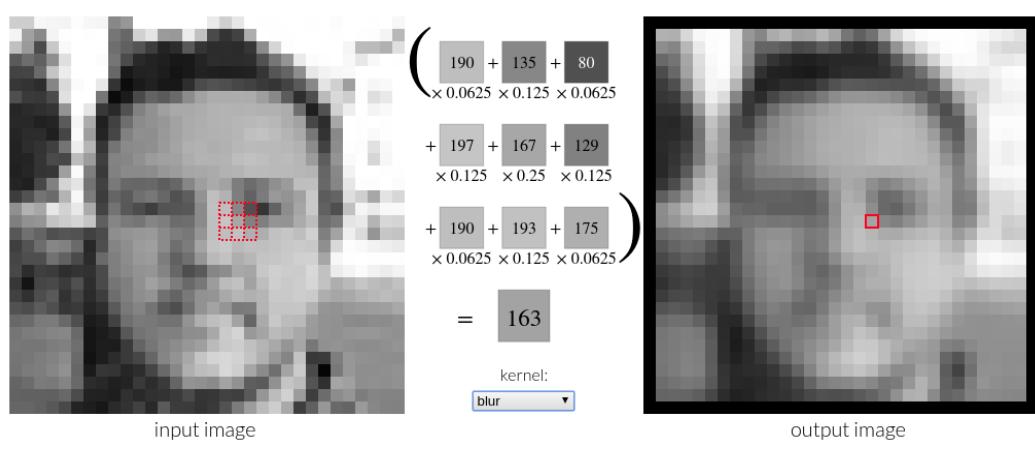
大小为28x28的灰度图像

可以操作图像的每个像素。例如，如果希望随机像素具有另一个值，则有两种方法。第一种是通过直接更改矩阵中的点来更改这一点。第二种是使用内核东西来实现。

内核是具有一定值的小矩阵，通常为 3x3，叠加在图像上充当滤波器。



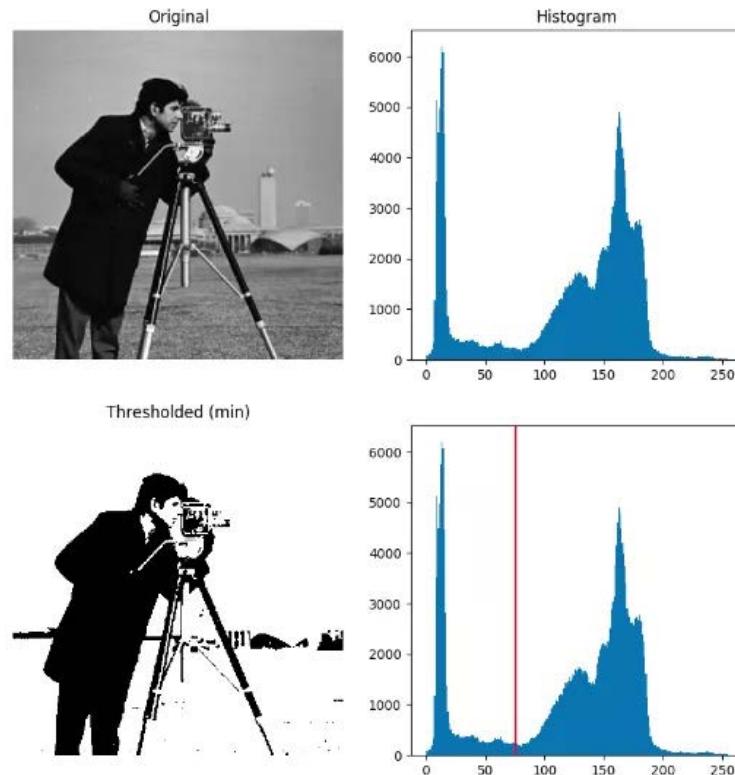
上图显示了图像与内核卷积的结果。**卷积**是通过数值的相乘相加得到输出结果的过程。卷积可以实现图像的模糊，例如下图所示。



所选内核对输入图像进行了模糊



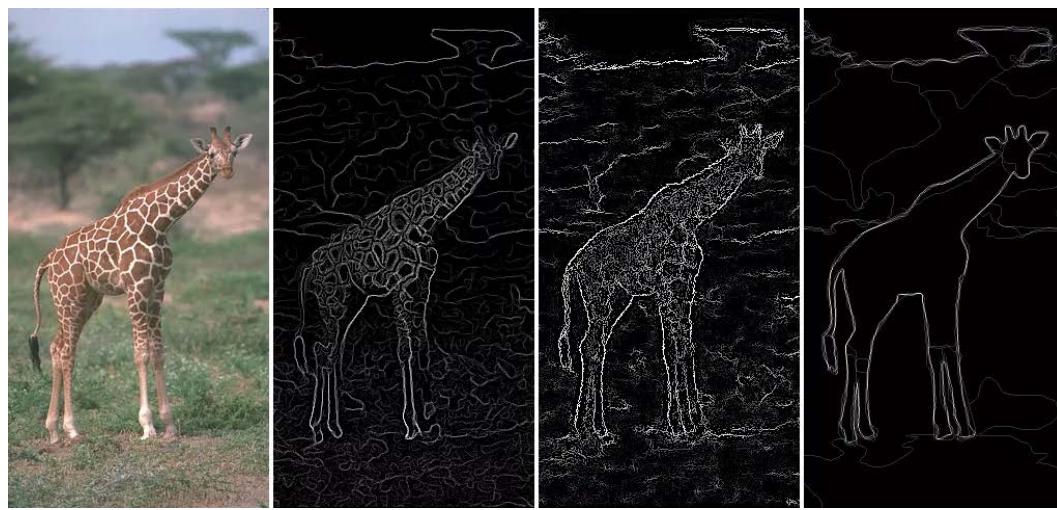
阈值的概念很简单，给定一个图像，绘制其直方图并选择一个值。比该值大的每个像素都将变为黑色，比该值小的每个像素将变为白色，具体如下所示。



根据照明选择不同阈值的自适应阈值方法（这一方法可用于检测坑洞）。更多算法可以在OpenCV阈值文档中找到。

边缘检测

边缘检测算法将在图像中找到边缘。Canny是一种边缘检测算法，它将检测图像的边缘，并输出仅具有轮廓的图像。进一步的解释可以在这里找到。



使用不同参数应用的 Canny 图像

坑洼检测

我们可以将前面介绍的内核+阈值+边缘检测结合起来，并在道路上找到坑洼。

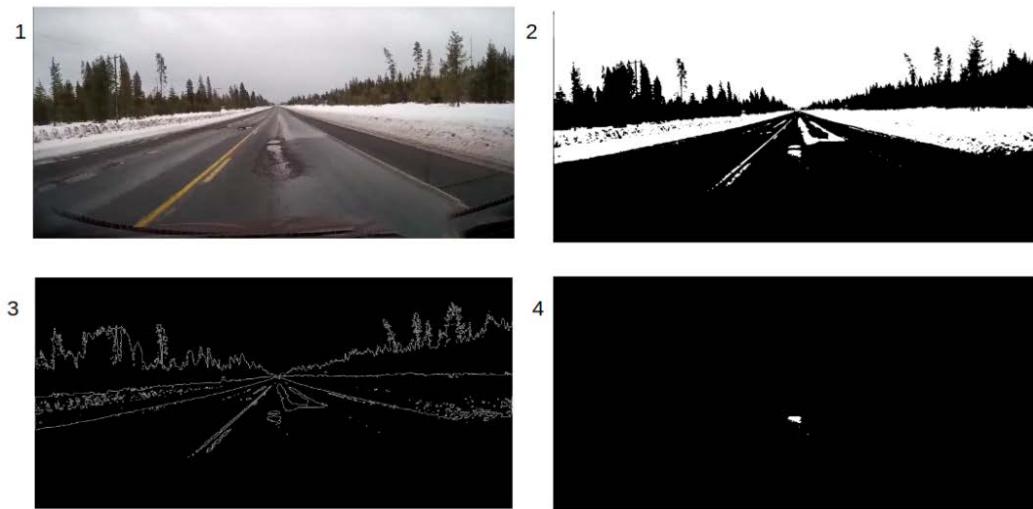


图1显示了从道路上拍摄的图像，该道路的坑洼直接位于汽车前。

图2显示了应用了阈值处理的图像，坑洼和清洁街道区域被突出显示。

Canny应用于图片3，其中可以找到轮廓。在这里，可以创建一个算法，以便查看轮廓是否为坑洞。

图4显示了选中坑洞的图像。



最终图像，带有绿色标记的区域为坑洞的位置。

更多坑洼检测的结果如下图所示。

a.



b.



c.



d.



使用OpenCV进行坑洞检测并不难。此外，我们可以构建检测系统并将其与云和地图服务结合，以便提供有关选定区域坑洞的实时信息。



//
//

使用OpenCV实现车道线检测

原创 小林 小白学视觉 6月18日

来自专辑

OpenCV应用

点击上方“**小白学视觉**”，选择“**星标**”公众号

重磅干货，第一时间送达



采集

图0 印度泰米尔纳德邦安纳马莱森林公路上的车道检测

本文源码：<https://github.com/KushalBKusram/AdvancedLaneDetection>

计算机视觉在自动化系统观测环境、预测该系统控制器输入值等方面起着至关重要的作用。本文介绍了使用计算机视觉技术进行车道检测的过程，并引导我们完成识别车道区域、计算道路RoC和估计车道中心距离的步骤。

摄像机校准 (calibrateCamera.py)

几乎所有摄像机使用的镜头在聚焦光线以捕捉图像时都存在一定的误差，因为这些光线由于折射在镜头边缘发生了弯曲。这种现象会导致图像边缘的扭曲。以下视频用示例解释了两种主要的失真类型，强烈建议观看。

02:50

假设我们现在了解什么是径向失真，需要利用失真系数（ k_1 、 k_2 和 k_3 ）来校正径向失真。`calibrateCamera.py`是摄像机校准程序，默认情况下不运行该程序。建议在生成目标上的特征点和图像上的特征点的过程中至少使用 20 个棋盘图像。Main 中的 `calibrate()` 将在`/data/calibration`中查找图像，但是我们也可以选择其他目录。

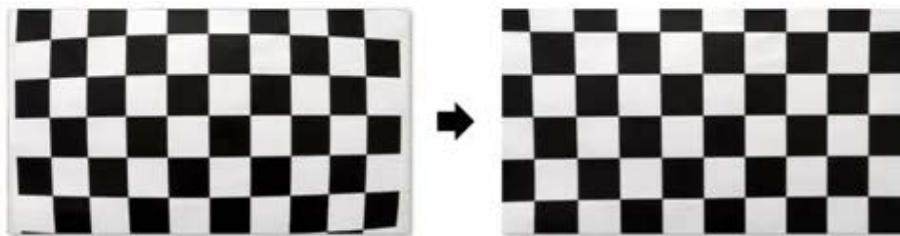


图1 左图：图像失真;右：未失真的图像

去除图像失真的整个过程是相当有趣的，OpenCV有一个很好的教程，解释了概念并举出一些例子。

透视变换 (`preprocess.py`: 8-19)

检测车道的第一步是调整我们的视觉系统，以鸟瞰的角度来观察前方的道路，这将有助于计算道路的曲率，因此将有助于我们预测未来几百米的转向角大小。自上而下视图的另一个好处是，它解决了车道线相交的问题。实际上只要沿道路行驶，车道线就是平行线。

鸟瞰图可以通过应用透视变换来实现，即将输入图像中车道区域四个点映射到所需点上，从而生成自顶向下的视图。这些点是根据个案确定，决定因素主要是摄像头在车辆中的位置及其视野。图2中的图片分别表示输入和转换后输出图像。



图2 左图：之前、右侧：之后

阈值 (preprocess.py: 22)

现在车道线是平行的，下一步将它们从输入图像上分割出来。输入图像包含RGB3个通道，车道线为白色或黄色。基于这个假设，输入图像可以转换为单个通道灰度图像，从而消除我们不需要的通道。另一个要转换为的颜色空间是HLS颜色空间，其中S通道可能会根据照明情况提供较好的结果。在以下示例中，将使用图像阈值，因为在给定的输入图像中它可以正常工作。图3在阈值处理后可视化输出。



图3 cv2.threshold(image, 220, 225, cv2.THRESH_BINARY)

下阈值（220）和上阈值（225）将根据输入图像手动调整。OpenCV有基于整体嵌套边缘检测的先进技术，而无需对阈值进行任何手动调整，但本文仍然使用的是简单的阈值技术。

车道像素查找 (laneDetection.py: 4~70)

预处理输入图像后，将在图像空间中确定并绘制车道。方法是在二进制图像（阈值图像）的下半部分绘制非零像素直方图，以观察模式：

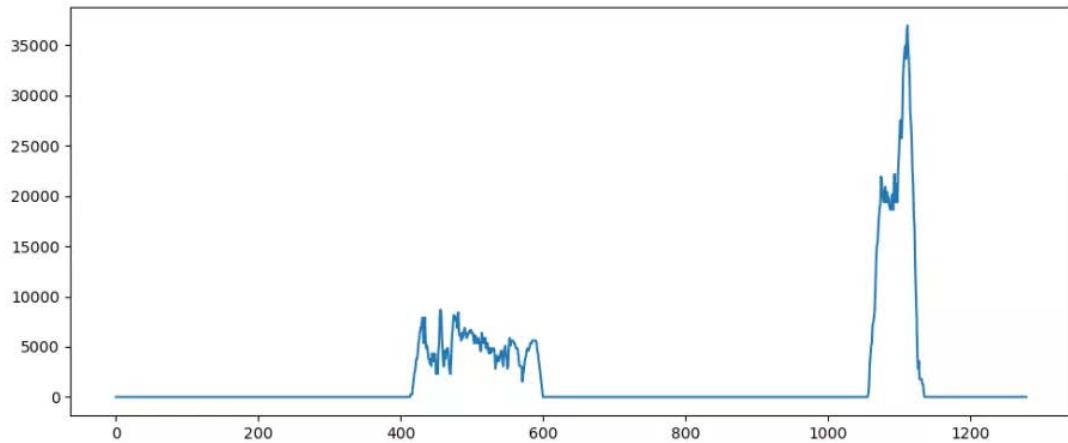


图4直方图x=像素, y = 计数

由于像素值是二进制的，峰值代表大多数非零像素的位置，因此可以很好地指示车道线。直方图中的x坐标用作搜索相应通道的起点。滑动窗口方法的概念将应用在这里，以下视频说明了滑动窗口的概念，图5中是结果。

00:25

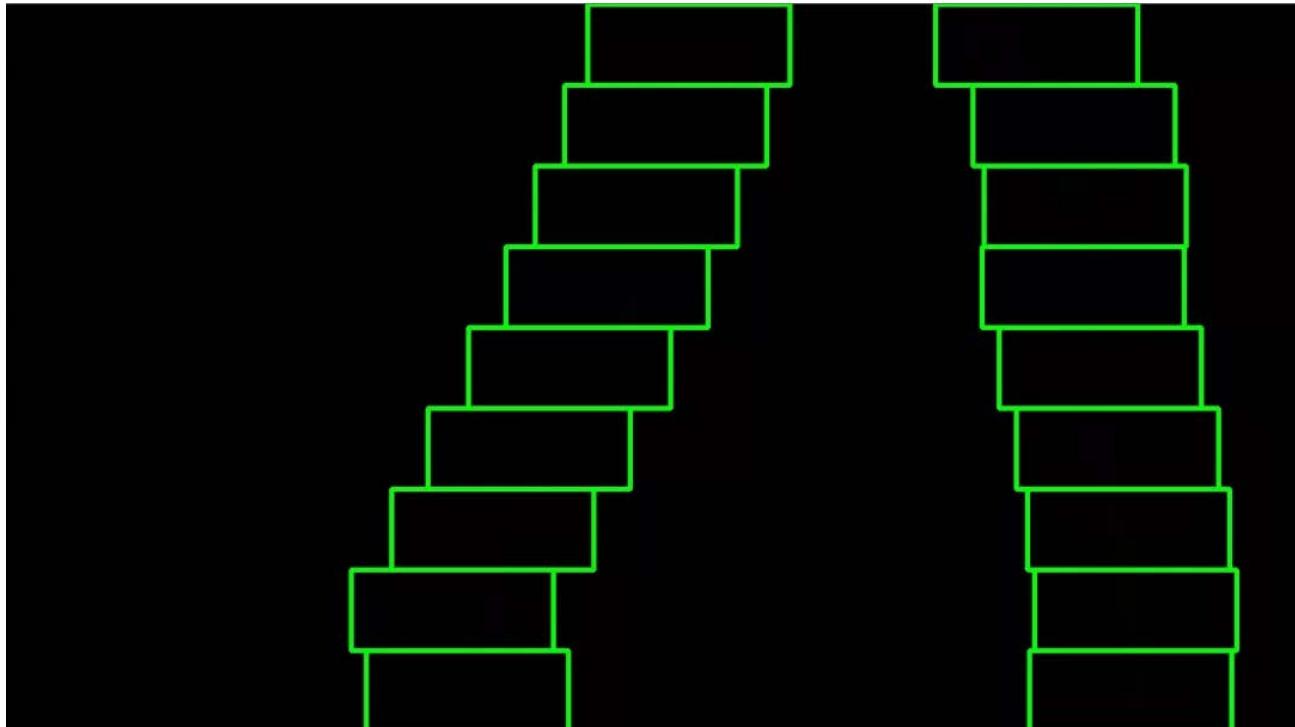


图5.滑动窗口的概念应用于图 4 的结果。

识别车道面积 (laneDetection.py: 85~149)

滑动窗口有助于估计每个车道区域的中心，使用这些 x 和 y 像素定位函数 `search_around_poly()` 可以适合二阶多项曲线。该函数适合 $f(y)$ 而不是 $f(x)$ ，因为通道在图像中是垂直的。图6很好地说明了这一步。

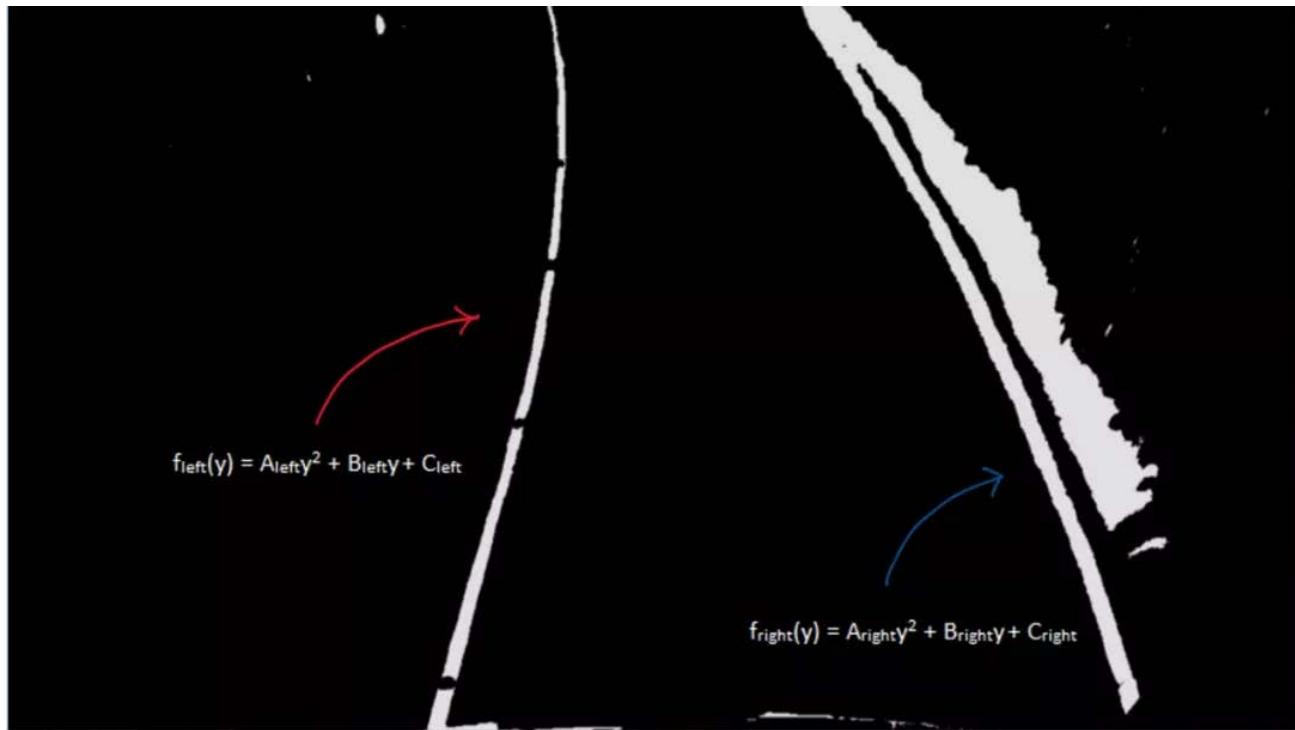


图6 在这些通道上检测到二阶多项形

下一步是计算曲率半径，该半径可以使用与曲线局部部分附近的点紧密拟合的圆进行计算，如图 7 所示。曲线在特定点的曲率半径可以定义为近似圆的半径。此半径可以使用图 7 中的公式计算。

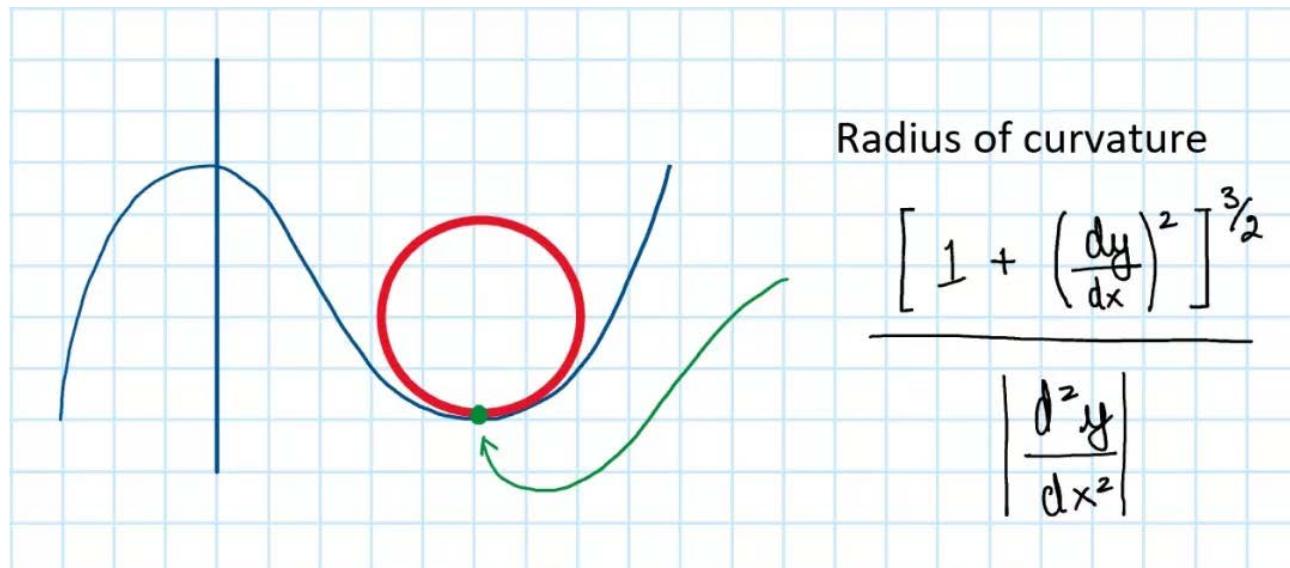


图7 曲率概念图的半径和用于计算 RoC 的方程

最后一步是在这些点之间放置一个四边形，并将其投影回原始图像，来突出显示车道区域。曲率的车道面积和半径是根据像素值计算的，像素值与真实世界空间不同，因此必须转换为现实世界的值，这涉及到测量我们投射扭曲图像的车道部分的长度和宽度。为简单起见，我们可以假设大多数车道通常长 30 米，宽 3.7 米，并且代码使用这些值将像素空间值转换为实际仪表测量值。



图 8 最终预期结果突出显示车道区域

结果

下面的视频显示，我们的结果还是非常不错的。

00:50

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM、三维视觉、传感器、自动驾驶、计算摄影、检测、分割、识别、医学影像、GAN、算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过。**添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





//
//

使用OpenCV实现道路车辆计数

原创 努比 小白学视觉 7月10日

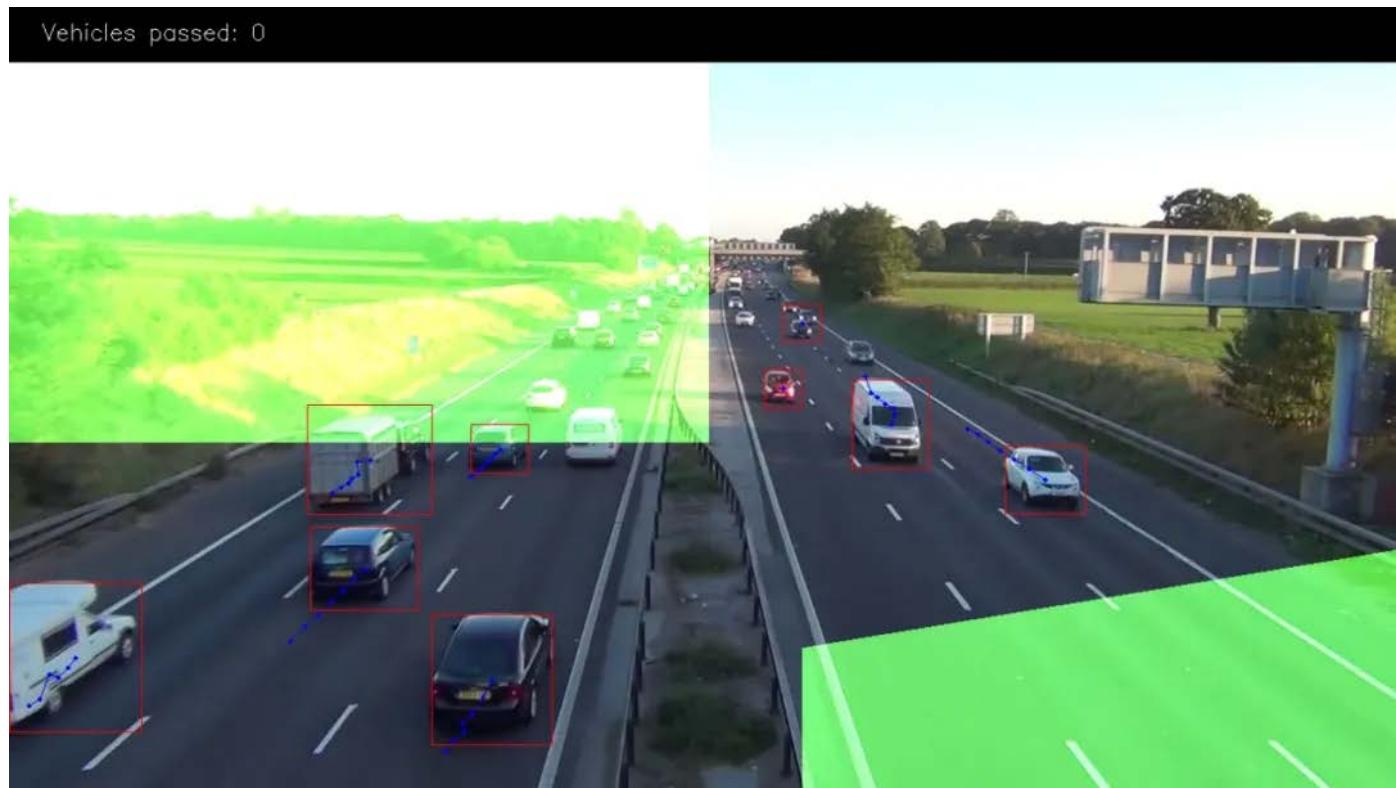
来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

Vehicles passed: 0



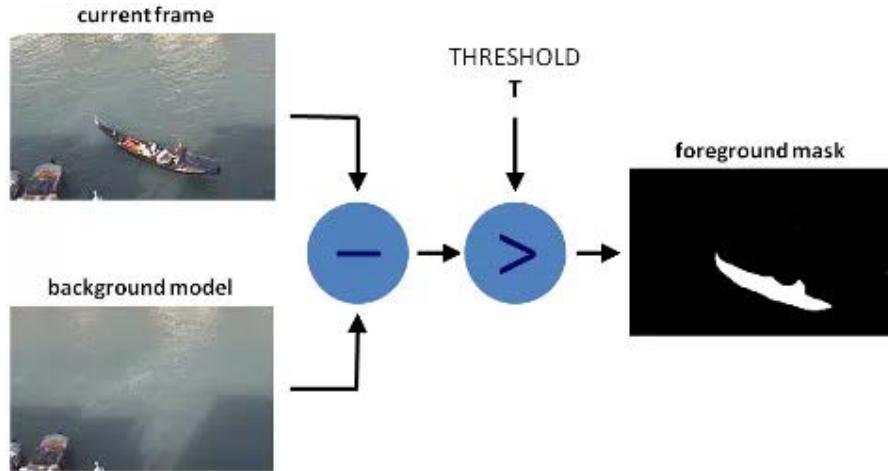
今天，我们将一起探讨如何基于计算机视觉实现道路交通计数。

在本教程中，我们将仅使用Python和OpenCV，并借助背景减除算法非常简单地进行运动检测。

我们将从以下四个方面进行介绍：

1. 用于物体检测的背景减法算法主要思想。
2. OpenCV图像过滤器。
3. 利用轮廓检测物体。
4. 建立进一步数据处理的结构。

背景扣除算法



有许多不同的背景扣除算法，但是它们的主要思想都很简单。

假设有一个房间的视频，在某些帧上没有人和宠物，那么此时的视频基本为静态的，我们将其称为背景（background_layer）。因此要获取在视频上移动的对象，我们只需要：用当前帧减去背景即可。

由于光照变化，人为移动物体，或者始终存在移动的人和宠物，我们将无法获得静态帧。在这种情况下，我们从视频中选出一些图像帧，如果绝大多数图像帧中都具有某个相同的像素点，则此将像素作为background_layer中的一部分。

我们将使用MOG算法进行背景扣除



原始帧

代码如下所示：

```

1 import os
2 import logging
3 import logging.handlers
4 import random
5
6 import numpy as np
7 import skvideo.io
8 import cv2
9 import matplotlib.pyplot as plt
10
11 import utils
12 # without this some strange errors happen

```

```
13 cv2.ocl.setUseOpenCL(False)
14 random.seed(123)
15
16 # =====
17 IMAGE_DIR = "./out"
18 VIDEO_SOURCE = "input.mp4"
19 SHAPE = (720, 1280) # HxW
20 # =====
21
22 def train_bg_subtractor(inst, cap, num=500):
23     """
24         BG subtractor need process some amount of frames to start giving result
25     """
26     print ('Training BG Subtractor...')
27     i = 0
28     for frame in cap:
29         inst.apply(frame, None, 0.001)
30         i += 1
31         if i >= num:
32             return cap
33
34 def main():
35     log = logging.getLogger("main")
36
37     # creating MOG bg subtractor with 500 frames in cache
38     # and shadow detection
39     bg_subtractor = cv2.createBackgroundSubtractorMOG2(
40         history=500, detectShadows=True)
41
42     # Set up image source
43     # You can use also CV2, for some reason it not working for me
44     cap = skvideo.io.vreader(VIDEO_SOURCE)
45
46     # skipping 500 frames to train bg subtractor
47     train_bg_subtractor(bg_subtractor, cap, num=500)
48
49     frame_number = -1
50     for frame in cap:
51         if not frame.any():
52             log.error("Frame capture failed, stopping...")
53             break
54
```

```

55     frame_number += 1
56     utils.save_frame(frame, "./out/frame_%04d.png" % frame_number)
57     fg_mask = bg_subtractor.apply(frame, None, 0.001)
58     utils.save_frame(frame, "./out/fg_mask_%04d.png" % frame_number)
59 # =====
60
61 if __name__ == "__main__":
62     log = utils.init_logging()
63
64     if not os.path.exists(IMAGE_DIR):
65         log.debug("Creating image directory `%s`...", IMAGE_DIR)
66         os.makedirs(IMAGE_DIR)
67
68     main()

```

处理后得到下面的前景图像



去除背景后的前景图像

我们可以看出前景图像上有一些噪音，可以通过标准滤波技术可以将其消除。

滤波

针对我们现在的情况，我们将需要以下滤波函数：**Threshold**、**Erode**、**Dilate**、**Opening**、**Closing**。

首先，我们使用“**Closing**”来移除区域中的间隙，然后使用“**Opening**”来移除个别独立的像素点，然后使用“**Dilate**”进行扩张以使对象变粗。代码如下：

```

1 def filter_mask(img):
2     kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
3     # Fill any small holes
4     closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
5     # Remove noise

```

```

6  opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
7  # Dilate to merge adjacent blobs
8  dilation = cv2.dilate(opening, kernel, iterations=2)
9  # threshold
10 th = dilation[dilation < 240] = 0
11 return th

```

处理后的前景如下：



利用轮廓进行物体检测

我们将使用cv2.findContours函数对轮廓进行检测。我们在使用的时候可以选择的参数为：

cv2.CV_RETR_EXTERNAL-----仅获取外部轮廓。

cv2.CV_CHAIN_APPROX_TC89_L1-----使用Teh-Chin链逼近算法（更快）

代码如下：

```

1 def get_centroid(x, y, w, h):
2     x1 = int(w / 2)
3     y1 = int(h / 2)
4     cx = x + x1
5     cy = y + y1
6     return (cx, cy)
7
8 def detect_vehicles(fg_mask, min_contour_width=35, min_contour_height=35):
9     matches = []
10    # finding external contours
11    im, contours, hierarchy = cv2.findContours(
12        fg_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_TC89_L1)
13    # filtering by width, height
14    for (i, contour) in enumerate(contours):
15        (x, y, w, h) = cv2.boundingRect(contour)
16        contour_valid = (w >= min_contour_width) and (

```

```

17         h >= min_contour_height)
18     if not contour_valid:
19         continue
20     # getting center of the bounding box
21     centroid = get_centroid(x, y, w, h)
22     matches.append(((x, y, w, h), centroid))
23
return matches

```

建立数据处理框架

我们都知道在ML和CV中，没有一个算法可以处理所有问题。即使存在这种算法，我们也不会使用它，因为它很难大规模有效。例如几年前Netflix公司用300万美元的奖金悬赏最佳电影推荐算法。有一个团队完成这个任务，但是他们的推荐算法无法大规模运行，因此其实对公司毫无用处。但是，Netflix公司仍奖励了他们100万美元。

接下来我们来建立解决当前问题的框架，这样可以使数据的处理更加方便

```

1 class PipelineRunner(object):
2     ...
3     Very simple pipeline.
4     Just run passed processors in order with passing context from one to
5     another.
6     You can also set log level for processors.
7     ...
8     def __init__(self, pipeline=None, log_level=logging.DEBUG):
9         self.pipeline = pipeline or []
10        self.context = {}
11        self.log = logging.getLogger(self.__class__.__name__)
12        self.log.setLevel(log_level)
13        self.log_level = log_level
14        self.set_log_level()
15    def set_context(self, data):
16        self.context = data
17    def add(self, processor):
18        if not isinstance(processor, PipelineProcessor):
19            raise Exception(
20                'Processor should be an instance of PipelineProcessor.')
21        processor.log.setLevel(self.log_level)
22        self.pipeline.append(processor)
23
24    def remove(self, name):

```

```

25     for i, p in enumerate(self.pipeline):
26         if p.__class__.__name__ == name:
27             del self.pipeline[i]
28             return True
29     return False
30
31     def set_log_level(self):
32         for p in self.pipeline:
33             p.log.setLevel(self.log_level)
34
35     def run(self):
36         for p in self.pipeline:
37             self.context = p(self.context)
38         self.log.debug("Frame #{} processed.", self.context['frame_number'])
39         return self.context
40
41     class PipelineProcessor(object):
42         ...
43         Base class for processors.
44         ...
45     def __init__(self):
46         self.log = logging.getLogger(self.__class__.__name__)

```

首先我们获取一张处理器运行顺序的列表，让每个处理器完成一部分工作，在按顺序完成执行以获得最终结果。

我们首先创建轮廓检测处理器。轮廓检测处理器只需将前面的背景扣除，滤波和轮廓检测部分合并在一起即可，代码如下所示：

```

1  class ContourDetection(PipelineProcessor):
2      ...
3          Detecting moving objects.
4          Purpose of this processor is to subtract background, get moving objects
5          and detect them with a cv2.findContours method, and then filter off-by
6          width and height.
7          bg_subtractor - background subtractor is instance.
8          min_contour_width - min bounding rectangle width.
9          min_contour_height - min bounding rectangle height.
10         save_image - if True will save detected objects mask to file.
11         image_dir - where to save images(must exist).
12         ...

```

```
13
14     def __init__(self, bg_subtractor, min_contour_width=35, min_contour_height=35,
15                  super(ContourDetection, self).__init__()
16                  self.bg_subtractor = bg_subtractor
17                  self.min_contour_width = min_contour_width
18                  self.min_contour_height = min_contour_height
19                  self.save_image = save_image
20                  self.image_dir = image_dir
21
22     def filter_mask(self, img, a=None):
23         ...
24
25             This filters are hand-picked just based on visual tests
26
27             ...
28
29             kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
30
31             # Fill any small holes
32             closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
33
34             # Remove noise
35             opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
36
37             # Dilate to merge adjacent blobs
38             dilation = cv2.dilate(opening, kernel, iterations=2)
39
40             return dilation
41
42
43     def detect_vehicles(self, fg_mask, context):
44
45         matches = []
46
47         # finding external contours
48         im2, contours, hierarchy = cv2.findContours(
49             fg_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_TC89_L1)
50
51         for (i, contour) in enumerate(contours):
52
53             (x, y, w, h) = cv2.boundingRect(contour)
54
55             contour_valid = (w >= self.min_contour_width) and (
56                 h >= self.min_contour_height)
57
58             if not contour_valid:
59
60                 continue
61
62             centroid = utils.get_centroid(x, y, w, h)
63
64             matches.append(((x, y, w, h), centroid))
65
66         return matches
67
68
69     def __call__(self, context):
70
71         frame = context['frame'].copy()
72
73         frame_number = context['frame_number']
74
75         fg_mask = self.bg_subtractor.apply(frame, None, 0.001)
76
77         # just thresholding values
```

```

55     fg_mask[fg_mask < 240] = 0
56     fg_mask = self.filter_mask(fg_mask, frame_number)
57     if self.save_image:
58         utils.save_frame(fg_mask, self.image_dir +
59                         "/mask_%04d.png" % frame_number, flip=False)
60     context['objects'] = self.detect_vehicles(fg_mask, context)
61     context['fg_mask'] = fg_mask
62     return context

```

现在，让我们创建一个处理器，该处理器将找出不同的帧上检测到的相同对象，创建路径，并对到达出口区域的车辆进行计数。代码如下所示：

```

1     ...
2     Counting vehicles that entered in exit zone.
3
4     Purpose of this class based on detected object and local cache create
5     objects pathes and count that entered in exit zone defined by exit masks.
6
7     exit_masks - list of the exit masks.
8     path_size - max number of points in a path.
9     max_dst - max distance between two points.
10    ...
11
12    def __init__(self, exit_masks=[], path_size=10, max_dst=30, x_weight=1.0, y_weight=1.0):
13        super(VehicleCounter, self).__init__()
14
15        self.exit_masks = exit_masks
16
17        self.vehicle_count = 0
18        self.path_size = path_size
19        self.pathes = []
20        self.max_dst = max_dst
21        self.x_weight = x_weight
22        self.y_weight = y_weight
23
24    def check_exit(self, point):
25        for exit_mask in self.exit_masks:
26            try:
27                if exit_mask[point[1]][point[0]] == 255:
28                    return True
29            except:

```

```
30         return True
31     return False
32
33     def __call__(self, context):
34         objects = context['objects']
35         context['exit_masks'] = self.exit_masks
36         context['pathes'] = self.pathes
37         context['vehicle_count'] = self.vehicle_count
38
39         if not objects:
40             return context
41
42         points = np.array(objects)[:, 0:2]
43         points = points.tolist()
44
45         # add new points if pathes is empty
46         if not self.pathes:
47             for match in points:
48                 self.pathes.append([match])
49
50         else:
51             # link new points with old pathes based on minimum distance between
52             # points
53             new_pathes = []
54
55             for path in self.pathes:
56                 _min = 999999
57                 _match = None
58
59                 for p in points:
60                     if len(path) == 1:
61                         # distance from last point to current
62                         d = utils.distance(p[0], path[-1][0])
63
64                     else:
65                         # based on 2 prev points predict next point and calculate
66                         # distance from predicted next point to current
67                         xn = 2 * path[-1][0][0] - path[-2][0][0]
68                         yn = 2 * path[-1][0][1] - path[-2][0][1]
69                         d = utils.distance(
70                             p[0], (xn, yn),
71                             x_weight=self.x_weight,
72                             y_weight=self.y_weight
73                         )
74
75             new_pathes.append(path)
76
77             for p in points:
78                 if len(path) == 1:
79                     d = utils.distance(p[0], path[-1][0])
80
81                     if d < _min:
82                         _min = d
83                         _match = path
84
85                 else:
86                     # based on 2 prev points predict next point and calculate
87                     # distance from predicted next point to current
88                     xn = 2 * path[-1][0][0] - path[-2][0][0]
89                     yn = 2 * path[-1][0][1] - path[-2][0][1]
90                     d = utils.distance(
91                         p[0], (xn, yn),
92                         x_weight=self.x_weight,
93                         y_weight=self.y_weight
94                     )
95
96                     if d < _min:
97                         _min = d
98                         _match = path
99
100                if _match:
101                    _match.append(p)
102
103            new_pathes.append(_match)
104
105        context['pathes'] = new_pathes
106
107        context['vehicle_count'] = len(new_pathes)
108
109        return context
```

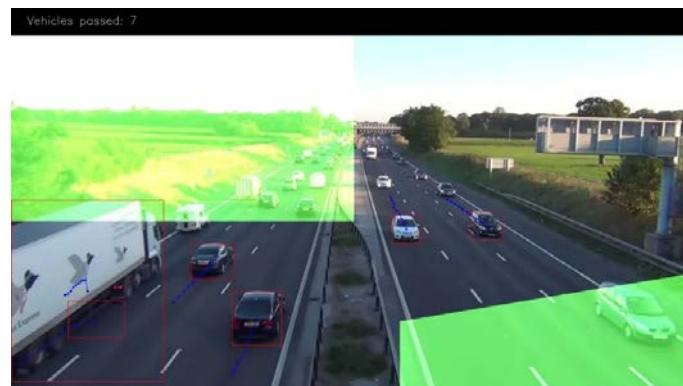
```
72         if d < _min:
73             _min = d
74             _match = p
75
76             if _match and _min <= self.max_dst:
77                 points.remove(_match)
78                 path.append(_match)
79                 new_pathes.append(path)
80
81             # do not drop path if current frame has no matches
82             if _match is None:
83                 new_pathes.append(path)
84
85             self.pathes = new_pathes
86
87             # add new pathes
88             if len(points):
89                 for p in points:
90                     # do not add points that already should be counted
91                     if self.check_exit(p[1]):
92                         continue
93                     self.pathes.append([p])
94
95             # save only last N points in path
96             for i, _ in enumerate(self.pathes):
97                 self.pathes[i] = self.pathes[i][self.path_size * -1:]
98
99             # count vehicles and drop counted pathes:
100            new_pathes = []
101            for i, path in enumerate(self.pathes):
102                d = path[-2:]
103
104                if (
105                    # need at least two points to count
106                    len(d) >= 2 and
107                    # prev point not in exit zone
108                    not self.check_exit(d[0][1]) and
109                    # current point in exit zone
110                    self.check_exit(d[1][1]) and
111                    # path len is bigger than min
112                    self.path_size <= len(path)
113                ):
```

```

114         self.vehicle_count += 1
115     else:
116         # prevent linking with path that already in exit zone
117         add = True
118         for p in path:
119             if self.check_exit(p[1]):
120                 add = False
121                 break
122         if add:
123             new_pathes.append(path)
124
125     self.pathes = new_pathes
126
127     context['pathes'] = self.pathes
128     context['objects'] = objects
129     context['vehicle_count'] = self.vehicle_count
130
131     self.log.debug('#VEHICLES FOUND: %s' % self.vehicle_count)
132
133     return context

```

上面的代码有点复杂，因此让我们一个部分一个部分的介绍一下。



上面的图像中绿色的部分是出口区域。我们在这里对车辆进行计数，只有当车辆移动的长度超过3个点我们才进行计算

我们使用掩码来解决这个问题，因为它比使用矢量算法有效且简单得多。只需使用“二进制和”即可选出车辆区域中点。设置方式如下：

```

1 EXIT PTS = np.array([
2     [[732, 720], [732, 590], [1280, 500], [1280, 720]],
3     [[0, 400], [645, 400], [645, 0], [0, 0]]

```

```

4  ])
5
6  base = np.zeros(SHAPE + (3,), dtype='uint8')
7  exit_mask = cv2.fillPoly(base, EXIT_PTS, (255, 255, 255))[:, :, 0]

```

现在我们将检测到的点链接起来。

对于第一帧图像，我们将所有点均添加为新路径。

接下来，如果len (path) == 1，我们在新检测到的对象中找到与每条路径最后一点距离最近的对象。

如果len (path) > 1，则使用路径中的最后两个点，即在同一条线上预测新点，并找到该点与当前点之间的最小距离。

具有最小距离的点将添加到当前路径的末端并从列表中删除。如果在此之后还剩下一些点，我们会将其添加为新路径。这个过程中我们还会限制路径中的点数。

```

1 new_paths = []
2  for path in self.paths:
3      _min = 999999
4      _match = None
5      for p in points:
6          if len(path) == 1:
7              # distance from last point to current
8              d = utils.distance(p[0], path[-1][0])
9          else:
10              # based on 2 prev points predict next point and calculate
11              # distance from predicted next point to current
12              xn = 2 * path[-1][0][0] - path[-2][0][0]
13              yn = 2 * path[-1][0][1] - path[-2][0][1]
14              d = utils.distance(
15                  p[0], (xn, yn),
16                  x_weight=self.x_weight,
17                  y_weight=self.y_weight
18              )
19
20          if d < _min:
21              _min = d
22              _match = p
23
24      if _match and _min <= self.max_dst:
25          points.remove(_match)
26          path.append(_match)

```

```

27     new_pathes.append(path)
28
29     # do not drop path if current frame has no matches
30     if _match is None:
31         new_pathes.append(path)
32
33 self.pathes = new_pathes
34
35 # add new pathes
36 if len(points):
37     for p in points:
38         # do not add points that already should be counted
39         if self.check_exit(p[1]):
40             continue
41         self.pathes.append([p])
42
43 # save only last N points in path
44 for i, _ in enumerate(self.pathes):
45     self.pathes[i] = self.pathes[i][self.path_size * -1:]

```

现在，我们将尝试计算进入出口区域的车辆。为此，我们需获取路径中的最后2个点，并检查 `len(path)` 是否应大于限制。

```

1 # count vehicles and drop counted pathes:
2     new_pathes = []
3     for i, path in enumerate(self.pathes):
4         d = path[-2:]
5         if (
6             # need at least two points to count
7             len(d) >= 2 and
8             # prev point not in exit zone
9             not self.check_exit(d[0][1]) and
10            # current point in exit zone
11            self.check_exit(d[1][1]) and
12            # path len is bigger than min
13            self.path_size <= len(path)
14        ):
15            self.vehicle_count += 1
16        else:
17            # prevent linking with path that already in exit zone
18            add = True

```

```

19     for p in path:
20         if self.check_exit(p[1]):
21             add = False
22             break
23         if add:
24             new_pathes.append(path)
25     self.pathes = new_pathes
26
27     context['pathes'] = self.pathes
28     context['objects'] = objects
29     context['vehicle_count'] = self.vehicle_count
30     self.log.debug('#VEHICLES FOUND: %s' % self.vehicle_count)
31     return context

```

最后两个处理器是CSV编写器，用于创建报告CSV文件，以及用于调试和精美图片的可视化。

```

1 class CsvWriter(PipelineProcessor):
2     def __init__(self, path, name, start_time=0, fps=15):
3         super(CsvWriter, self).__init__()
4         self.fp = open(os.path.join(path, name), 'w')
5         self.writer = csv.DictWriter(self.fp, fieldnames=['time', 'vehicles'])
6         self.writer.writeheader()
7         self.start_time = start_time
8         self.fps = fps
9         self.path = path
10        self.name = name
11        self.prev = None
12    def __call__(self, context):
13        frame_number = context['frame_number']
14        count = _count = context['vehicle_count']
15        if self.prev:
16            _count = count - self.prev
17        time = ((self.start_time + int(frame_number / self.fps)) * 100
18                + int(100.0 / self.fps) * (frame_number % self.fps))
19        self.writer.writerow({'time': time, 'vehicles': _count})
20        self.prev = count
21    return context
22 class Visualizer(PipelineProcessor):
23     def __init__(self, save_image=True, image_dir='images'):
24         super(Visualizer, self).__init__()
25         self.save_image = save_image

```

```
26     self.image_dir = image_dir
27
28     def check_exit(self, point, exit_masks=[]):
29         for exit_mask in exit_masks:
30             if exit_mask[point[1]][point[0]] == 255:
31                 return True
32         return False
33
34     def draw_pathes(self, img, pathes):
35         if not img.any():
36             return
37
38         for i, path in enumerate(pathes):
39             path = np.array(path)[:, 1].tolist()
40
41             for point in path:
42                 cv2.circle(img, point, 2, CAR_COLOURS[0], -1)
43                 cv2.polyline(img, [np.int32(path)], False, CAR_COLOURS[0], 1)
44
45         return img
46
47     def draw_boxes(self, img, pathes, exit_masks=[]):
48         for (i, match) in enumerate(pathes):
49             contour, centroid = match[-1][:2]
50
51             if self.check_exit(centroid, exit_masks):
52                 continue
53
54             x, y, w, h = contour
55
56             cv2.rectangle(img, (x, y), (x + w - 1, y + h - 1),
57                           BOUNDING_BOX_COLOUR, 1)
58
59             cv2.circle(img, centroid, 2, CENTROID_COLOUR, -1)
60
61         return img
62
63     def draw_ui(self, img, vehicle_count, exit_masks=[]):
64         # this just add green mask with opacity to the image
65
66         for exit_mask in exit_masks:
67
68             _img = np.zeros(img.shape, img.dtype)
69             _img[:, :] = EXIT_COLOR
70
71             mask = cv2.bitwise_and(_img, _img, mask=exit_mask)
72             cv2.addWeighted(mask, 1, img, 1, 0, img)
73
74         # drawing top block with counts
75
76         cv2.rectangle(img, (0, 0), (img.shape[1], 50), (0, 0, 0), cv2.FILLED)
77         cv2.putText(img, ("Vehicles passed: {total} ".format(total=vehicle_count),
78                      cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 1)
79
80         return img
81
82     def __call__(self, context):
83
84         frame = context['frame'].copy()
85         frame_number = context['frame_number']
86         pathes = context['pathes']
87         exit_masks = context['exit_masks']
```

```
68     vehicle_count = context['vehicle_count']
69     frame = self.draw_ui(frame, vehicle_count, exit_masks)
70     frame = self.draw_pathes(frame, pathes)
71     frame = self.draw_boxes(frame, pathes, exit_masks)
72     utils.save_frame(frame, self.image_dir +
73                       "/processed_%04d.png" % frame_number)
74
return context
```

结论

正如我们看到的那样，它并不像许多人想象的那么难。但是，如果小伙伴运行脚本，小伙伴会发现此解决方案并不理想，存在前景对象存在重叠的问题，并且它也没有按类型对车辆进行分类。但是，当相机有较好位置，例如位于道路正上方时，该算法具有很好的准确性。

如果本文对小伙伴有帮助，希望可以在文末来个“一键三连”。

 分享

 赞 107

 在看 128

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过**。添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





//
//

使用C#和OpenCV实现人脸替换

原创 努比 小白学视觉 7月13日

来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

本期我们将学习如何通过OpenCV实现图片中人脸的替换。



下面是已经完成替换的图片，是不是很酷。



在原图片中位于中前方的实际上是布拉德利·库珀。我们首先使用C#的“换脸”程序将另外一张脸叠加到布拉德利的脸上，然后用数字得到方式将其插入到布拉德利奥斯卡自拍照中。



图像获取

在C#中要解决这个问题，我们将使用**Accord库**、**OpenCvSharp3**以及**DLib**。Accord库非常适合创建计算机视觉应用程序。OpenCvSharp3是一个基于C#的OpenCV库，我们将使用这个库中的几个图像转换功能。在计算机视觉世界中，DLib则是人脸检测的首选库。虽然DLib完全用C++编写，但是DlibDotNet，将所有程序封装到C#中。

我们首先需要获得一张布拉德利的原始自拍照和单人照：



原始自拍

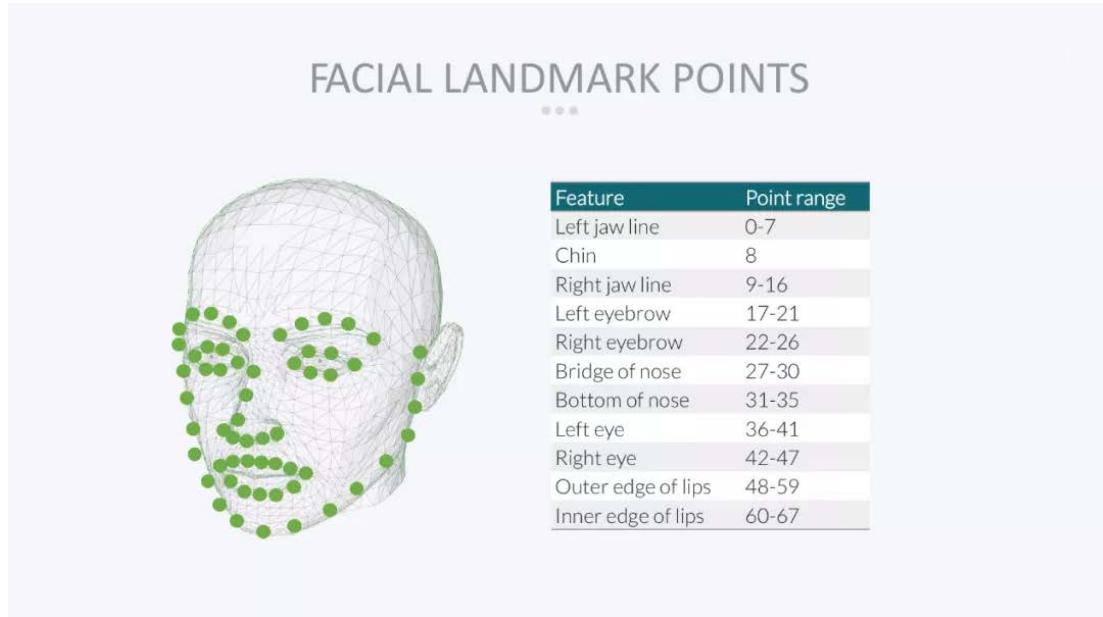


单人照

说明：使用以下代码可以将单人照与自拍照中的任何人交换面孔，但是就以上两幅图而言选择替换布拉德利·库珀效果最好，因为两个人具有相同的视线方向且脸型相似度很高。

界标点检测

接下来我们将使用Dlib库，对人脸进行检测。Dlib面部检测器可以识别出覆盖面部、下巴、眉毛、鼻子、眼睛和嘴唇的68个界标点。这些标记点预先确定的，并有给予其特定的标号，如下图所示。



Dlib运行速度很快，计算所有这些点的计算开销仅为1ms！因此它也可以实时跟踪这些点。以下C#代码，用于检测图片中脸上的所有界标点：

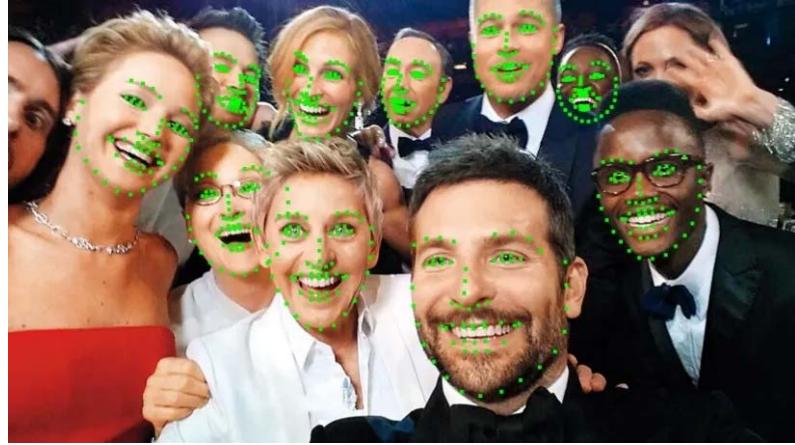
```

1  /// <summary>
2  /// Process the original selfie and produce the face-swapped image.
3  /// </summary>
4  /// <param name="image">The original selfie image.</param>
5  /// <param name="newImage">The new face to insert into the selfie.</param>
6  /// <returns>A new image with faces swapped.</returns>
7  private Bitmap ProcessImage(Bitmap image, Bitmap newImage)
8  {
9      // set up Dlib facedetectors and shapendetectors
10     using (var fd = FrontalFaceDetector.GetFrontalFaceDetector())
11     using (var sp = new ShapePredictor("shape_predictor_68_face_landmarks.dat"))
12     {
13         // convert image to dlib format
14         var img = image.ToArray2D<RgbPixel>();
15
16         // find bradley's faces in image
17         var faces = fd.Detect(img);
18         var bradley = faces[0];
19
20         // get bradley's Landmark points
21         var bradleyShape = sp.Detect(img, bradley);
22         var bradleyPoints = (from i in Enumerable.Range(0, (int)bradleyShape.Parts)
23                             let p = bradleyShape.GetPart((uint)i)
24                             select new OpenCvSharp.Point(p.X, p.Y)).ToArray();
25

```

```

26     // remainder of code goes here...
27 }
28 }
```



界标点检测结果

在这段代码中，我们首先实例化**FrontalFaceDetector**和**ShapePredictor**。为此小伙伴们需要注意以下两个问题：

- 在Dlib中，检测面部和检测界标点（或者称为“检测形状”）是两件不同的事情，它们的性能差异很大。人脸检测速度非常慢，而形状检测仅需约1毫秒，并且可以实时进行。
- ShapePredictor实际上是一个从完成训练的数据文件中加载出来的机器学习模型。我们也可以用自己喜欢的任何物体重新训练ShapePredictor，像人脸、猫狗脸、植物等。

接下来Dlib使用的图片格式与NET框架所使用的图片格式不同，因此我需要在运行上述代码之前先转换自拍的图片格式。其中**ToArray2D<>**方法即可将位图转换为阵列**RgbPixel**结构，这中结构正好可用于Dlib。

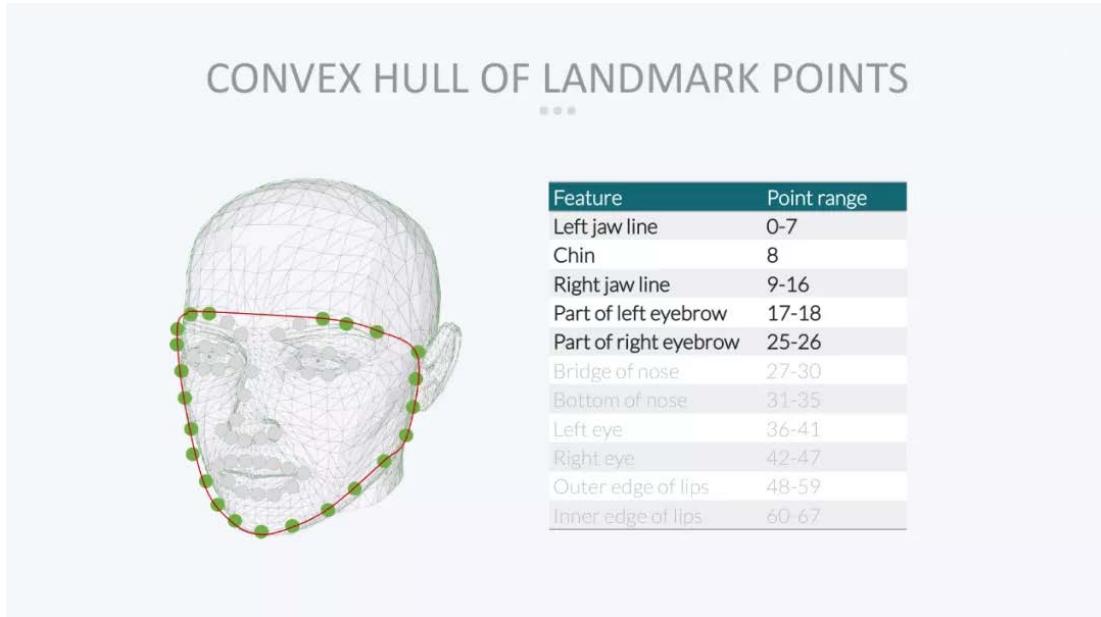
完成图像格式转换以后，我们使用**Detect()**来检测图像中的所有面孔。我们选取布拉德利·库珀的面孔提供后续使用，在本次检测中刚好为**faces(0)**。并且我们还用一个矩形来标识布拉德利的脸在图片中的位置。

接下来，我们在ShapePredictor上调用**Detect()**并提供自拍照和用于识别位置的脸部矩形。该函数的返回值是**GetPart()**方法的类，我们可以使用**GetPart()**方法来检索所有界标点的坐标。

我们的后续人脸交换工作将在OpenCV上完成，而OpenCV拥有自己特定的指针结构，因此在代码的最后我们将Dlib点转换为OpenCV点。

凸包提取

接下来，我们需要计算界点的**凸包**。一种简单的表达方式即，链接最外面的点形成围绕脸部的平滑边界。



OpenCV的内置功能可以帮助我们计算凸包：

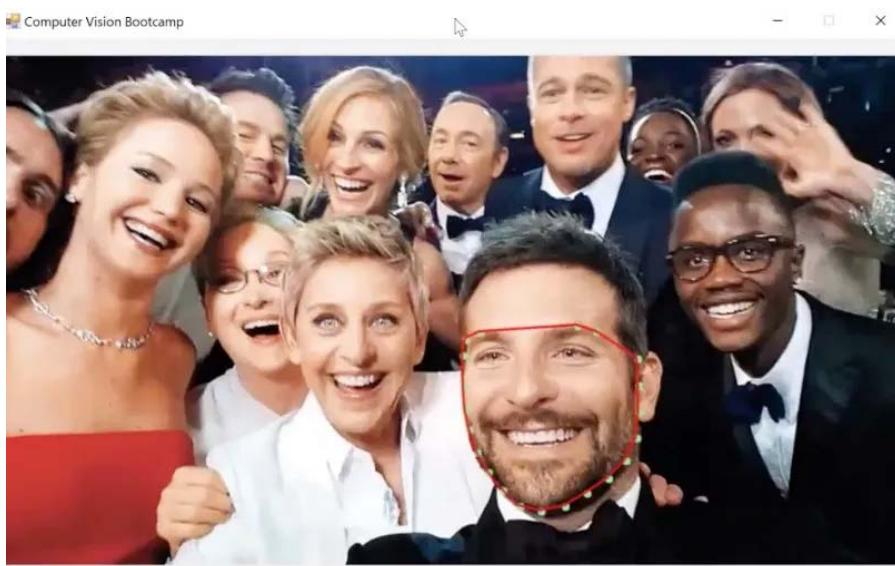
```

1 // get convex hull of bradley's points
2 var hull = Cv2.ConvexHullIndices(bradleyPoints);
3 var bradleyHull = from i in hull
4                     select bradleyPoints[i];
5
6 // the remaining code goes here...

```

ConvexHullIndices() 方法可以计算所有凸包界标点的指数，因此我们需要做的就是运行一个LINQ查询，以获取布莱德利·库珀的这些界标点的枚举。

下图是布莱德利脸上的凸包外观。



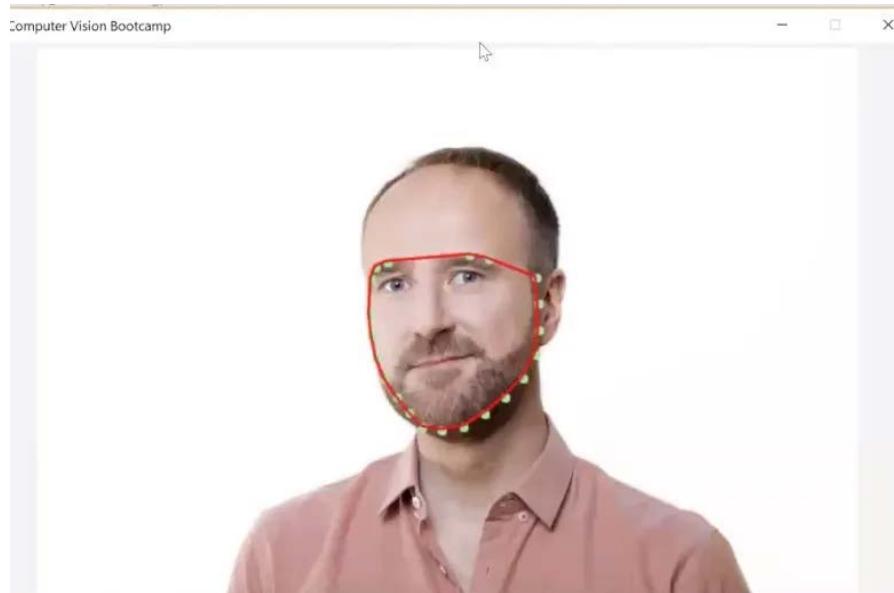
完成上述内容后，我们需要对单人照中的脸重复这些步骤：

```

1 // find Landmark points in face to swap
2 var imgMark = newImage.ToArray2D<RgbPixel>();
3 var faces2 = fd.Detect(imgMark);
4 var mark = faces2[0];
5 var markShape = sp.Detect(imgMark, mark);
6 var markPoints = (from i in Enumerable.Range(0, (int)markShape.Parts)
7                     let p = markShape.GetPart((uint)i)
8                     select new OpenCvSharp.Point(p.X, p.Y)).ToArray();
9
10 // get convex hull of mark's points
11 var hull2 = Cv2.ConvexHullIndices(bradleyPoints);
12 var markHull = from i in hull2
13                 select markPoints[i];
14
15 // the remaining code goes here...

```

这里的代码完全相同，只是将newImage换成了image。下面是从单人照中检测到的凸包外观。



到目前为止，我们已经获得了两个凸包外观，第一个是布莱德利脸上的凸包外观，第二个是单人照上的外观。

Delaunay三角形变形

单人照与布拉德利的凸包点的坐标之间没有线性关系。如果我们尝试直接移动所有像素，则必须使用慢速非线性变换。但是，通过首先在**Delaunay**三角形中覆盖布莱德利的脸，然后分别对每个三角形进行变形，整个操作将变得线性（且速度很快！）。

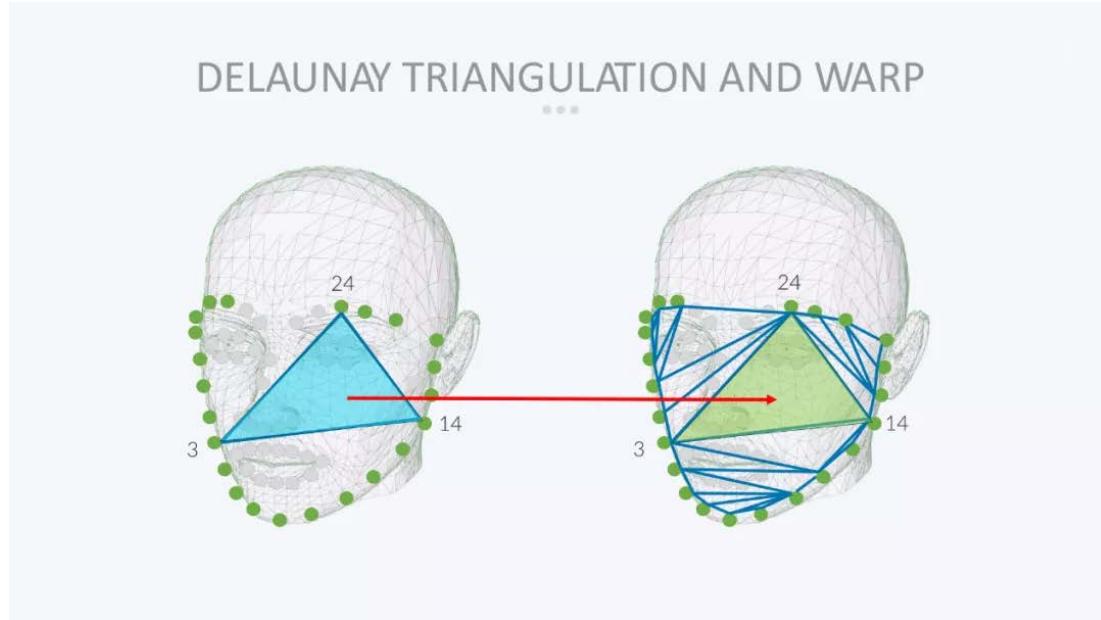
因此我们将为两人的脸计算Delaunay三角形。获取单人照中的三角形以后，对它们进行一定的变形，使其与布莱德利的脸完全匹配。

Delaunay Triangulation是一个创建三角形网格的过程，该三角形网格完全覆盖了布莱德利的脸，每个三角形由凸包上的三个特定的界标点组成。结果如下，蓝线即组成了Delaunay三角形：



接下来，我们将对单人照中Delaunay三角形进行变形，使之与布莱德利脸上的每个三角形保持一致，使新的面孔更加适应这张自拍照。在这个过程的每个三角形扭曲都是线性变换，因此可以使用超快速线性矩阵运算来移动每个三角形内的像素。

在下图中，我们扭曲了单人照中由界标点3、14和24组成的Delaunay三角形，以使其正好适合布莱德利的脸，并且这三个点与布莱德利的3、14和24界标点精确匹配：



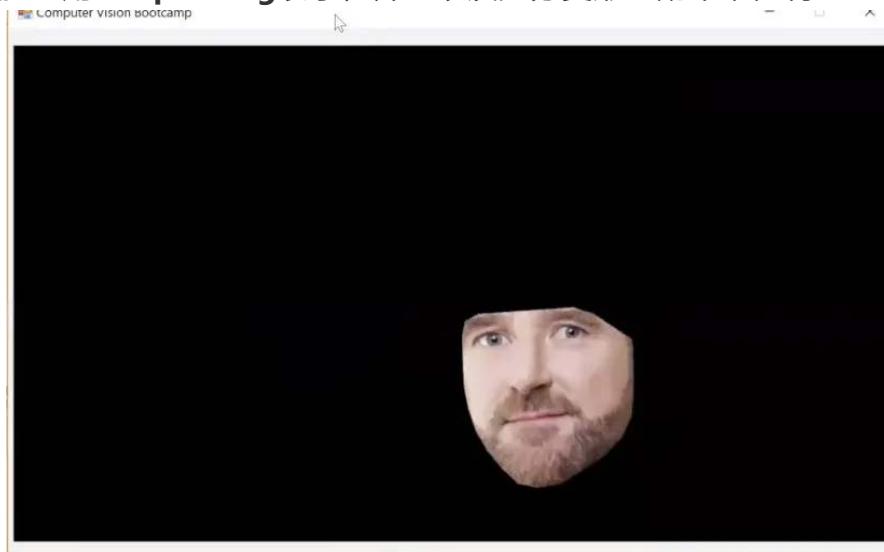
在C#中执行Delaunay三角剖分和变形的代码如下：

```
1 // calculate Delaunay triangles
```

```
2 var triangles = Utility.GetDelaunayTriangles(bradleyHull);  
3  
4 // get transformations to warp the new face onto Bradley's face  
5 var warps = Utility.GetWarps(markHull, bradleyHull, triangles);  
6  
7 // apply the warps to the new face to prep it for insertion into the main image  
8 var warpedImg = Utility.ApplyWarps(newImage, image.Width, image.Height, warps);  
9  
10 // the remaining code goes here...
```

我们使用一个便捷类 **Utility**，该类包含有 **GetDelaunayTriangles** 方法用于计算三角形，**GetWarps**方法用于计算每个三角形的翘曲，以及**ApplyWarps**方法使单人照脸部与布莱德利的脸部凸包相匹配。

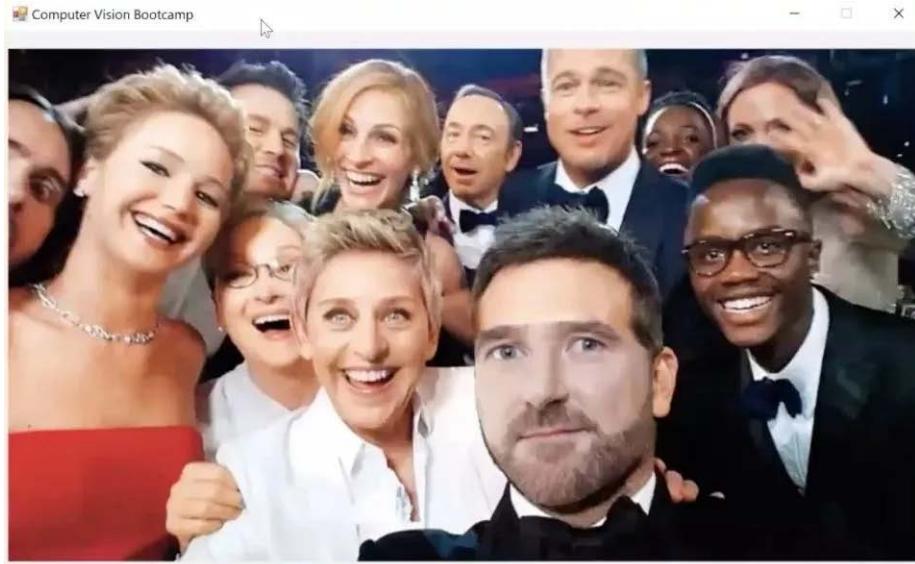
现在，单人照中的脸已用**warpedImg**表示，并且以及充分变形匹配布莱德利：



颜色转换

单人照与布拉德利的凸包点

我们还有一件事需要处理，单人照中人物的肤色与布拉德利的肤色并不相同。因此，如果我只是在自拍照中将图像放在其顶部，我们将在图像边缘看到剧烈的颜色变化：



为了解决这一问题，我们将使用OpenCV中的一个函数**SeamlessClone**，该函数可以将一个图像无缝地融合到另一个图像中，并消除任何颜色差异。

这是在C#中进行无缝克隆的方法：

```
1 // prepare a mask for the warped image
2 var mask = new Mat(image.Height, image.Width, MatType.CV_8UC3);
3 mask.setTo(0);
4 Cv2.FillConvexPoly(mask, bradleyHull, new Scalar(255, 255, 255), LineTypes.Link8);
5
6 // find the center of the warped face
7 var r = Cv2.BoundingRect(bradleyHull);
8 var center = new OpenCvSharp.Point(r.Left + r.Width / 2, r.Top + r.Height / 2);
9
10 // blend the warped face into the main image
11 var selfie = BitmapConverter.ToMat(image);
12 var blend = new Mat(selfie.Size(), selfie.Type());
13 Cv2.SeamlessClone(warpedImg, selfie, mask, center, blend, SeamlessCloneMethods.Normal);
14
15 // return the modified main image
16 return BitmapConverter.ToBitmap(blend);
```

使用**SeamlessClone**方法需要我们完成以下两件事：

- 首先需要一个mask来告诉它要混合哪些像素。我们在获取布拉德利面部凸包时使用**FillConvexPoly**方法即可计算所需的mask。
- 中心点处应该完全是单人照的肤色100%，距离中心点越远的像素将获得越接近的布拉德利肤色。我们通过调用**BoundingRect**获得布拉德利脸的边界框，然后取该框的中心来估计中心的位置。

然后，我调用SeamlessClone进行克隆并将结果存储在blend变量中，最终结果如下所示：

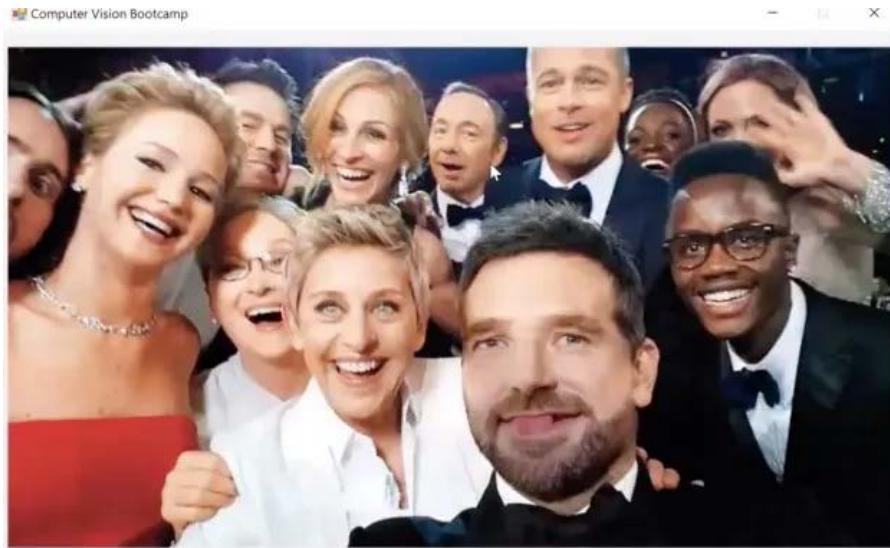


其他

看到这里小伙伴们可能在想为什么在这个过程中需要使用凸包，而不是直接不使用所有界标点来计算三角形？

原因实际上很简单，我们比较一下布拉德利的自拍与单人照。不难发现一个人在笑而另一个人没有？如果我们直接使用所有界标点，该程序将尝试把整个脸都进行变形，以便于和布拉德利的嘴唇，鼻子和眼睛完全匹配。这会使单人照中的人的嘴唇张开，以使单人照中的人物微笑并露出牙齿。

但结果似乎不太好。



如果只使用凸包壳点，该程序可以使单人照中人物的下巴变形，以匹配布拉德利的下颌线。但是它无法处理该人物的眼睛，鼻子和嘴巴。这意味着表情等在新图像中保持不变，看起来也更加自然。

最后，我们将使用**Instagram**滤镜来进一步消除色差：



总结

通过以上方式对面部进行一定的变形即可完成一幅图像的人脸插入工作，是不是很简单呢！

码字不易，如果小伙伴觉得对自己有帮助，可以在文末给小白一个“一键三连”嘛，让小白更有动力的干下去！

分享

赞 107

在看 128

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM、三维视觉、传感器、自动驾驶、计算摄影、检测、分割、识别、医学影像、GAN、算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过。**添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





//

//

使用Python, Keras和OpenCV进行实时面部检测

原创 努比 小白学视觉 今天

来自专辑

OpenCV应用

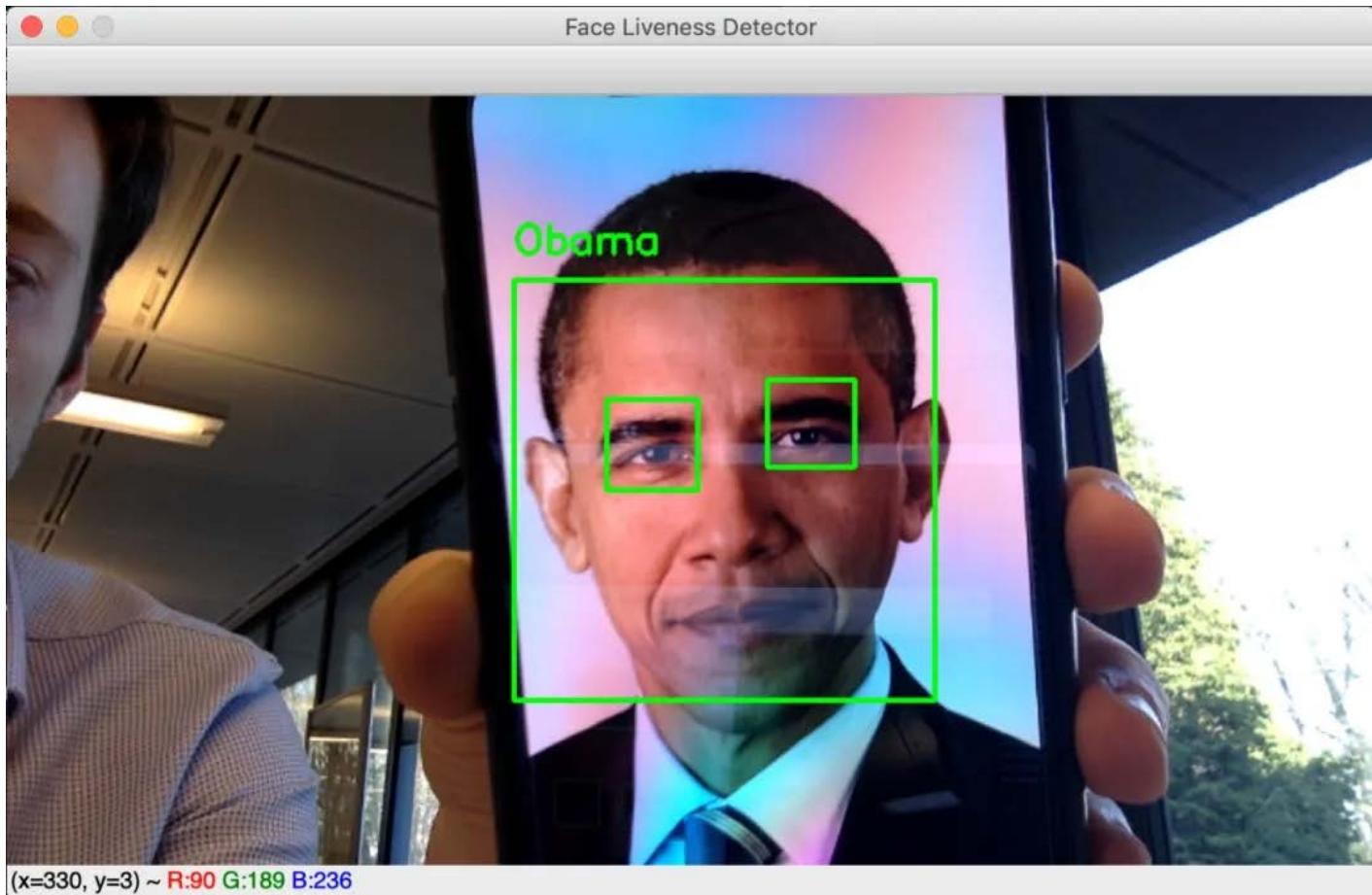
点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达



目前我们在互联网和论文中看到的大多数面部识别算法都是以图像为基础进行处理。这些方法在检测和识别来自摄像头的图像、或视频流各帧中的人脸时效果很好。但是，他们无法区分现实生活中的脸和照片上的人脸，因为这些算法处理的是2D帧。

现在，让我们想象一下，如果我们想要实现一个面部识别开门器。该系统可以很好地区分已知面孔和未知面孔，保证只有特定人员才能访问。尽管如此，任意一个陌生人只要拥有他们的照片就很容易进入该区域，这时3D检测器（类似于Apple的FaceID）就被纳入考虑范围。但是，如果我们没有3D探测器怎么办？



奥巴马脸部照片识别案例

本文旨在实现一种基于眨眼检测的面部活动检测算法来阻止照片的使用。该算法通过网络摄像头实时工作，并且仅在眨眼时才显示该人的姓名。程序流程如下：

1. 对网络摄像头生成的每一帧图像，进行面部检测。
2. 对于每个检测到的脸部区域，进行眼睛检测。
3. 对于检测到的每只眼睛，进行眨眼检测。
4. 如果在某个时刻检测到眼睛合上后又睁开了，则认为该人眨了眨眼，程序将显示他的名字（对于面部识别开门器，我们将授权该人进入）。

为了检测和识别面部，我们需要安装`face_recognition`库，该库提供了非常棒的深度学习算法来查找和识别图像中的人脸。特别是`face_locations`, `face_encodings`和`compare_faces`函数是3个最常用的函数。`face_locations`函数有两种可使用两种方法进行人脸检测：梯度方向的*Histogram (HOG)* 和*Convolutional神经网络 (CNN)*。由于时间限制，选择了*HOG*方法。`face_encodings`函数是一个预训练的卷积神经网络，能够将图像编码为128个特征的向量。这些向量的信息足以区分两个不同的人。最后，使用`compare_faces`计算两个嵌入向量之间的距离。它将允许算法识别从摄像头帧中提取的面部，并将其嵌入矢量与我们数据集中的所有编码面部进行比较。最近的向量对应于同一个人。

1. 已知的人脸数据集编码

就我们的算法而言，它能够识别我们自己和巴拉克·奥巴马。分别选择了约10张图片。以下是用于处理和编码已知面孔数据库的代码。

```

1 def process_and_encode(images):
2     known_encodings = []
3     known_names = []
4     print("[LOG] Encoding dataset ...")
5
6     for image_path in tqdm(images):
7         # Load image
8         image = cv2.imread(image_path)
9         # Convert it from BGR to RGB
10        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
11
12        # detect face in the image and get its location (square boxes coordinates)
13        boxes = face_recognition.face_locations(image, model='hog')
14
15        # Encode the face into a 128-d embeddings vector
16        encoding = face_recognition.face_encodings(image, boxes)
17
18        # the person's name is the name of the folder where the image comes from
19        name = image_path.split(os.path.sep)[-2]
20
21        if len(encoding) > 0 :
22            known_encodings.append(encoding[0])
23            known_names.append(name)
24
25    return {"encodings": known_encodings, "names": known_names}

```

现在我们知道了要识别的每个人的编码，我们可以尝试通过网络摄像头识别和识别面部。但是，在进行此部分操作之前，我们需要区分面部照片和活人的面部。

2.面部活跃度检测

提醒一下，目标是在某个点检测“睁开-闭合-睁开”的眼图。我训练了卷积神经网络来对眼睛是闭合还是睁开进行分类。选择的模型是LeNet-5，该模型已在 [Closed Eyes In The Wild \(CEW\)](#) 数据集中进行了训练。它由大小约为24x24的4800眼图像组成。

```

1 from keras.models import Sequential
2 from keras.layers import Conv2D
3 from keras.layers import AveragePooling2D

```

```
4 from keras.layers import Flatten
5 from keras.layers import Dense
6 from keras.preprocessing.image import ImageDataGenerator
7
8 IMG_SIZE = 24
9 def train(train_generator, val_generator):
10    STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
11    STEP_SIZE_VALID=val_generator.n//val_generator.batch_size
12
13    model = Sequential()
14
15    model.add(Conv2D(filters=6, kernel_size=(3, 3), activation='relu', input_shape=(IM
16    model.add(AveragePooling2D())
17
18    model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
19    model.add(AveragePooling2D())
20
21    model.add(Flatten())
22
23    model.add(Dense(units=120, activation='relu'))
24
25    model.add(Dense(units=84, activation='relu'))
26
27    model.add(Dense(units=1, activation = 'sigmoid'))
28
29
30    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
31
32    print('[LOG] Training CNN')
33
34    model.fit_generator(generator=train_generator,
35                         steps_per_epoch=STEP_SIZE_TRAIN,
36                         validation_data=val_generator,
37                         validation_steps=STEP_SIZE_VALID,
38                         epochs=20
39    )
40    return model
```

在评估模型时，准确率达到94%。

每次检测到眼睛时，我们都会使用模型预测其状态，并跟踪每个人的眼睛状态。因此，借助以下功能，可使检测眨眼变得很容易，该功能尝试在眼睛状态历史记录中查找闭合-闭合-闭合模式。

```

1 def isBlinking(history, maxFrames):
2     """ @history: A string containing the history of eyes status
3         where a '1' means that the eyes were closed and '0' open.
4         @maxFrames: The maximal number of successive frames where an eye is closed """
5     for i in range(maxFrames):
6         pattern = '1' + '0'*(i+1) + '1'
7         if pattern in history:
8             return True
9     return False

```

3.活人的面部识别

我们拥有构建“真实”面部识别算法的所有要素，只需要一种实时检测面部和眼睛的方法即可。我们选择使用OpenCV预训练的Haar级联分类器执行这些任务。

```

1 def detect_and_display(model, video_capture, face_detector, open_eyes_detector, left_eyes_detector):
2     frame = video_capture.read()
3     # resize the frame
4     frame = cv2.resize(frame, (0, 0), fx=0.6, fy=0.6)
5
6     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
7     rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
8
9     # Detect faces
10    faces = face_detector.detectMultiScale(
11        gray,
12        scaleFactor=1.2,
13        minNeighbors=5,
14        minSize=(50, 50),
15        flags=cv2.CASCADE_SCALE_IMAGE
16    )
17
18    # for each detected face
19    for (x,y,w,h) in faces:
20        # Encode the face into a 128-d embeddings vector
21        encoding = face_recognition.face_encodings(rgb, [(y, x+w, y+h, x)])[0]
22
23        # Compare the vector with all known faces encodings

```

```
24 matches = face_recognition.compare_faces(data["encodings"], encoding)
25
26 # For now we don't know the person name
27 name = "Unknown"
28
29 # If there is at least one match:
30 if True in matches:
31     matchedIdxs = [i for (i, b) in enumerate(matches) if b]
32     counts = {}
33     for i in matchedIdxs:
34         name = data["names"][i]
35         counts[name] = counts.get(name, 0) + 1
36
37     # The known encoding with the most number of matches corresponds to
38     name = max(counts, key=counts.get)
39
40     face = frame[y:y+h, x:x+w]
41     gray_face = gray[y:y+h, x:x+w]
42
43     eyes = []
44
45     # Eyes detection
46     # check first if eyes are open (with glasses taking into account)
47     open_eyes_glasses = open_eyes_detector.detectMultiScale(
48         gray_face,
49         scaleFactor=1.1,
50         minNeighbors=5,
51         minSize=(30, 30),
52         flags = cv2.CASCADE_SCALE_IMAGE
53     )
54     # if open_eyes_glasses detect eyes then they are open
55     if len(open_eyes_glasses) == 2:
56         eyes_detected[name]+='1'
57         for (ex,ey,ew,eh) in open_eyes_glasses:
58             cv2.rectangle(face,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
59
60     # otherwise try detecting eyes using left and right_eye_detector
61     # which can detect open and closed eyes
62     else:
63         # separate the face into left and right sides
64         left_face = frame[y:y+h, x+int(w/2):x+w]
65         left_face_gray = gray[y:y+h, x+int(w/2):x+w]
```

```
66
67     right_face = frame[y:y+h, x:x+int(w/2)]
68     right_face_gray = gray[y:y+h, x:x+int(w/2)]
69
70     # Detect the left eye
71     left_eye = left_eye_detector.detectMultiScale(
72         left_face_gray,
73         scaleFactor=1.1,
74         minNeighbors=5,
75         minSize=(30, 30),
76         flags = cv2.CASCADE_SCALE_IMAGE
77     )
78
79     # Detect the right eye
80     right_eye = right_eye_detector.detectMultiScale(
81         right_face_gray,
82         scaleFactor=1.1,
83         minNeighbors=5,
84         minSize=(30, 30),
85         flags = cv2.CASCADE_SCALE_IMAGE
86     )
87
88     eye_status = '1' # we suppose the eyes are open
89
90     # For each eye check wether the eye is closed.
91     # If one is closed we conclude the eyes are closed
92     for (ex,ey,ew,eh) in right_eye:
93         color = (0,255,0)
94         pred = predict(right_face[ey:ey+eh,ex:ex+ew],model)
95         if pred == 'closed':
96             eye_status='0'
97             color = (0,0,255)
98             cv2.rectangle(right_face,(ex,ey),(ex+ew,ey+eh),color,2)
99     for (ex,ey,ew,eh) in left_eye:
100         color = (0,255,0)
101         pred = predict(left_face[ey:ey+eh,ex:ex+ew],model)
102         if pred == 'closed':
103             eye_status='0'
104             color = (0,0,255)
105             cv2.rectangle(left_face,(ex,ey),(ex+ew,ey+eh),color,2)
106     eyes_detected[name] += eye_status
107
```

```

108     # Each time, we check if the person has blinked
109     # If yes, we display its name
110     if isBlinking(eyes_detected[name],3):
111         cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
112         # Display name
113         y = y - 15 if y - 15 > 15 else y + 15
114         cv2.putText(frame, name, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,
115
116     return frame

```

上面的功能是用于检测和识别真实面部的代码。它所需的输入参数：

- 型号：睁眼/闭眼分类器
- video_capture：流视频
- face_detector：Haar级联的人脸分类器。我们选择了`haarcascade_frontalface_alt.xml`
- open_eyes_detector：Haar级联睁眼分类器。我选择了
`haarcascade_eye_tree_eyeglasses.xml`
- left_eye_detector：Haar级联的左眼分类器。我选择了`haarcascade_lefteye_2splits.xml`, 它可以检测睁眼或闭眼。
- right_eye_detector：Haar级联的右眼分类器。我们选择了
`haarcascade_righteye_2splits.xml`, 它可以检测睁眼或闭眼。
- 数据：已知编码和已知名称的字典
- eyes_detected：包含每个名称的眼睛状态历史记录的字典。

在**第2至4行**，我们从网络摄像头流中抓取一帧，然后调整其大小以加快计算速度。在**第10行**，我们从帧中检测人脸，然后在**第21行**，将其编码为128-d向量。在**第23-38行中**，我们将此向量与已知的面部编码进行比较，然后通过计算匹配次数确定该人的姓名。匹配次数最多的一个被选中。从**第45行**开始，我们在脸部范围内检测眼睛是否存在。首先，我们尝试使用`open_eye_detector`检测睁眼。如果检测器成功，则在**第54行**，将 "1"添加到眼睛状态历史记录。如果第一个分类器失败了（可能是因为闭眼或仅仅是因为它不识别眼睛），这意味着`open_eye_detector`无法检测到闭合的眼睛，则使用`left_eye`和`right_eye`检测器。该面部分为左侧和右侧，以便对各个检测器进行分类。从**第92行**开始，提取眼睛部分，经过训练的模型预测眼睛是否闭合。如果检测到一只闭合的眼睛，则预测两只眼

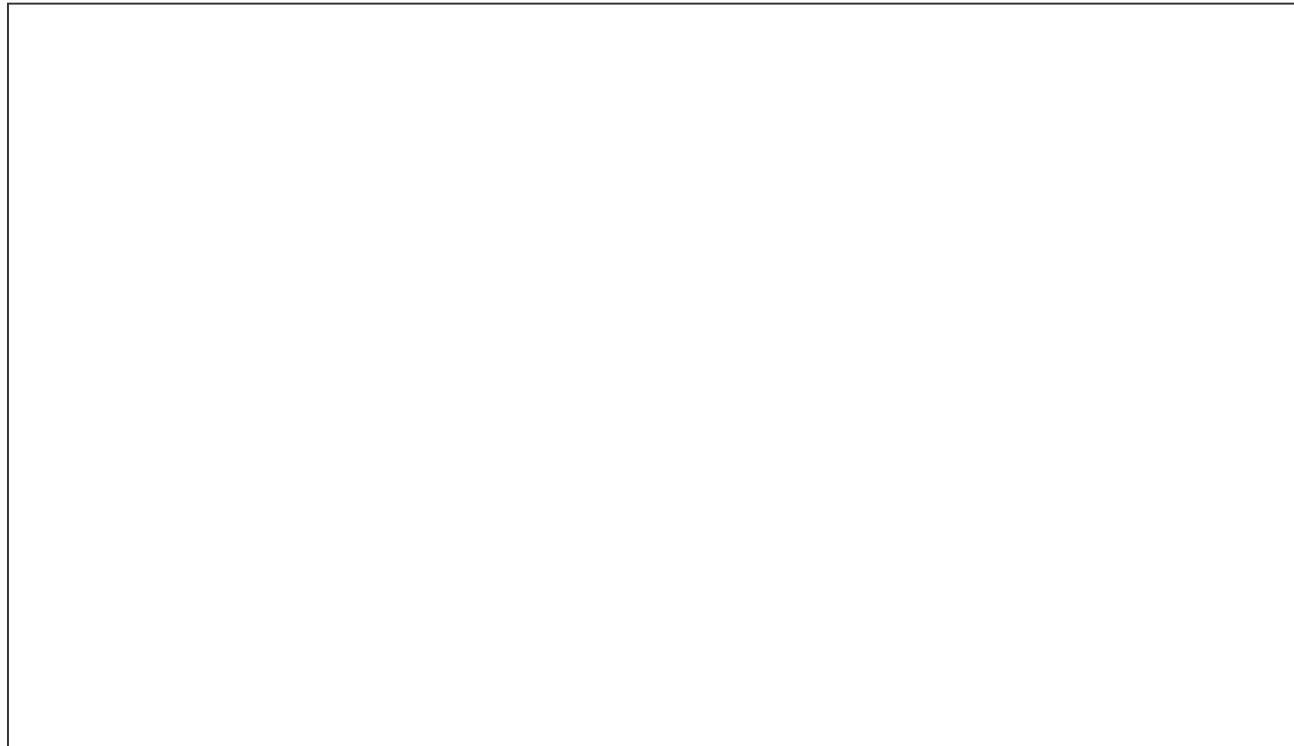
睛都闭合，并且将“o”添加到眼睛状态历史记录中。否则，可以得出结论，眼睛睁开了。最后在第110行，`isBlinking()`功能用于检测眨眼以及是否眨眼的人。

参考资料

- https://docs.opencv.org/3.4.3/d7/d8b/tutorial_py_face_detection.html
- <https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/>

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过**。添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~



使用TensorFlow和OpenCV实现口罩检测

原创 花生 小白学视觉 6月8日

来自专辑

OpenCV应用

点击上方“**小白学视觉**”，选择“**星标**”公众号

重磅干货，第一时间送达

在这段艰难的疫情期间，我们决定建立一个非常简单和基本的卷积神经网络(CNN)模型，使用TensorFlow与Keras库和OpenCV来检测人们是否佩戴口罩。



图片来源于澳门图片社

为了建立这个模型，我们将使用由Prajna Bhandary 提供的口罩数据集。这个数据集包括大约1,376幅图像，其中690幅图像包含戴口罩的人，686幅图像包含没有戴口罩的人。

我们将使用这些图像悬链一个基于TensorFlow框架的CNN模型，之后通过电脑端的网络摄像头来检测人们是否戴着口罩。此外，我们也可以使用手机相机做同样的事情。

数据可视化

首先，我们需要标记数据集中两个类别的全部图像。我们可以看到这里有690张图像在'yes'类里，也就是戴口罩的一类；有686张图像在'no'类中，也就是没有带口罩的一类。

```

1 The number of images with facemask labelled 'yes': 690
2 The number of images with facemask labelled 'no': 686

```

数据增强

这里，我们需要增强我们的数据集，为训练提供更多数量的图像。在数据增强时，我们旋转并翻转数据集中的每幅图像。在数据增强后，我们总共有2751幅图像，其中'yes'类中有1380幅图像，'no'类中有1371幅图像。

```

1 Number of examples: 2751
2 Percentage of positive examples: 50.163576881134134%, number of pos examples: 1380
3 Percentage of negative examples: 49.836423118865866%, number of neg examples: 1371

```

数据分割

我们将我们的数据分割成训练集和测试集，训练集中包含将要被CNN模型训练的图像，测试集中包含将要被我们模型测试的图像。

在此，我们取split_size=0.8，这意味着80%的图像将进入训练集，其余20%的图像将进入测试集。

```

1 The number of images with facemask in the training set labelled 'yes': 1104
2 The number of images with facemask in the test set labelled 'yes': 276
3 The number of images without facemask in the training set labelled 'no': 1096
4 The number of images without facemask in the test set labelled 'no': 275

```

在分割后，我们看到图像已经按照分割的百分比分配给训练集和测试集。

建立模型

在这一步中，我们将使用Conv2D，MaxPooling2D，Flatten，Dropout和Dense等各种层构建顺序CNN模型。在最后一个Dense层中，我们使用'softmax'函数输出一个向量，给出两个类中每个类的概率。

```

1 model = tf.keras.models.Sequential([
2     tf.keras.layers.Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
3     tf.keras.layers.MaxPooling2D(2,2),
4

```

```

5   tf.keras.layers.Conv2D(100, (3,3), activation='relu'),
6   tf.keras.layers.MaxPooling2D(2,2),
7
8   tf.keras.layers.Flatten(),
9   tf.keras.layers.Dropout(0.5),
10  tf.keras.layers.Dense(50, activation='relu'),
11  tf.keras.layers.Dense(2, activation='softmax')
12 ])
13 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

```

在这里，我们使用‘adam’优化器和‘binary_crossentropy’作为我们的损失函数，因为只有两个类。此外，也甚至可以使用MobileNetV2以获得更高的精度。

预训练CNN模型

在构建我们的模型之后，我们创建“train_generator”和“validation_generator”，以便在下一步将它们与我们的模型相匹配。我们看到训练集中总共有2200张图像，测试集中有551张图像。

```

1 Found 2200 images belonging to 2 classes.
2 Found 551 images belonging to 2 classes.

```

训练CNN模型

这一步是主要的步骤，我们使用训练集中的图像来训练我们的模型，并使用测试集中的数据来测试我们的训练结果，给出准确率。我们进行了30次迭代，我们训练的输出结果在下面给出。同时，我们可以训练更多的迭代，以获得更高的精度，以免发生过拟合。

```

1 history = model.fit_generator(train_generator,
2                                 epochs=30,
3                                 validation_data=validation_generator,
4                                 callbacks=[checkpoint])>>Epoch 30/30
5 220/220 [=====] - 231s 1s/step - loss: 0.0368 - acc: 0.9886

```

我们看到，在第30个时期之后，我们的模型中训练集的精度为98.86%，测试集的精度为96.19%。这意味着它是训练结果很好，没有任何过度拟合。

标记信息

在建立模型后，我们为我们的结果标记了两个概率

[‘0’作为‘without_mask’ 和‘1’作为‘with_mask’]。我们还使用RGB值设置边界矩形颜色。 [‘RED’ 代表 ‘without_mask’ 和‘GREEN’ 代表 ‘with_mask’]

```
1 labels_dict={0: 'without_mask', 1: 'with_mask'}  
2 color_dict={0:(0,0,255), 1:(0,255,0)}
```

导入人脸检测程序

在此之后，我们打算使用PC的网络摄像头来检测我们是否佩戴口罩。为此，首先我们需要实现人脸检测。在此，我们使用基于Haar特征的级联分类器来检测人脸的特征。

```
1 face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

这种级联分类器是由OpenCV设计的，通过训练数千幅图像来检测正面的人脸。代码中需要的.xml文件可以在公众号后台回复“口罩检测”获得。

检测是否戴口罩

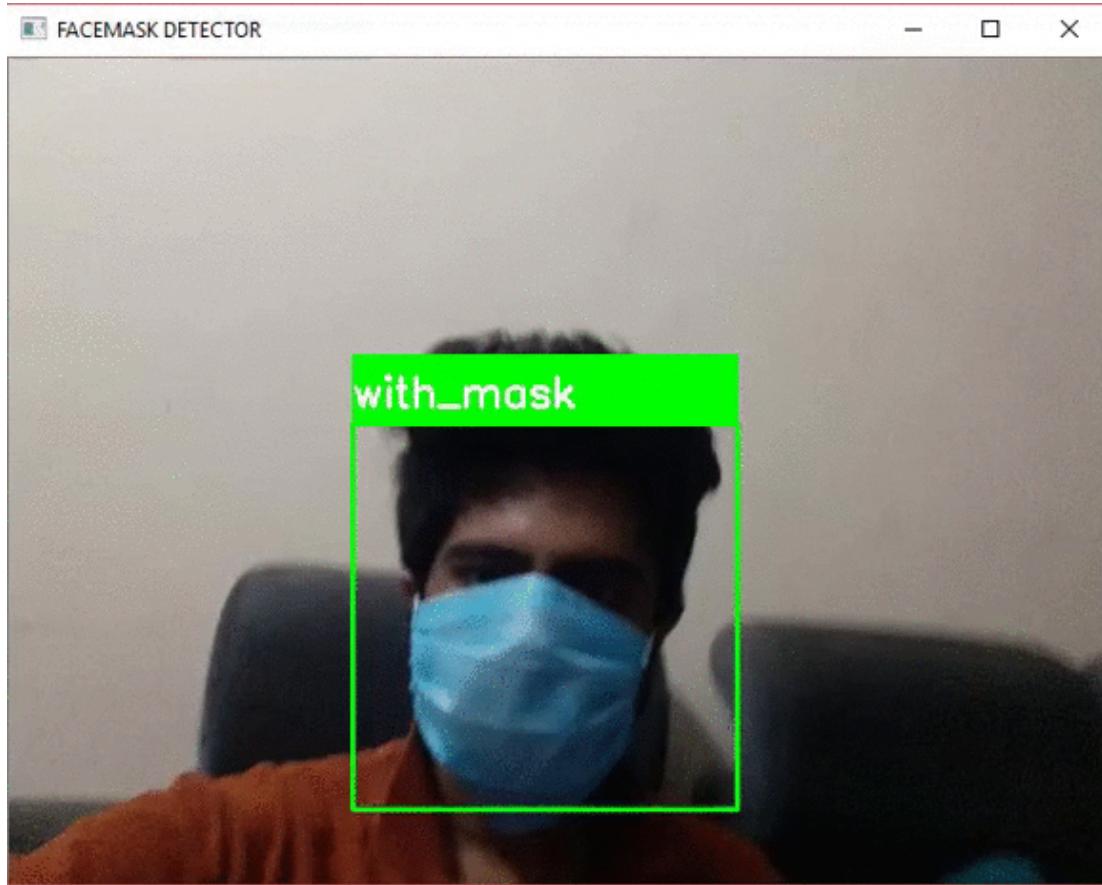
在最后一步中，我们通过OpenCV库运行一个无限循环程序，使用我们的网络摄像头，在其中我们使用Cascade Classifier检测人脸。代码是webcam = cv2.VideoCapture(0)表示使用网络摄像头。

该模型将预测两类中每一类的可能性([without_mask, with_mask])。基于概率的大小，标签将被选择并显示在我们脸的区域。

此外，还可以下载用于手机和PC的DroidCam 应用程序来使用我们的移动相机，并将代码中的0改为1 webcam= cv2.VideoCapture(1).

测试：

我们来看一下测试的结果



从上面的演示视频中，我们看到模型能够正确地检测是否佩戴面具并在标签上显示相同的内容。

如果小伙伴需要本文的相关代码和文件，可以在公众号后台回复【**口罩检测**】获得。



快速指南：使用OpenCV预处理神经网络中的面部图像的

原创 努比 小白学视觉 3天前

来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

本期将介绍脸部检测、眼睛检测；图像拉直、裁剪、调整大小、归一化等内容



目前，涉及面部分类的**计算机视觉**问题，通常都需要使用深度学习。因此在将图像输入神经网络之前，需要经过一个预处理阶段，以便达到更好的分类效果。

图像预处理通常来说非常简单，只需执行几个简单的步骤即可轻松完成。但为了提高模型的准确性，这也是一项非常重要的任务。对于这些问题，我们可以使用[OpenCV](#)完成：一个针对（实时）计算机视觉应用程序的高度优化的开源库，包括C ++，Java和Python语言。

接下来我们将一起探索可能会应用在每个面部分类或识别问题上应用的基本原理，示例和代码。

注意：下面使用的所有图像均来自memes.。

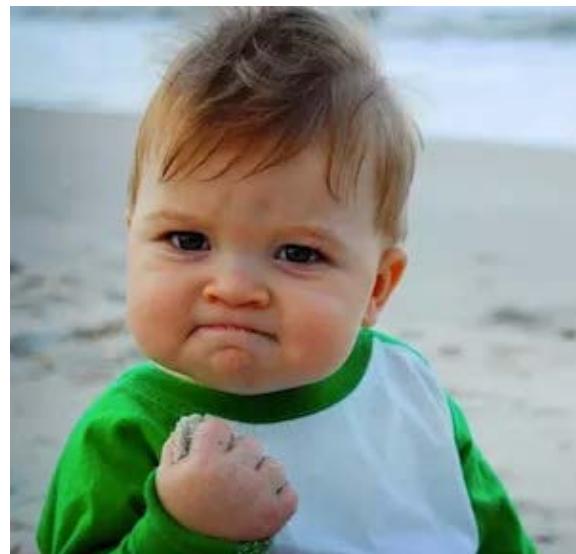
图片载入

我们使用该 `imread()` 函数加载图像，并指定文件路径和图像模式。第二个参数对于运行基本通道和深度转换很重要。

```
img = cv2.imread('path/image.jpg', cv2.IMREAD_COLOR)
```

要查看图像可以使用 `imshow()` 功能：

```
cv2.imshow(img)
```



如果使用的 `type(img)` 话，将显示该图像的尺寸包括高度、重量、通道数。

彩色图像有**3个通道**：蓝色，绿色和红色（在OpenCV中按此顺序）。

blue channel



green channel



red channel



我们可以很轻松查看单个通道：

```
# Example for green channel  
img[:, :, 0]; img[:, :, 2]; cv2.imshow(img)
```

Grayscale version

灰度图像

为了避免在人脸图像分类过程中存在的干扰，通常选择黑白图像（当然也可以使用彩图！请小伙伴们自行尝试两者并比较结果）。要获得灰度图像，我们只需要在图像加载函数中通过将适当的值作为第二个参数传递来指定它：

```
img = cv2.imread('path/image.jpg', cv2.IMREAD_GRAYSCALE)
```



现在，我们的图像只有一个灰度通道了！

面部和眼睛检测

在处理人脸分类问题时，我们可能需要先对图形进行裁剪和拉直，再进行人脸检测以验证是否有人脸的存在。[为此，我们将使用OpenCV中自带的基于Haar特征的级联分类器进行对象检测。](#)

首先，我们选择用于面部和眼睛检测的预训练分类器。以下时[可用的XML文件](#)列表：

1) 对于**面部检测**，OpenCV提供了这些（从最松的先验到最严格的先验）：

- *haarcascade_frontalface_default.xml*
- *haarcascade_frontalface_alt.xml*
- *haarcascade_frontalface_alt2.xml*

- haarcascade_frontalface_alt_tree.xml

2) 对于**眼睛检测**，我们可以选择以下两种：

- haarcascade_eye.xml
- haarcascade_eye_tree_eyeglasses.xml (正在尝试处理眼镜!)

我们以这种方式加载预训练的分类器：

```
face_cascade      = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')eyes_cascade      =
cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
```

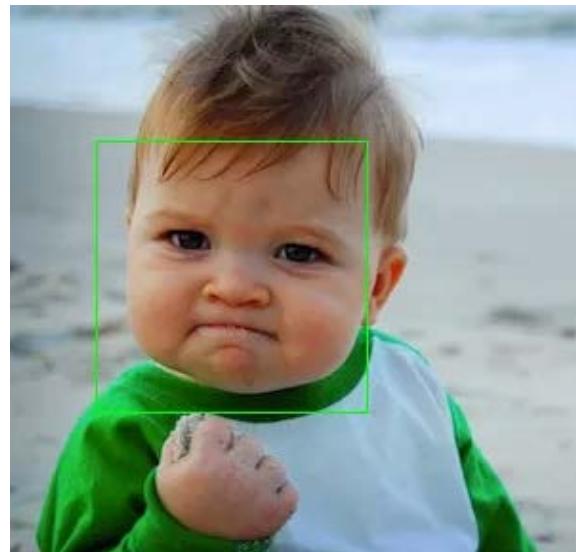
我们可以测试几种组合，但我们要记住一点，没有一种分类器在所有情况下都是最好的（如果第一个分类失败，您可以尝试第二个分类，甚至尝试所有分类）。

对于人脸检测，我们可使用以下代码：

```
faces_detected = face_cascade.detectMultiScale(img, scaleFactor=1.1,
minNeighbors=5)
```

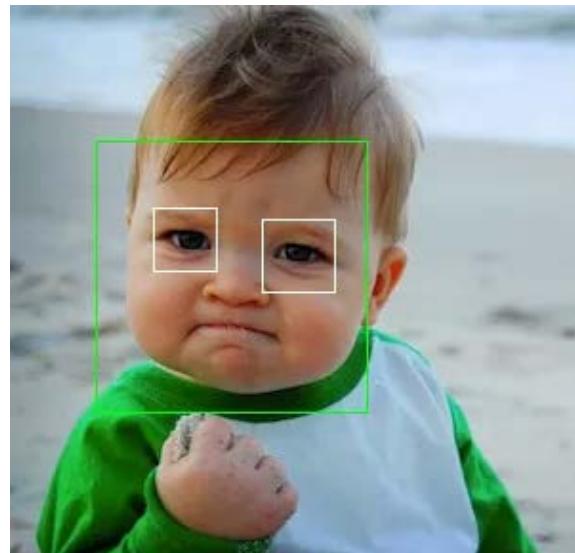
结果是一个数组，其中包含所有检测到的脸部特征的矩形位置。我们可以很容易地绘制它：

```
(x, y, w, h) = faces_detected[0]
cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 1);
cv2.imshow(img)
```



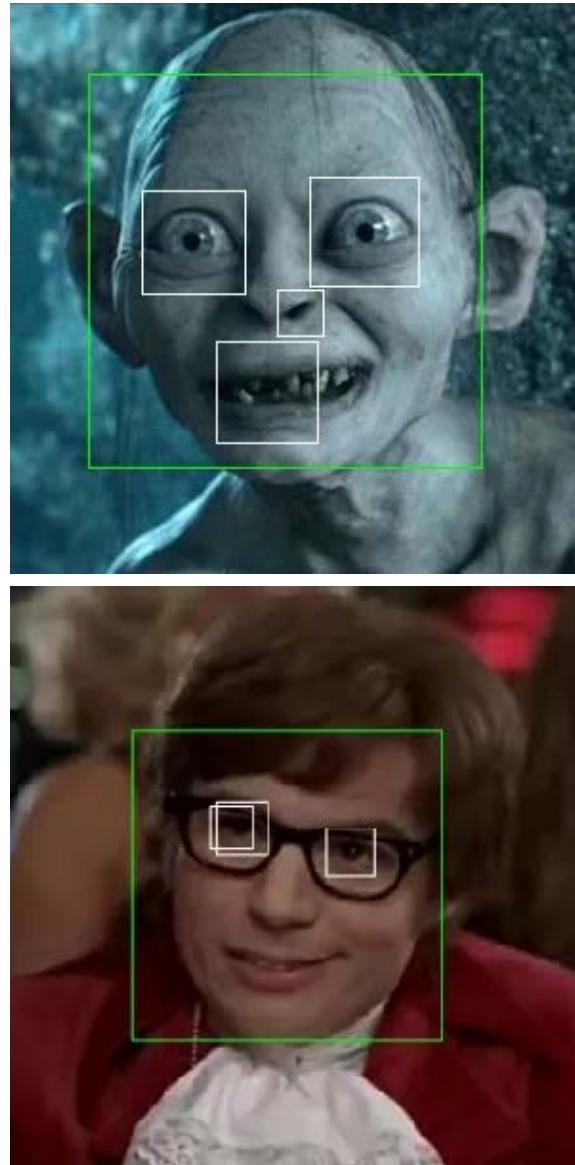
对于眼睛，我们以类似的方式进行，但将搜索范围缩小到刚刚提取出来的面部矩形框内：

```
eyes = eyes_cascade.detectMultiScale(img[y:y+h, x:x+w]) for (ex, ey, ew, eh) in eyes:  
    cv2.rectangle(img, (x+ex, y+ey), (x+ex+ew, y+ey+eh),  
                 (255, 255, 255), 1)
```



尽管这是预期的结果，但是很多时候再提取的过程中我们会遇到一些难以解决的问题。比如我们没有正面清晰的人脸视图。



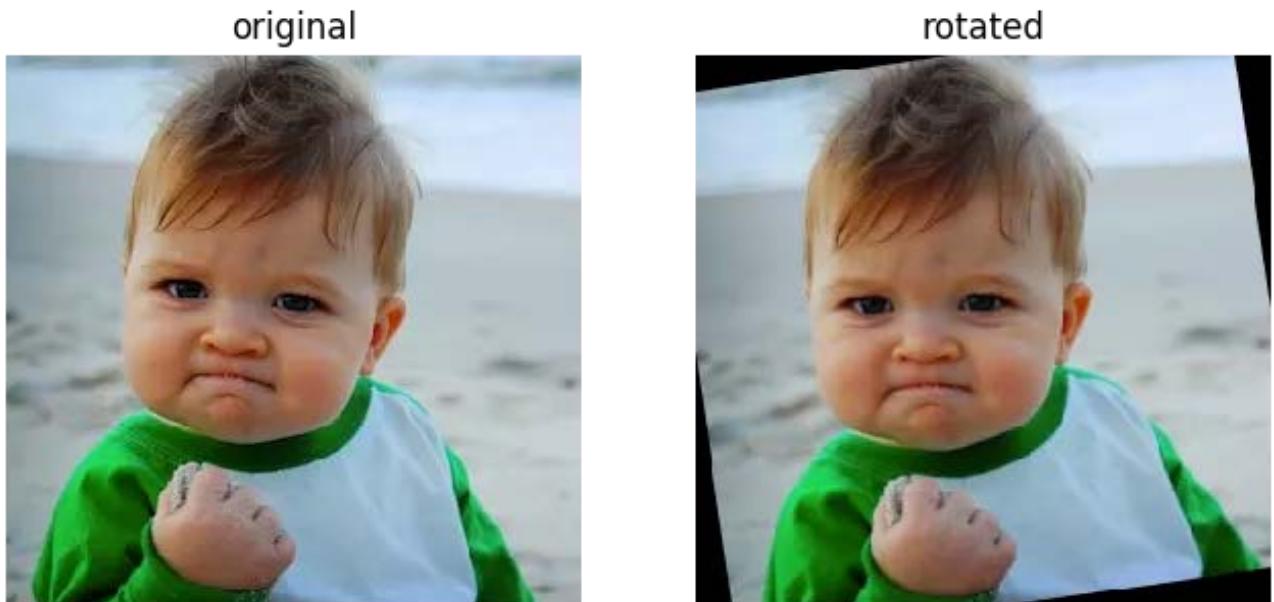


不能正确检测的案例

脸部旋转

通过计算两只眼睛之间的角度，我们就可以拉直面部图像（这很容易）。计算之后，我们仅需两个步骤即可旋转图像：

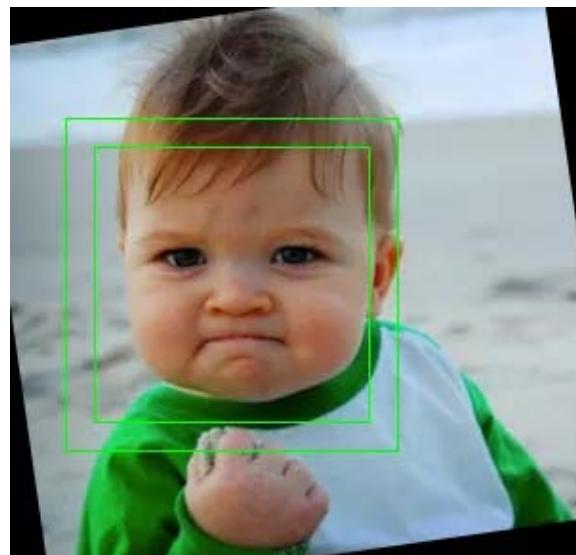
```
rows, cols = img.shape[:2]
M = cv2.getRotationMatrix2D((cols/2, rows/2), <angle>, 1)
img_rotated = cv2.warpAffine(face_orig, M, (cols,rows))
```



裁脸

为了帮助我们的神经网络完成面部分类任务，最好去除外界无关信息，例如背景，衣服或配件。在这些情况下，面部裁切非常方便。

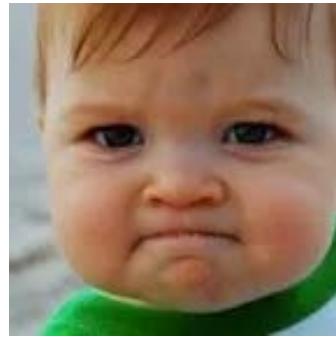
我们需要做的第一件事是再次从旋转后的图像中获取面部矩形。然后我们需要做出决定：我们可以按原样裁剪矩形区域，也可以添加额外的填充，以便在周围获得更多空间。这取决于要解决的具体问题（按年龄，性别，种族等分类）；也许我们需要保留头发，也许不需要。



最后进行裁剪（`p`用于填充）：

```
cv2.imwrite('crop.jpg', img_rotated[y-p+1:y+h+p, x-p+1:x+w+p])
```

现在这张脸的图像是非常单一的，基本可用于深度学习：



图像调整大小

神经网络需要的所有输入图像具有相同的形状和大小，因为GPU应用相同的指令处理一批相同大小图像，可以达到较快的速度。我们虽然可以随时调整它们的大小，但这并不是一个好主意，因为需要在训练期间将对每个文件执行几次转换。因此，如果我们的数据集包含大量图像，我们应该考虑在训练阶段之前实施批量调整大小的过程。

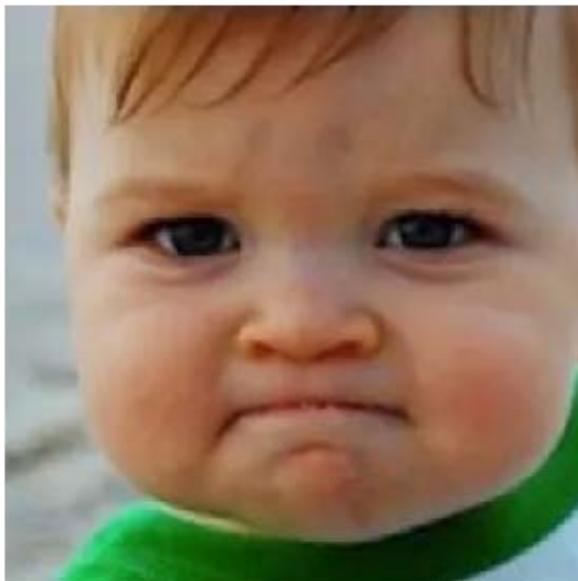
在OpenCV中，我们可以与同时执行缩小和升频`resize()`，有几个插值方法[可用](#)。指定最终大小的示例：

```
cv2.resize(img, (<width>, <height>), interpolation=cv2.INTER_LINEAR)
```

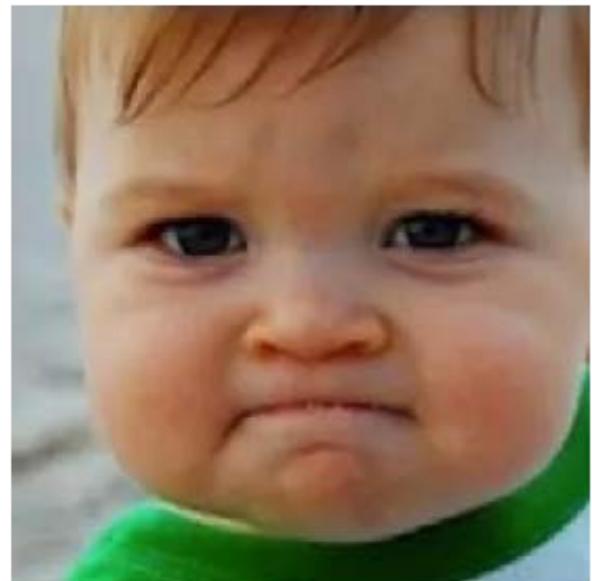
要缩小图像，OpenCV建议使用`INTER_AREA`插值法，而放大图像时，可以使用`INTER_CUBIC`（慢速）或`INTER_LINEAR`（更快，但效果仍然不错）。最后，这是质量和时间之间的权衡。

我对升级进行了快速比较：

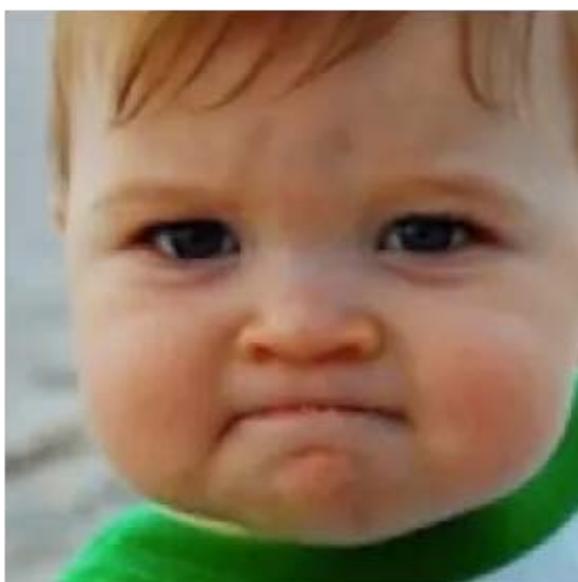
cubic



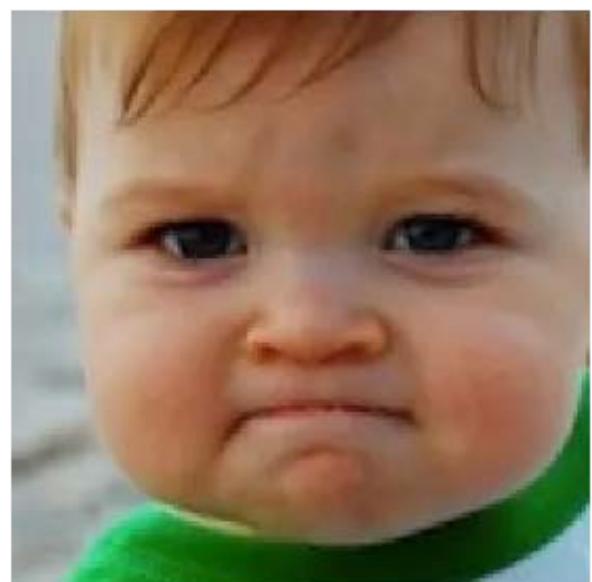
lanczos



linear



area



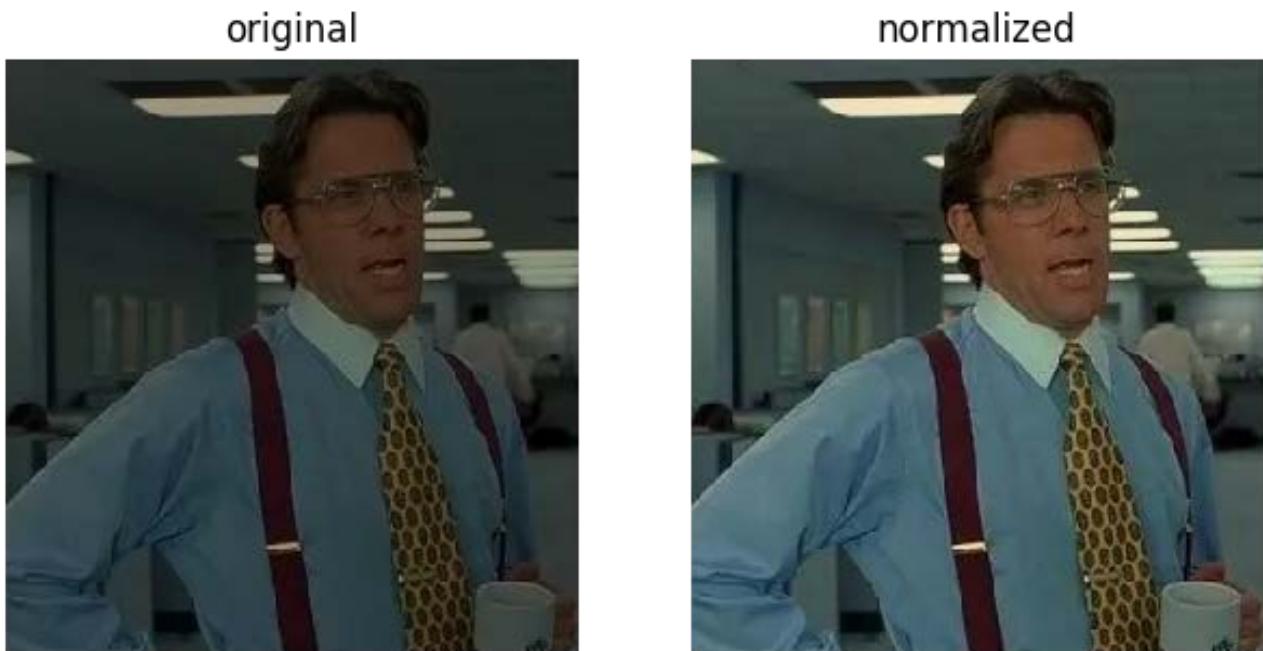
前两个图像似乎质量更高（但是您可以观察到一些压缩伪像）。线性方法的结果显然更平滑（没有对比度）并且噪点更少（黑白图像证明）。最后一个像素化。

归一化

我们可以使用[`normalize\(\)`](#)功能使视觉图像标准化，以修复非常暗/亮的图像（甚至可以修复低对比度）。该[归一化类型](#)是在函数参数指定：

```
norm_img = np.zeros((300, 300))  
norm_img = cv2.normalize(img, norm_img, 0, 255, cv2.NORM_MINMAX)
```

例子：



当使用图像作为深度卷积神经网络的输入时，无需应用这种归一化（上面的结果对我们来说似乎不错，但是并不针对他们的眼睛）。在实践中，我们将对每个通道进行适当的归一化，例如减去均值并除以像素级别的标准差（这样我们得到均值0和偏差1）。如果我们使用转移学习，最好的方法总是使用预先训练的模型统计信息。

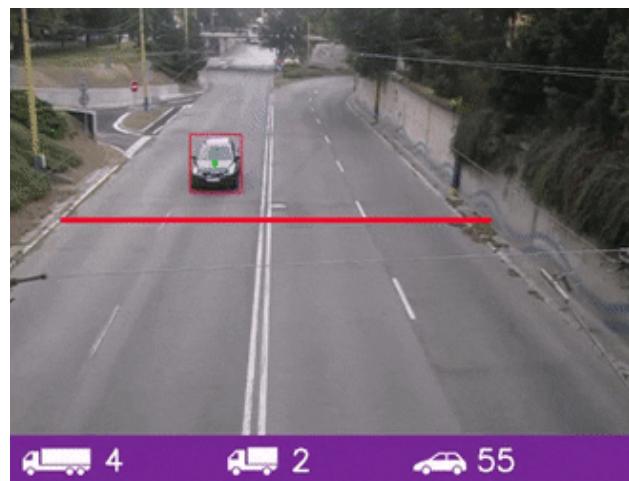
结论

当我们处理面部分类/识别问题时，如果输入的图像不是护照照片时，检测和分离面部是一项常见的任务。

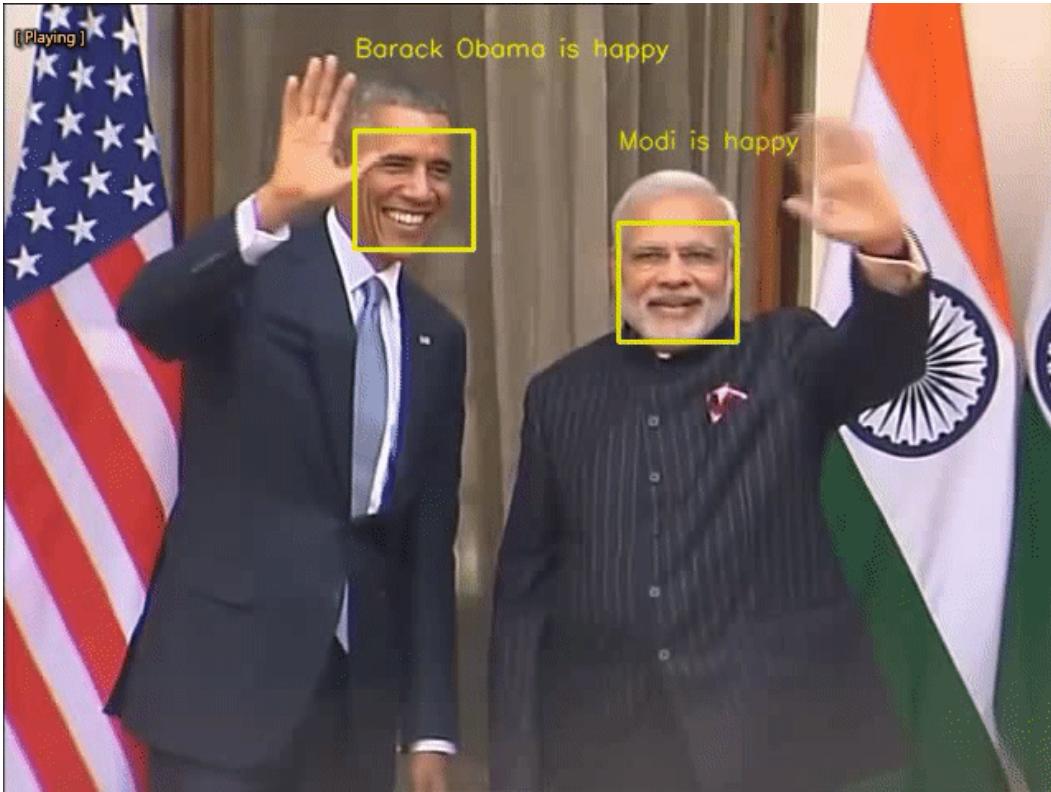
OpenCV是一个很好的图像预处理任务库，不仅限于此。对于许多计算机视觉应用来说，它也是一个很好的工具.....



<https://www.youtube.com/watch?v=GebcshN40dE>



https://www.youtube.com/watch?v=z1Cvn3_4yGo



<https://github.com/vjgpt/Face-and-Emotion-Recognition>

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM、三维视觉、传感器、自动驾驶、计算摄影、检测、分割、识别、医学影像、GAN、算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过。**添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





//
//

基于OpenCV和Tensorflow的深蹲检测器

原创 努比 小白学视觉 1周前

来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

本期我们将介绍如何使用OpenCV以及Tensorflow实现深蹲检测

在检疫期间，我们的体育活动非常有限，这样并不好。在进行一些居家运动时，我们必须时刻保持高度的注意力集中，以便记录自己每天的运动量。因此我们希望建立一个自动化的系统来实现运动量计算。考虑到我们在深蹲时，有明确阶段和大幅度变化的基本运动，实现对深蹲的计数会相对比较简单。

下面我们就一起尝试实现它吧！

数据采集

使用带相机的Raspberry Pi来获取图片是非常方便的，完成图像的拍摄后再利用OpenCV即可将获取的图像写入文件系统。

运动识别

最初，我们打算使用图像分割完成人物的提取工作。但是我们都清楚图像分割是一项非常繁琐的操作，尤其是在Raspberry资源有限的情况下。

除此之外，图像分割忽略了一个事实。当前我们所拥有的是一系列图像帧，而不是单个图片。该图像序列具有明显功能，并且我们后续将要使用到它。

因此，我们从OpenCV着手进行背景去除，以提供可靠的结果。

背景扣除

首先，创建一个背景减法器：

```
backSub = cv.createBackgroundSubtractorMOG2 ()
```

向其中添加图像帧：

```
mask = backSub.apply(frame)
```

最后我们可以得到一张带有身体轮廓的图片：



然后扩大图像以突出轮廓。

```
mask = cv.dilate(mask, None, 3)
```

将此算法应用于所有图像帧可以得出每一幅图像中的姿势。之后，我们将它们分类为站立，下蹲以及无三种情况。

接下来我们要把图像中的人提取出来，OpenCV可以帮助我们找到相应的找到轮廓：

```
cnts, _ = cv.findContours(img, cv.RETR_CCOMP, cv.CHAIN_APPROX_SIMPLE)
```

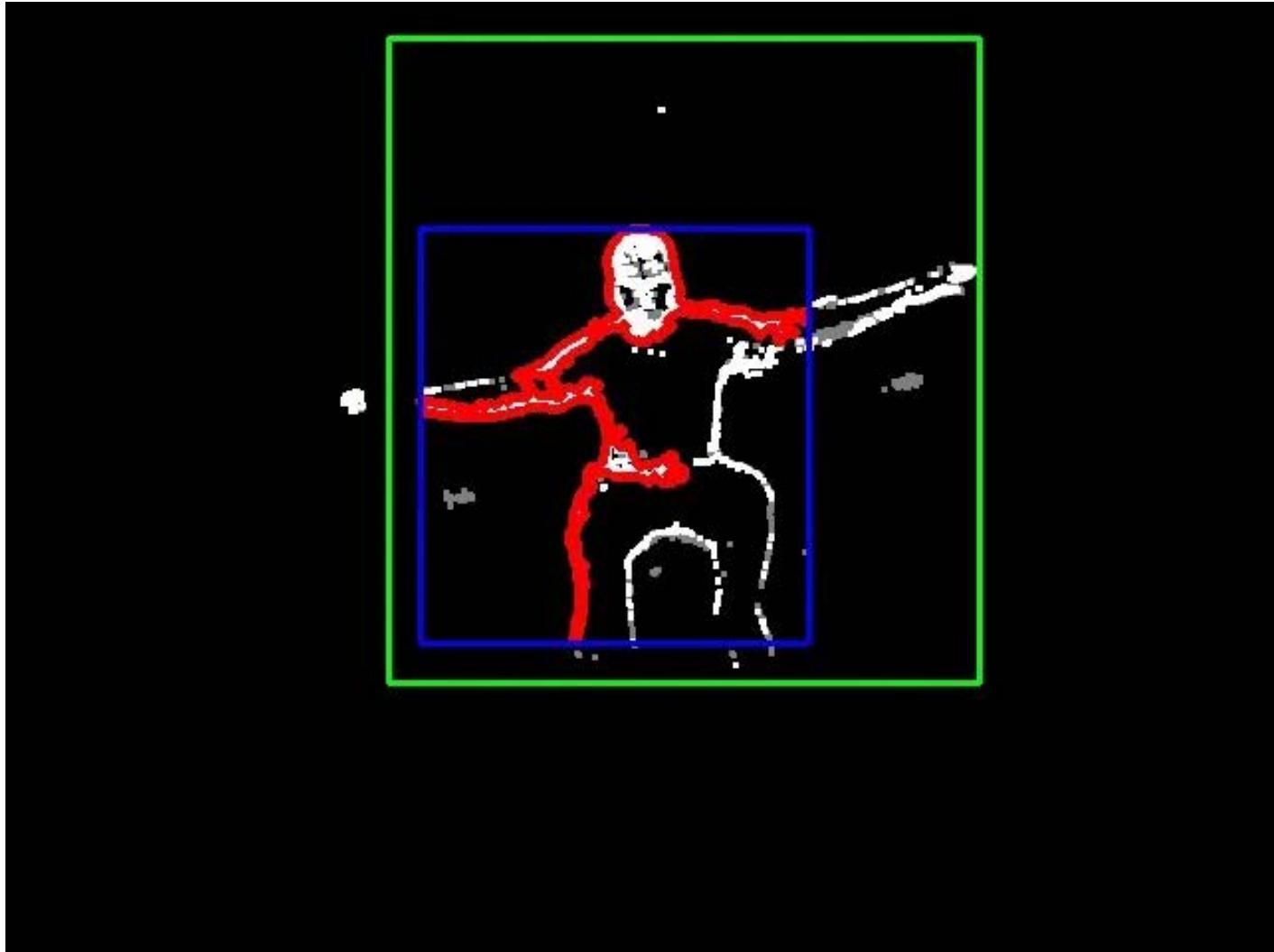
这种方法或多或少适用于人物的最大轮廓的提取，但不幸的是，这样处理的结果并不稳定。例如，检测得到最大的轮廓只能包括人的身体，而不包括他的脚。

但不管怎么说，拥有一系列图像对我很有帮助。通常情况下我们做深蹲运动都发生在同一地点，因此我们可以假设所有动作都在某个区域内进行并且该区域是稳定的。为此我们可以迭代构建边界矩形，如果需要，可以以最大轮廓增加边界矩形。

有一个例子：

- 最大的轮廓是红色

- 轮廓边界矩形为蓝色
- 图边界矩形为绿色



通过以上的边缘提取以及轮廓绘制，可以为进一步处理做好充足准备。

分类

接下来我们将从图像中提取出边界矩形，并将其转化为按尺寸 64×64 正方形。

以下Mask用作分类器输入：

站立姿势：



下蹲姿势：



接下来我们将使用Keras 与Tensorflow进行分类。

最初，我们使用了经典的Lenet-5模型，运行结果良好。随后由于阅读了一些有关Lenet-5变体的文章后，我们决定尝试简化架构。

事实证明，简化后的CNN在当前示例中的精度几乎相同：

```

1 model = Sequential([
2     Convolution2D(8,(5,5), activation='relu', input_shape=input_shape),
3     MaxPooling2D(),
4     Flatten(),
5     Dense(512, activation='relu'),
6     Dense(3, activation='softmax')
7 ])
8 model.compile(loss="categorical_crossentropy", optimizer=SGD(lr=0.01), metrics=[ "accu
  
```

10个纪元的准确度为86%，20个的准确度为94%，而30个的准确度为96%。训练如果在增加的话可能会导致过拟合引起准确度的下降，因此接下来我们将把这个模型运用到生活中去。

模型运用

我们将在Raspberry上运行。

加载模型：

```

1 with open(MODEL_JSON, 'r') as f:
2     model_data = f.read()
3     model = tf.keras.models.model_from_json(model_data)
4     model.load_weights(MODEL_H5)
5     graph = tf.get_default_graph()
  
```

并以此对下蹲Mask进行分类：

```
1 img = cv.imread(path + f, cv.IMREAD_GRAYSCALE)
2 img = np.reshape(img,[1,64,64,1])
3 with graph.as_default():
4     c = model.predict_classes(img)
5 return c[0] if c else None
```

在Raspberry上，输入为64x64的分类调用大约需要60-70毫秒，几乎接近实时。

最后让我们将以上所有部分整合到一个应用程序中：

- GET / ——一个应用页面（下面有更多信息）
- GET / status—获取当前状态，下蹲次数和帧数
- POST / start —开始练习
- POST / stop —完成练习
- GET / stream —来自摄像机的视频流

如果本文对小伙伴有帮助，希望可以在文末来个“一键三连”。

↑ 分享

赞 107 在看 128

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过**。添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





//
//

利用OpenCV实现基于深度学习的超分辨率处理

原创 小白 小白学视觉 5月24日

来自专辑

OpenCV应用

点击上方“**小白学视觉**”，选择“**星标**”公众号
重磅干货，第一时间送达

OpenCV是一个非常强大的计算机视觉处理的工具库。很多小伙伴在入门图像处理时都需要学习OpenCV的使用。但是随着计算机视觉技术的发展，越来越多的算法涌现出来，人们逐渐觉得OpenCV比较落后而放弃了使用OpenCV。

但是，实际上OpenCV时一个与时俱进的开源代码库。正在逐渐的吸收和接纳最新的算法。本文我们来介绍如何使用OpenCV实现基于深度学习的图像超分辨率（SR）。使用OpenCV的好处就是，我们不需要知道任何图像超分辨率的相关知识，就可以使用这个代码，并实现图像超分辨率。

具体操作步骤：

1. 安装OpenCV contrib模块

OpenCV中的超分辨率功能被集成在了contrib模块中，因此我们首先需要安装OpenCV的扩展模块。安装过程可以参考【从零学习OpenCV 4】opencv_contrib扩展模块的安装。超分辨率被集成在dnn_superres模块中，如果小伙伴们电脑空间有限，可以只编译这一个模块。

近期有小伙伴反馈自己安装扩展模块失败，为了解决这个问题，小白近期在筹划搭建一个各个版本opencv-contrib编译完成的数据库。各位小伙伴随时关注我们公众号的动态。

2. 下载训练的模型

由于某些模型比较大，因此OpenCV代码库中没有包含他们，因此我们在使用的时候需要单独的下载经过训练的模型。目前，仅支持4种不同的超分辨率模型，他们可以实现2倍、3倍、4倍甚至8倍的图像方法。这些模型具体如下：

EDSR: 这个是表现最好的模型。但是这个模型也是最大的，所以运行速度会比较慢。

ESPCN: 这个模型具有速度快，效果好的特点，并且模型较小。它可以进行对视频进行实时处理（取决于图像大小）。

FSRCNN: 这也是具有快速准确推断功能的小型模型。也可以进行实时视频升频。

LapSRN: 这是一个中等大小的模型，它的特点是最大可以将图像放大8倍。

公众号后台回复“**SR模型**”获取下载这四个模型的方式。

3. 通过程序实现超分辨率

我们首先给出C++完整程序，之后对程序中每一行代码进行介绍。完整程序如下：

```
1 #include <opencv2/dnn_superres.hpp>
2 #include <opencv2/imgproc.hpp>
3 #include <opencv2/highgui.hpp>
4
5 using namespace std;
6 using namespace cv;
7 using namespace dnn;
8 using namespace dnn_superres;
9
10 int main(int argc, char *argv[])
11 {
12     //Create the module's object
13     DnnSuperResImpl sr;
14
15     //Set the image you would like to upscale
16     string img_path = "image.png";
17     Mat img = cv::imread(img_path);
18
19     //Read the desired model
20     string path = "FSRCNN_x2.pb";
21     sr.readModel(path);
22
23     //Set the desired model and scale to get correct pre- and post-processing
24     sr.setModel("fsrcnn", 2);
25
26     //Upscale
27     Mat img_new;
28     sr.upsample(img, img_new);
```

```

29     cv::imwrite( "upscaled.png", img_new);
30
31     return 0;
32 }
```

首先加载我们选择的模型，并将其输入到神经网络的变量中。需要注意的是模型文件所存在的地址，本文放置在了程序的根目录中。

```

1 //Read the desired model
2 string path = "FSRCNN_x2.pb";
3 sr.readModel(path);
```

之后设置模型的种类和放大系数。本文选择的模型是fsrcnn，放大系数选择的2。

```

1 //Set the desired model and scale to get correct pre- and post-processing
2 sr.setModel("fsrcnn", 2);
```

可以选择的模型有“edsr”，“fsrcnn”，“lapsrn”，“espcn”，这几个参数就是我们刚才介绍的4中模型。需要注意的是，每个模型能够放大的倍数是不一致的。前三种模型能够放大2、3、4倍，最后一个模型能够放大2、3、4、8倍。

之后通过upsample()函数进行超分辨率放大。

```

1 //Upscale
2 Mat img_new;
3 sr.upsample(img, img_new);
4 cv::imwrite( "upscaled.png", img_new);
```

上述是C++代码，接下来给出Python实现超分辨率的代码

```

1 import cv2
2 from cv2 import dnn_superres
3
4 # Create an SR object
5 sr = dnn_superres.DnnSuperResImpl_create()
6
7 # Read image
8 image = cv2.imread('./input.png')
```

```
9  
10 # Read the desired model  
11 path = "EDSR_x3.pb"  
12 sr.readModel(path)  
13  
14 # Set the desired model and scale to get correct pre- and post-processing  
15 sr.setModel("edsr", 3)  
16  
17 # Upscale the image  
18 result = sr.upsample(image)  
19  
20 # Save the image  
21 cv2.imwrite("./upscaled.png", result)
```

不同于C++代码，在使用python代码时，需要先通过如下代码进行声明。

```
1 # Create an SR object  
2 sr = dnn_superres.DnnSuperResImpl_create()
```

4. 处理结果



输入图像



双线性插值放大3倍



FSRCNN放大3倍



ESDR放大3倍

聚集地
计算机视觉学者

小白学视觉

计算机视觉
论文解读 求职感想
SLAM技术 深度学习 学习感受

距离我们只差一个
长按关注

STEINWAY & SONS



//
//

使用TensorFlow物体检测模型、Python和OpenCV的社交距离检测器

原创 小鱼 小白学视觉 1周前

来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”
重磅干货，第一时间送达

0.介绍

疫情期间，我们在GitHub上搜索TensorFlow预训练模型，发现了一个包含25个**物体检测预训练模型**的库，并且这些预训练模型中包含其性能和速度指标。结合一定的计算机视觉知识，使用其中的模型来构建社交距离程序会很有趣。

学习OpenCV的过程中，小伙伴们应该知道对于一些小型项目OpenCV具有很强大的功能，其中一个就是对图片进行鸟瞰转换，鸟瞰图是对一个场景自上而下的表示，也是构建自动驾驶应用程序时经常需要执行的任务。

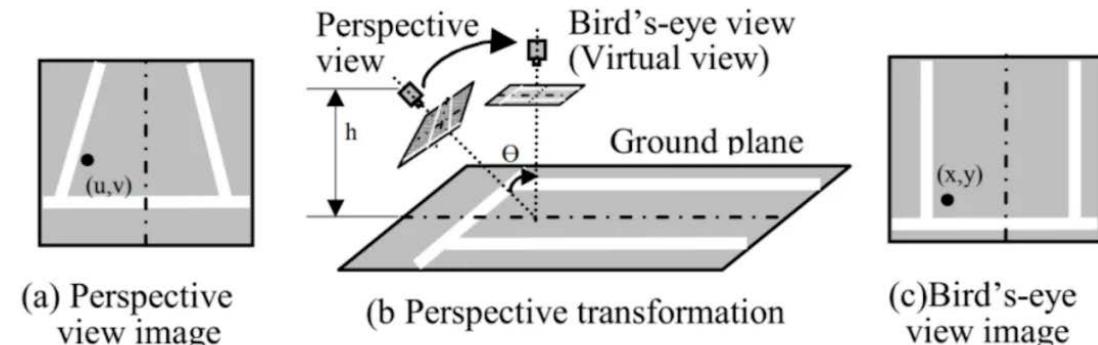


Fig. 1. Illustration of perspective transformation in a parking lot scene

车载摄像头鸟瞰系统的实现

这说明将鸟瞰转换的技术应用到监视社交距离的场景中可以提高监视质量。

本期我们将介绍了如何使用深度学习模型以及计算机视觉方面的一些知识来构建强大的社交距离检测器。

本文的结构如下：

- 模型选择
- 人员检测
- 鸟瞰图转换
- 社交距离测量

- 结果与改进

所有代码及安装说明可以以下链接中找到：<https://github.com/basileroth75/covid-social-distancing-detection>

1.模型选择

在TensorFlow物体检测模型zoo中的所有可用模型已经在**COCO数据集**（Context中的通用物体）上进行了预训练。COCO数据集包含120000张图像，这些图像中总共包含880000个带标签的物体。这些模型经过预训练可以检测90种不同类型的物体，物体类型的完整列表可以在GitHub的data部分得到，地址为：

https://github.com/tensorflow/models/blob/master/research/object_detection/data/mscoco_complete_label_map.pbtxt

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes

可用模型的非详尽清单

模型的预测速度不同，性能表现也不同。为了决定如何根据模型的预测速度来利用模型，我进行了一些测试。由于社交距离检测器的目标不是执行实时分析，因此最终选择了**fast_rcnn_inception_v2_coco**，它的mAP（验证集上检测器的性能）为28，执行速度为58ms，非常强大，下载地址为：

http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_v2_coco_2018_01_28.tar.gz

2.人员检测

使用上述模型检测人员，必须完成一些步骤：

- 将包含模型的文件加载到TensorFlow图中，并定义我们想从模型获得的输出。
- 对于每一帧，将图像输入到TensorFlow图以获取所需的输出。
- 过滤掉弱预测和不需要检测的物体。

加载并启动模型：

TensorFlow模型的工作方式是使用**graphs**(图)。第一步意味着将模型加载到TensorFlow图中，该图将包含所需检测。下一步是创建一个**session**（会话），该会话是负责执行定义在图中操作的一个实体。有关图和会话的更多说明，参见<https://danijar.com/what-is-a-tensorflow-session/>。在这里我们实现了一个类，将与TensorFlow图有关的所有数据关联在一起。

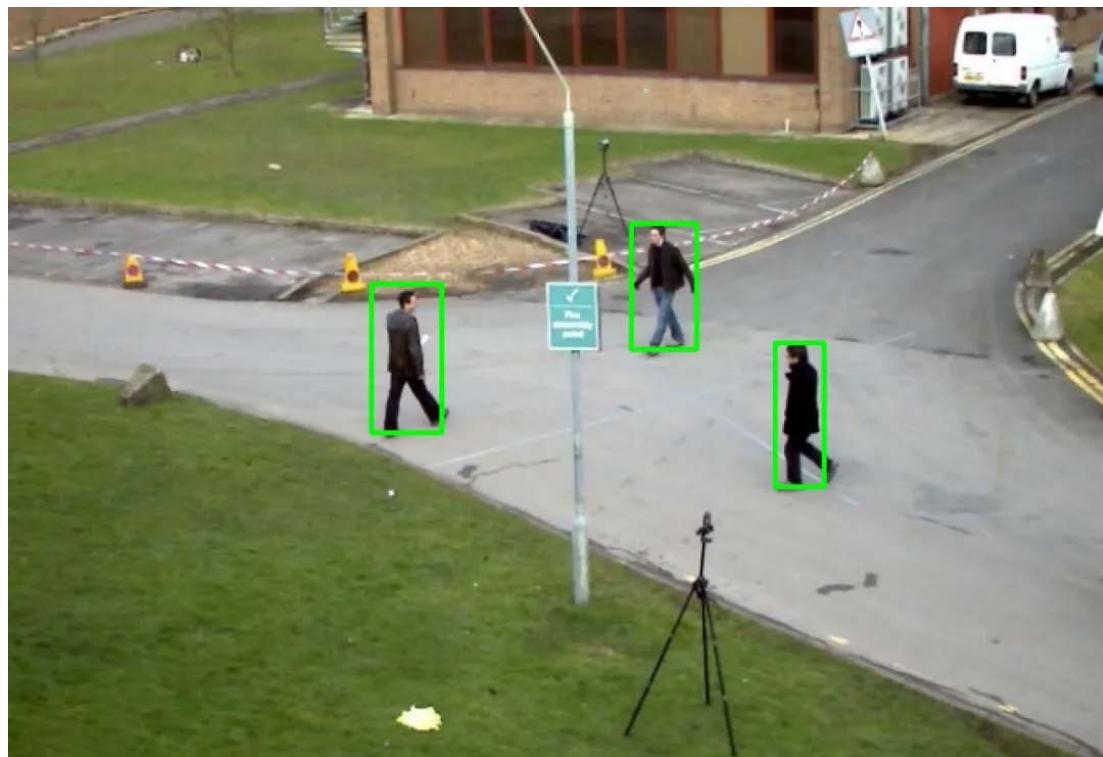
```
1 class Model:
```

```
2 """
3     Class that contains the model and all its functions
4 """
5 def __init__(self, model_path):
6 """
7     Initialization function
8     @ model_path : path to the model
9 """
10
11 # Declare detection graph
12         self.detection_graph = tf.Graph()
13 # Load the model into the tensorflow graph
14 with self.detection_graph.as_default():
15     od_graph_def = tf.compat.v1.GraphDef()
16 with tf.io.gfile.GFile(model_path, 'rb') as file:
17     serialized_graph = file.read()
18     od_graph_def.ParseFromString(serialized_graph)
19     tf.import_graph_def(od_graph_def, name='')
20
21 # Create a session from the detection graph
22         self.sess = tf.compat.v1.Session(graph=self.detection_graph)
23
24 def predict(self,img):
25 """
26     Get the prediction results on 1 frame
27     @ img : our img vector
28 """
29
30 # Expand dimensions since the model expects images to have shape: [1, None
31         img_exp = np.expand_dims(img, axis=0)
32 # Pass the inputs and outputs to the session to get the results
33         (boxes, scores, classes) = self.sess.run([self.detection_graph.get
return (boxes, scores, classes)
```

通过模型传递每一帧

对于需要处理的每个帧，都会启动一个新会话，这是通过调用**run ()** 函数完成的。这样做时必须指定一些参数，这些参数包括模型所需的输入类型以及我们要从中获取的输出。在我们的案例中所需的输出如下：

- 每个物体的边界框坐标
- 每个预测的置信度（0到1）
- 预测类别（0到90）
- 过滤弱预测和不相关物体



人员检测结果

模型能检测到的很多物体类别，其中之一是人并且与其关联的类为1。为了排除弱预测（**阈值：0.75**）和除人以外的所有其他类别的物体，我使用了if语句，将这两个条件结合起来以排除任何其他物体，以免进一步计算。

```
1 if int(classes[i]) == 1 and scores[i] > 0.75
```

但是因为这些模型已经经过预训练，不可能仅检测此类（人）。因此，**这些模型要花很长时间才能运行**，因为它们试图识别场景中所有90种不同类型的物体。

3.鸟瞰图转换

如引言中所述，执行鸟瞰图转换可为我们提供**场景的俯视图**。值得庆幸的是OpenCV具有强大的内置函数，此函数可以将从透视图角度拍摄的图像转换为俯视图。我使用了Adrian Rosebrock的教程来了解如何做到这一点：<https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/>

第一步选择原始图像上的4个点，这些点将成为要转换的图的角点。这些点必须形成一个矩形，至少两个相对的边平行，如果不这样做，则转换发生时的比例将不同。我已经在我的仓库中实现了一个

脚本，该脚本使用OpenCV的**setMouseCallback ()** 函数来获取这些坐标。计算变换矩阵的函数还需要使用图像的**image.shape**属性计算图像尺寸。

```
1 width, height, _ = image.shape
```

这将返回宽度、高度和其他不相关的颜色像素值。让我们看看如何使用它们计算变换矩阵：

```
1 def compute_perspective_transform(corner_points, width, height, image):
2     """ Compute the transformation matrix
3     @ corner_points : 4 corner points selected from the image
4     @ height, width : size of the image
5     return : transformation matrix and the transformed image
6     """
7     # Create an array out of the 4 corner points
8     corner_points_array = np.float32(corner_points)
9     # Create an array with the parameters (the dimensions) required to build t
10    img_params = np.float32([[0,0],[width,0],[0,height],[width,height]])
11    # Compute and return the transformation matrix
12    matrix = cv2.getPerspectiveTransform(corner_points_array, img_params)
13    img_transformed = cv2.warpPerspective(image, matrix, (width, height))
14    return matrix, img_transformed
```

注意函数的返回值是矩阵，因为在下一步中将使用这个矩阵计算每个被检测到的人的新坐标，新坐标是帧中每个人的“**GPS**”坐标，使用这些新坐标而不是使用原始基点结果**更为准确**，因为在透视图中当人们处于不同平面时，距离是不一样的，并且距相机的距离也不相同。与使用原始检测框中的点相比，这可以大大改善社会距离的测量。

对于检测到的每个人，将返回构建边界框所需的2个点，这两个点是边界框的左上角和右下角。通过获取两点之间的中点来计算**边界框的质心**，使用此结果，计算位于边界框底部中心的点的坐标，我认为这一点（称为“**基点**”）是图像中人坐标的最佳表示。

然后使用变换矩阵为每个检测到的基点计算变换后的坐标。在检测到人之后，在每一帧上使用**cv2.perspectiveTransform ()** 完成此操作。实现此任务的方式：

```
1 def compute_point_perspective_transformation(matrix, list_downoids):
2     """ Apply the perspective transformation to every ground point which have
3     @ matrix : the 3x3 matrix
```

```

4 @ list_downoids : list that contains the points to transform
5 return : list containing all the new points
6 """
7 # Compute the new coordinates of our points
8 list_points_to_detect = np.float32(list_downoids).reshape(-1, 1, 2)
9 transformed_points = cv2.perspectiveTransform(list_points_to_detect, mat
10 # Loop over the points and add them to the list that will be returned
11 transformed_points_list = list()
12 for i in range(0,transformed_points.shape[0]):
13     transformed_points_list.append([transformed_points[i][0][0],transforme
14 return transformed_points_list

```

4.社交距离测量

在每帧上调用此函数后，将返回一个包含所有新转换点的列表，从这个列表中，计算每对点之间的距离。这里我使用了`itertools`库的**Combination ()** 函数，该函数允许在列表中获取所有可能的组合而无需保留双精度。在<https://stackoverflow.com/questions/104420/how-to-generate-all-permutations-of-a-list> 堆栈溢出问题中对此进行了很好的解释。其余的是简单的数学运算：使用`math.sqrt ()` 函数计算两点之间的距离。选择的阈值为120像素，因为它在我们的场景中大约等于2英尺。

```

1 # Check if 2 or more people have been detected (otherwise no need to detec
2 if len(transformed_downoids) >= 2:
3     # Iterate over every possible 2 by 2 between the points combinations
4     list_indexes = list(itertools.combinations(range(len(transformed_downoids))
5     for i,pair in enumerate(itertools.combinations(transformed_downoids, r=2))
6         # Check if the distance between each combination of points is less t
7     if math.sqrt( (pair[0][0] - pair[1][0])**2 + (pair[0][1] - pair[1][1])**2
8         # Change the colors of the points that are too close from each oth
9     change_color_topview(pair)
10        # Get the equivalent indexes of these points in the original frame
11 index_pt1 = list_indexes[i][0]
12 index_pt2 = list_indexes[i][1]
13 change_color_originalframe(index_pt1,index_pt2)

```

一旦确定两个点之间的距离太近，标记该点的圆圈的颜色将从绿色更改为红色，原始框架上的边界框的颜色也做相同颜色变换操作。

5.结果

回顾项目的工作原理：

- 首先获取图的4个角点，然后应用透视变换获得该图的鸟瞰图并保存透视变换矩阵。
- 获取原始帧中检测到的每个人的边界框。
- 计算这些框的最低点，最低点是位于人双脚之间的点。
- 对这些点应用变换矩阵，获取每一个人的真实“GPS”坐标。
- 使用`itertools.combinations()`测量帧中每个点到所有其它点的距离。
- 如果检测到违反社交距离，将边框的颜色更改为红色。

我使用来自PETS2009 数据集<http://www.cvg.reading.ac.uk/PETS2009/a.html#s0> 的视频，该视频由包含不同人群活动的多传感器序列组成，它最初是为诸如人群中人员计数和密度估计之类的任务而构建的。我决定从第一个角度使用视频，因为它是最宽的一个，具有最佳的场景视角。该视频介绍了获得的结果：

https://youtu.be/3b2GPwN2_l0

6.结论与改进

如今，隔离以及其他基本卫生措施对抑制Covid-19的传播速度非常重要。但该项目仅是概念的证明，并且由于道德和隐私问题，不能用于监视公共或私人区域的社交距离。

这个项目存在一些小的缺陷，改进思路如下：

- 使用更快的模型来执行实时社交距离分析。
- 使用对遮挡更具鲁棒性的模型。
- 自动校准是计算机视觉中一个众所周知的问题，可以在不同场景上极大地改善鸟瞰图的转换。

7.参考资料

<https://towardsdatascience.com/analyse-a-soccer-game-using-tensorflow-object-detection-and-opencv-e321c230e8f2>

<https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/>

https://developer.ridgerun.com/wiki/index.php?title=Birds_Eye_View/Introduction/Research

<http://www.cvg.reading.ac.uk/PETS2009/a.html#s0>

如果本文对小伙伴有帮助，希望可以在文末来个“一键三连”。



交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM**、**三维视觉**、**传感器**、**自动驾驶**、**计算摄影**、**检测**、**分割**、**识别**、**医学影像**、**GAN**、**算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过。**添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~



//
//

使用深度学习和OpenCV的早期火灾检测系统

原创 花生 小白学视觉 1周前

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

创建用于室内和室外火灾检测的定制InceptionV3和CNN架构。



嵌入式处理技术的最新进展已使基于视觉的系统可以在监视过程中使用卷积神经网络检测火灾。在本文中，两个定制的CNN模型已经实现，它们拥有用于监视视频的高成本效益的火灾检测CNN架构。第一个模型是受AlexNet架构启发定制的基本CNN架构。我们将实现和查看其输出和限制，并创建一个定制的InceptionV3模型。为了平衡效率和准确性，考虑到目标问题和火灾数据的性质对模型进行了微调。我们将使用三个不同的数据集来训练我们的模型。

创建定制的CNN架构

我们将使用TensorFlow API Keras构建模型。首先，我们创建用于标记数据的ImageDataGenerator。[1] 和[2]数据集在这里用于训练。最后，我们将提供980张图像用于训练和239张图像用于验证。我们也将使用数据增强。

```

1 import tensorflow as tf
2 import keras.preprocessing
3 from keras.preprocessing import image
4 from keras.preprocessing.image import ImageDataGenerator
5 TRAINING_DIR = "Train"
6 training_datagen = ImageDataGenerator(rescale = 1./255,
7                                         horizontal_flip=True,
8                                         rotation_range=30,
9                                         height_shift_range=0.2,
10                                        fill_mode='nearest')
11 VALIDATION_DIR = "Validation"
12 validation_datagen = ImageDataGenerator(rescale = 1./255)
13 train_generator = training_datagen.flow_from_directory(TRAINING_DIR,
14                                                       target_size=(224,224),
15
16 class_mode='categorical',
17                                         batch_size = 64)
18 validation_generator = validation_datagen.flow_from_directory(
19                                         VALIDATION_DIR,
20                                         target_size=(224,224),
21
22 class_mode='categorical',
23                                         batch_size= 16)

```

在上面的代码中应用了3种数据增强技术，它们分别是水平翻转，旋转和高度移位。

现在，我们将创建我们的CNN模型。该模型包含三对Conv2D-MaxPooling2D层，然后是3层密集层。为了克服过度拟合的问题，我们还将添加dropout层。最后一层是softmax层，它将为我们提供火灾和非火灾两类的概率分布。通过将类数更改为1，还可以在最后一层使用‘Sigmoid’激活函数。

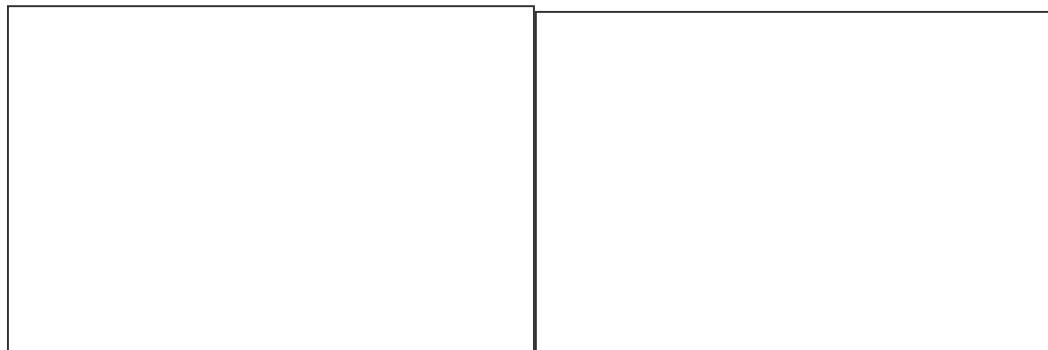
```

1 from tensorflow.keras.optimizers import Adam
2 model = tf.keras.models.Sequential([
3     tf.keras.layers.Conv2D(96, (11,11), strides=(4,4), activation='relu', input_shape=(2
4     tf.keras.layers.Conv2D(256, (5,5), activation='relu'),
5     tf.keras.layers.MaxPooling2D(pool_size = (3,3), strides=(2,2)),

```

```
6 tf.keras.layers.Conv2D(384, (5,5), activation='relu'),  
7 tf.keras.layers.MaxPooling2D(pool_size = (3,3), strides=(2,2)),  
8 tf.keras.layers.Flatten(),  
9 tf.keras.layers.Dropout(0.2),  
10 tf.keras.layers.Dense(2048, activation='relu'),  
11 tf.keras.layers.Dropout(0.25),  
12 tf.keras.layers.Dense(1024, activation='relu'),  
13 tf.keras.layers.Dropout(0.2),  
14 tf.keras.layers.Dense(2, activation='softmax')])model.compile(loss='categorical_cros  
15 optimizer=Adam(lr=0.0001),  
16 metrics=['acc'])history = model.fit(  
17 train_generator,  
18 steps_per_epoch = 15,  
19 epochs = 50,  
20 validation_data = validation_generator,  
21 validation_steps = 15  
22 )
```

我们将使用Adam作为学习率为0.0001的优化器。经过50个时期的训练，我们得到了96.83的训练精度和94.98的验证精度。训练损失和验证损失分别为0.09和0.13。



我们的训练模型

让我们测试模型中的所有图像，看看它的猜测是否正确。为了进行测试，我们选择了3张图像，其中包括有火的图像，没有火的图像以及包含火样颜色和阴影的照片。

我们最终得到上面创建的模型在对图像进行分类时犯了一个错误。该模型52%的把握确定图像中有火焰。这是因为已进行训练的数据集中几乎没有图像可以说明室内火灾的模型。所以该模型仅知道室外火灾情况，而当给出一张室内火样的阴影图像时会出现错误。另一个原因是我们的模型不具备可以学习火的复杂特征。

接下来，我们将使用标准的InceptionV3模型并对其进行自定义。复杂模型能够从图像中学习复杂特征。

创建定制的InceptionV3模型

这次我们将使用不同的数据集[3]，其中包含室外和室内火灾图像。我们已经在该数据集中训练了我们之前的CNN模型，结果表明它是过拟合的，因为它无法处理这个相对较大的数据集和从图像中学习复杂的特

征。

我们开始为自定义的InceptionV3创建ImageDataGenerator。数据集包含3个类，但对于本文，我们将仅使用2个类。它包含用于训练的1800张图像和用于验证的200张图像。另外，我添加了8张客厅图像，以在数据集中添加一些噪点。

```
1 import tensorflow as tf
2 import keras.preprocessing
3 from keras.preprocessing import image
4 from keras.preprocessing.image import ImageDataGenerator
5 TRAINING_DIR = "Train"
6 training_datagen = ImageDataGenerator(rescale=1./255,
7 zoom_range=0.15,
8 horizontal_flip=True,
9 fill_mode='nearest')
VALIDATION_DIR = "/content/FIRE-SMOKE-DATASET/Test"
10 validation_datagen = ImageDataGenerator(rescale = 1./255)
11 train_generator = training_datagen.flow_from_directory(
12 TRAINING_DIR,
13 target_size=(224, 224),
14 shuffle = True,
15 class_mode='categorical',
16 batch_size = 128)
17 validation_generator = validation_datagen.flow_from_directory(
18 VALIDATION_DIR,
19 target_size=(224, 224),
20 shuffle = True,
21 class_mode='categorical',
22 batch_size= 14)
```

为了使训练更加准确，我们可以使用数据增强技术。在上面的代码中应用了2种数据增强技术-水平翻转和缩放。

让我们从Keras API导入InceptionV3模型。我们将在InceptionV3模型的顶部添加图层，如下所示。我们将添加一个全局空间平均池化层，然后是2个密集层和2个dropout层，以确保我们的模型不会过拟合。最后，我们将为2个类别添加一个softmax激活的密集层。

接下来，我们将首先仅训练我们添加并随机初始化的图层。我们将在这里使用RMSprop作为优化器。

```
1 from tensorflow.keras.applications.inception_v3 import InceptionV3
2 from tensorflow.keras.preprocessing import image
3 from tensorflow.keras.models import Model
4 from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input, Dropout
5 base_model = InceptionV3(input_tensor=input_tensor, weights='imagenet', include_top=False)
6 x = GlobalAveragePooling2D()(x)
```

```

7 x = Dense(2048, activation='relu')(x)
8 x = Dropout(0.25)(x)
9 x = Dense(1024, activation='relu')(x)
10 x = Dropout(0.2)(x)
11 predictions = Dense(2, activation='softmax')(x)
model = Model(inputs=base_model.input
12 layer.trainable = False
model.compile(optimizer='rmsprop', loss='categorical_crosse
13 train_generator,
14 steps_per_epoch = 14,
15 epochs = 20,
16 validation_data = validation_generator,
17 validation_steps = 14)

```

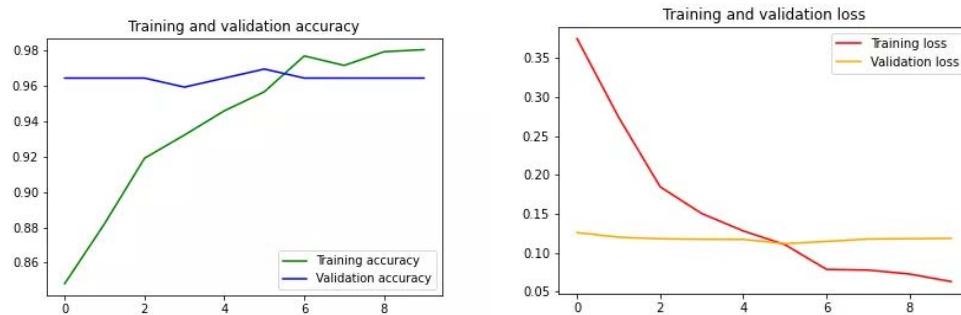
在训练了顶层20个周期之后，我们将冻结模型的前249层，并训练其余的层（即顶层2个初始块）。在这里，我们将使用SGD作为优化器，学习率为0.0001。

```

1 #To train the top 2 inception blocks, freeze the first 249 layers and unfreeze the re
2 layer.trainable = False
for layer in model.layers[249:]:
3     layer.trainable = True
#Recompile the model for these modifications to take effect
4 model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy'
5 train_generator,
6 steps_per_epoch = 14,
7 epochs = 10,
8 validation_data = validation_generator,
9 validation_steps = 14)

```

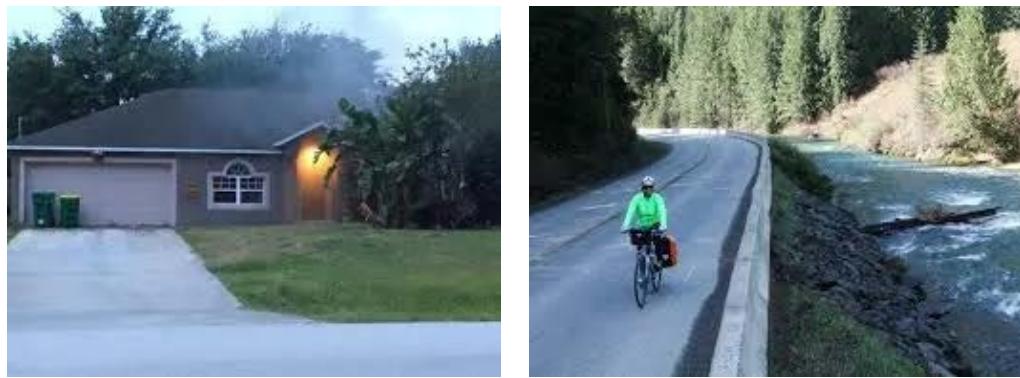
经过10个周期的训练，我们获得了98.04的训练准确度和96.43的验证准确度。训练损失和验证损失分别为0.063和0.118。



以上10个时期的训练过程

我们用相同的图像测试我们的模型，看看是否它可以正确猜出。

这次我们的模型可以使所有三个预测正确。96%的把握可以确定图像中没有任何火。我用于测试的其他两个图像如下：



来自下面引用的数据集中的非火灾图像

实时测试

现在，我们的模型已准备好在实际场景中进行测试。以下是使用OpenCV访问我们的网络摄像头并预测每帧图像中是否包含火的示例代码。如果框架中包含火焰，我们希望将该框架的颜色更改为B&W。

```

1 import cv2
2 import numpy as np
3 from PIL import Image
4 import tensorflow as tf
5 from keras.preprocessing import image#Load the saved model
6 model = tf.keras.models.load_model('InceptionV3.h5')
7 video = cv2.VideoCapture(0)while True:
8     _, frame = video.read()#Convert the captured frame into RGB
9     im = Image.fromarray(frame, 'RGB')#Resizing into 224x224 because we trained
10    im = im.resize((224,224))
11    img_array = image.img_to_array(im)
12    img_array = np.expand_dims(img_array, axis=0) / 255
13    probabilities = model.predict(img_array)[0]
14    #Calling the predict method on model to predict 'fire' on the image
15    prediction = np.argmax(probabilities)
16    #if prediction is 0, which means there is fire in the frame.
17    if prediction == 0:
18        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
19        print(probabilities[prediction])cv2.imshow("Capturing", frame)
20        key=cv2.waitKey(1)
21        if key == ord('q'):
22            break
23    video.release()
24    cv2.destroyAllWindows()

```

这个项目的Github链接在这里：<https://github.com/DeepQuestAI/Fire-Smoke-Dataset>。可以从那里找到数据集和上面的所有代码。



结论

使用智能相机可以识别各种可疑事件，例如碰撞，医疗紧急情况和火灾。其中，火灾是最危险的异常事件，因为在早期阶段无法控制火灾会导致巨大的灾难，从而造成人员，生态和经济损失。受CNN巨大潜力的启发，我们可以在早期阶段从图像或视频中检测到火灾。本文展示了两种用于火灾探测的自定义模型。考虑到CNN模型的火灾探测准确性，它可以帮助灾难管理团队按时管理火灾，从而避免巨额损失。

本文使用的数据集：

数据集1：<https://www.kaggle.com/atulyakumar98/test-dataset>

数据集2：<https://www.kaggle.com/phylake1337/fire-dataset>

数据集3: <https://github.com/DeepQuestAI/Fire-Smoke-Dataset>

如果本文对小伙伴有帮助，希望可以在文末来个“一键三连”。

分享

赞 107

在看 128

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM、三维视觉、传感器、自动驾驶、计算摄影、检测、分割、识别、医学影像、GAN、算法竞赛等微信群**（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过。**添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~



 小白掌视觉

计算机视觉
论文解读 求职感想
SLAM技术 深度学习 学习感受

聚集地
计算机视觉学者

距离我们只差一个
长按关注



STEINWAY & SONS

//
//

使用网络摄像头和Python中的OpenCV构建运动检测器(Translate)

原创 小林 小白学视觉 7月14日

来自专辑

OpenCV应用

点击上方“[小白学视觉](#)”，选择加“[星标](#)”或“[置顶](#)”

重磅干货，第一时间送达

本期我们将学习如何使用OpenCV实现运动检测

运动检测是指检测物体相对于周围环境的位置是否发生了变化。接下来，让我们一起使用Python实现一个运动检测器应用程序吧！

该运动检测器可以完成以下任务：

- 1) 在家工作时在屏幕前查找时间
- 2) 监控孩子在屏幕前的时间
- 3) 在你的后院发现非法侵入
- 4) 在你的房间/房子/小巷周围找到不需要的公共/动物活动.....。



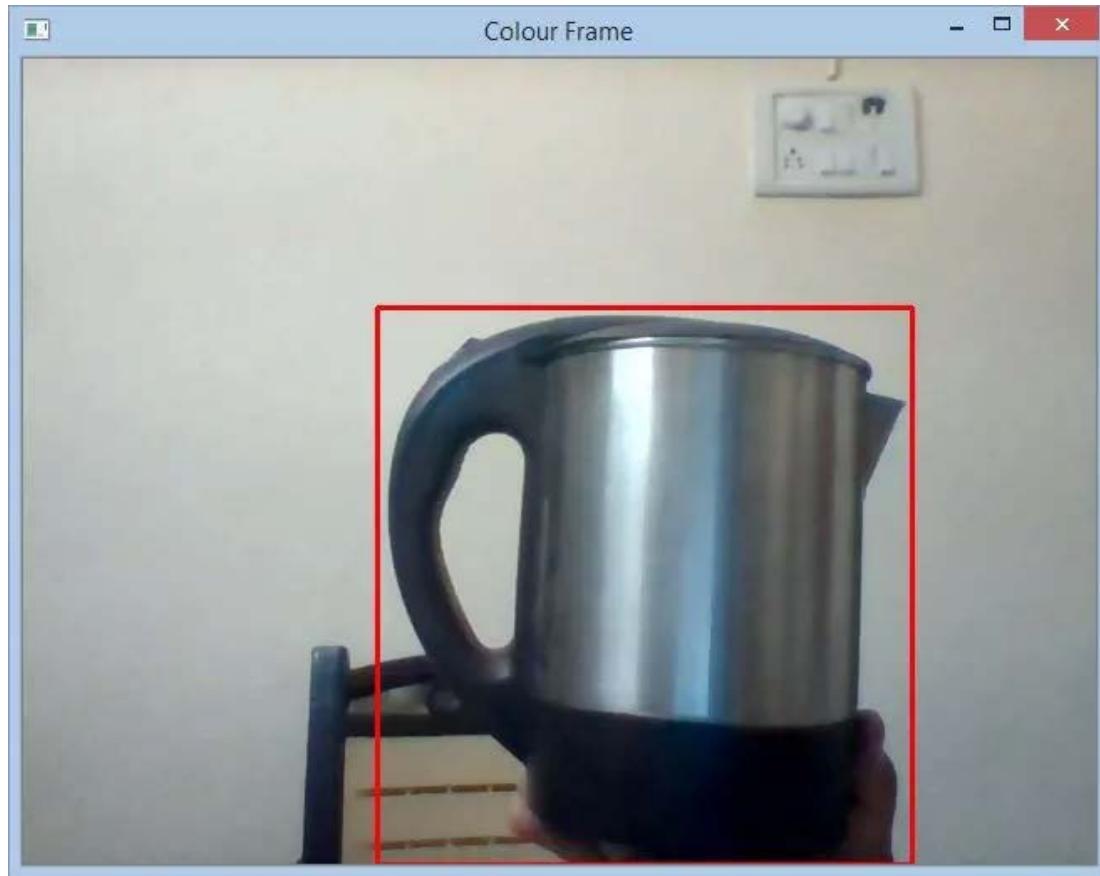
想要实现该运动检测器程序我们需要具备以下条件：

- 1) 硬件要求：装有网络摄像机或任何类型摄像机的计算机。
- 2) 软件需求：Python3或者更高版本。

3) 附加要求：对运动检测有一定的兴趣。

接下来我们将一步步的完成该应用程序的构建。

首先，我们将通过网络摄像头捕获第一帧，并将它视为基准帧，如下图所示。通过计算该基准帧中的对象与新帧对象之间的相位差来检测运动。我们也将得到的结果称为Delta帧。



接下来，我们将使用像素强度来优化Delta帧，优化后的帧称为阈值帧。并且，我们将应用一些复杂的图像处理技术，例如阴影消除、扩张轮廓等，以完成在阈值帧上提取对象物体。以下是您要实现的目标：

被探测对象

当这个对象进入帧和退出帧时，我们能够很容易的捕获这两帧的时间戳。因此，将能够准确的在视频中找到相关片段。

我们希望小伙伴都能自己实现这个程序，因此我们就不直接嵌入代码了。

从最基本的安装开始，我们需要安装Python3或更高版本，并使用pip安装pandas和OpenCV这两个库。这些工作做好，我们的准备工作就完成了。

第一步：导入需要的库：

第二步：初始化变量，列表， data frame：

在下面的代码中，我们将会了解到在什么时候需要使用上面涉及到的每一项。

第三步：使用网络摄像机捕获视频帧：

```
9  video=cv2.VideoCapture(0)
10
```

在OpenCV中有能够打开相机并捕获视频帧的内置函数。其中输入参数“0”表示计算机硬件端口号为0的摄像机。如果我们拥有了多个摄像头或闭路电视等设置，可以通过该参数提供相应的端口号。

第四步：将捕捉到的帧转换为灰度图像，并应用高斯模糊去除噪声：

```
11 while True:
12     check,color_frame=video.read()
13     status=0
14     gray=cv2.cvtColor(color_frame,cv2.COLOR_BGR2GRAY)
15     gray=cv2.GaussianBlur(gray,(21,21),0)
16
```

由于彩色图片中每个像素均具有三个颜色通道，实际上我们并不需要使用这么多的信息，因此首先将彩色帧转换成灰度帧。再利用高斯模糊对图像进行平滑处理，进而提高检测精度。在高斯模糊函数中，我们利用第2个参数定义了高斯核的宽度和高度；利用第3个参数，定义了标准偏差值。在这里我们可以使用核大小为 (21,21)，标准偏差为0的标准值。想要了解有关高斯平滑的更多信息，请参考：

[Smoothing Images - OpenCV 2.4.13.7 documentation](#) In an analogous way as the Gaussian filter, the bilateral filter also considers the neighboring pixels with weights...
[docs.opencv.org](#)

第五步：捕获第一个灰度帧

```
17     if first_frame is None:
18         first_frame=gray
19         continue
```

第一帧是整个处理过程中的基准帧。通过计算此基准帧与新帧之间特定对象的相位差来检测运动。在拍摄第一帧时，特定对象相机前不应有任何移动。但是得到的第一帧并不需要后续处理，因此我们可以用continue语句跳过后续过程。

第六步：创建Delta帧和阈值帧

```
21     delta_frame=cv2.absdiff(first_frame,gray)
22     thresh_frame=cv2.threshold(delta_frame,30,255,cv2.THRESH_BINARY)[1]
23
```

现在，我们需要找出第一帧和当前帧之间的区别。因此，我们使用absdiff函数并将得到的结果称为delta帧。对于我们的用例来说，仅仅找到一个差异是不够的，所以我们需要定义一个像素阈值，它可以被视为真实的对象。

我们可以选择30像素作为标准阈值，并将标准阈值的颜色定义为白色（颜色代码：255）。二元阈值函数THRESH_BINARY返回一个元组值，其中只有第二项（[0]是第一项，[1]是第二项）包含生成的阈值帧。二元阈值函数用于处理含有2个离散值的非连续函数：如0或1。如果摄影机前面没有对象，我们将当前帧的状态视为0；如果摄影机前面存在对象，则将当前帧的状态视为1。

更多阈值图像处理相关知识，请参考：

[Miscellaneous Image Transformations - OpenCV 2.4.13.7 documentation](#) Performs a marker-based image segmentation using the watershed algorithm. The function implements one of the variants...
[docs.opencv.org](#)

第七步：膨胀阈值帧并在其中找到轮廓像素

```

24     thresh_frame=cv2.dilate(thresh_frame,None,iterations=3)
25
26     (cnts,_) =cv2.findContours(thresh_frame.copy(),cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
27

```

“我们的眼睛总是被光线吸引，但阴影处有更多内容。”—格雷戈里·马奎尔

对象的每个部分都会在背景或自身的其他部分留下一定的阴影。这似乎总是让我们感到很困惑。例如，鼻子投射在嘴唇上的阴影，较大的静止物体在旁边的小物体上投射的阴影。飘动的光源，不同发光强度的多个光源，你房间的窗帘，光源的方向和视角等等都会对阴影造成一定的影响。

以下是在实时捕获的帧中发现的一些干扰。因此，为了使这些噪声最小化，我们需要对图像进行滤波。在膨胀函数Dilate中，我们可以通过设置迭代次数来设置平滑度。迭代次数越多，平滑度越高，处理时间也就越长。因此，建议保持标准化设置为3。膨胀函数中的“None”参数表示我们的应用中不需要元素结构。

关于膨胀的更多知识，你可以参考：

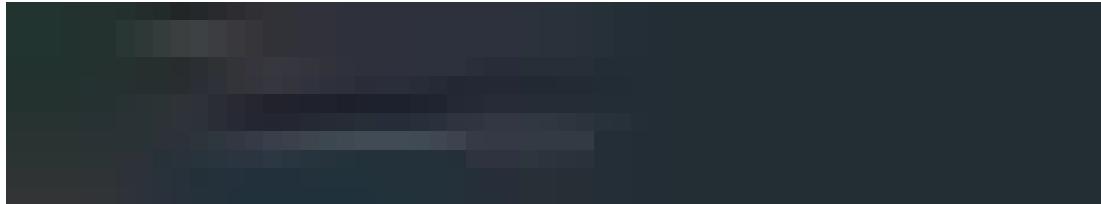
[Image Filtering - OpenCV 2.4.13.7 documentation](#) Functions and classes described in this section are used to perform various linear or non-linear filtering operations...
[docs.opencv.org](#)

完成过滤以后，我们需要在该帧中找到对象轮廓。我们用当前帧中的轮廓来识别对象的大小和位置。为了实现这一点，我们将该帧的一个副本传递到findCounters方法中，使用这个副本来自查轮廓。使用副本的原因是，我们不希望轮廓识别影响到原始过滤帧。

这里有个麻烦，因为我们必须将轮廓存储在一个元组中，并且只需要使用该元组的第一个值。请参阅Python3中声明元组的语法：(name, _)。

现在，我们只需要在过滤层上找到对象的外部轮廓。对于我们的用例来说，除了极端外部轮廓以外的其他轮廓都是无用的。因此我们必须使用一些近似方法来优化轮廓的提取过程。例如使用曲线近似或

曲线插值，也可以使用简单链近似规则，即压缩水平、垂直和对角线线段，只保留其端点。因此，我们能够很快得到最佳拟合轮廓。



第八步：找到轮廓区域，并在矩形中形成端点：

```

28     for contour in cnts:
29         if cv2.contourArea(contour)<10000:
30             continue
31         status=1
32         (x,y,w,h)=cv2.boundingRect(contour)
33         cv2.rectangle(color_frame,(x,y),(x+w,y+h),(0,0,255),2)
34     #for Loop ends here

```

实际上我们并不想捕捉像昆虫这样的小物体，而是要捕捉像人或动物这样的大物体。因此我们采用轮廓区域的概念，即跳过那些面积小于10000像素的对象。对于大于此区域的轮廓，我们将状态设置为1，即检测到对象。

想知道关于图像处理中的轮廓，可以参考：

[Structural Analysis and Shape Descriptors - OpenCV 2.4.13.7 documentation Draws contours outlines or filled contours. The function draws contour outlines in the image if or fills the area...](https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles.html)
docs.opencv.org

现在我们使用boundingRect函数捕捉轮廓的坐标。然后，我们使用这些坐标在彩色帧上绘制一个特定颜色、特定厚度的矩形。此矩形描述了实际检测到的对象。

第九步：捕获对象进入帧（场景）和退出帧（场景）时的时间戳

```

38     status_list.append(status)
39
40     if status_list[-1]==1 and status_list[-2]==0:
41         time_stamp.append(datetime.now())
42     if status_list[-1]==0 and status_list[-2]==1:
43         time_stamp.append(datetime.now())

```

“状态”列表status_list存储值0：代表未检测到对象，1：代表检测到对象。此状态值从0更改为1的时刻就是对象进入帧的那一时刻。同样，此状态值从1变为0的时刻就是对象从帧中消失的那一时刻。因此，我们从状态列表的最后两个值可以获得这两个切换事件的时间戳。

第十步：显示所有不同的画面（帧）

```

43     cv2.imshow("Gray Frame",gray)
44     cv2.imshow("Delta Frame",delta_frame)
45     cv2.imshow("Threshold Frame",thresh_frame)
46     cv2.imshow("Colour Frame",color_frame)
47

```

使用imshow（）方法，我们将在一个独立的窗口中显示每个帧并进行比较。



我们使用waitKey函数来延迟进程，直到按下某个键。在这里，我们使用waitKey（1）从摄像机获得连续的实时反馈。想停止拍摄视频时，只需按键盘上的“Q”键即可。

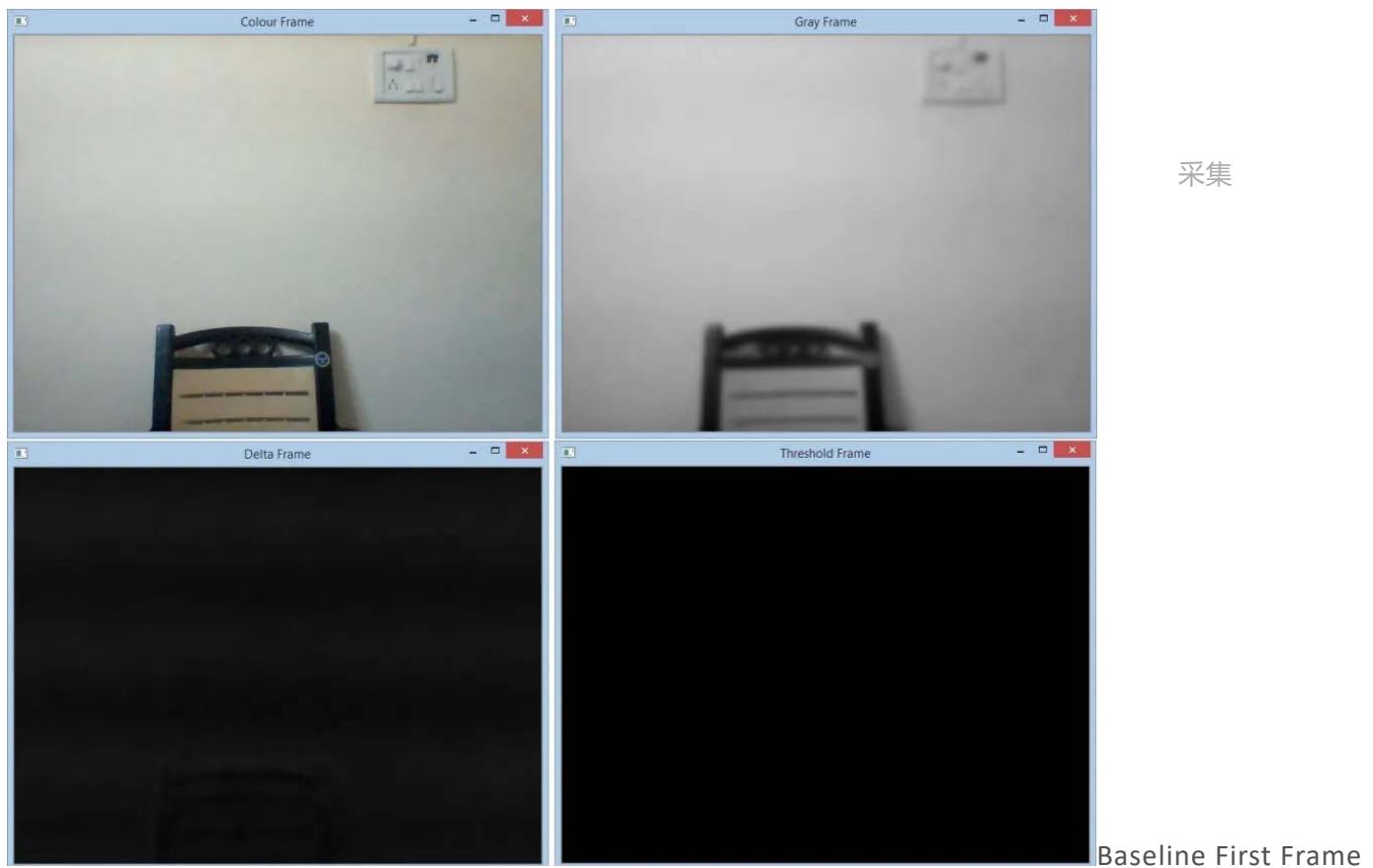
```

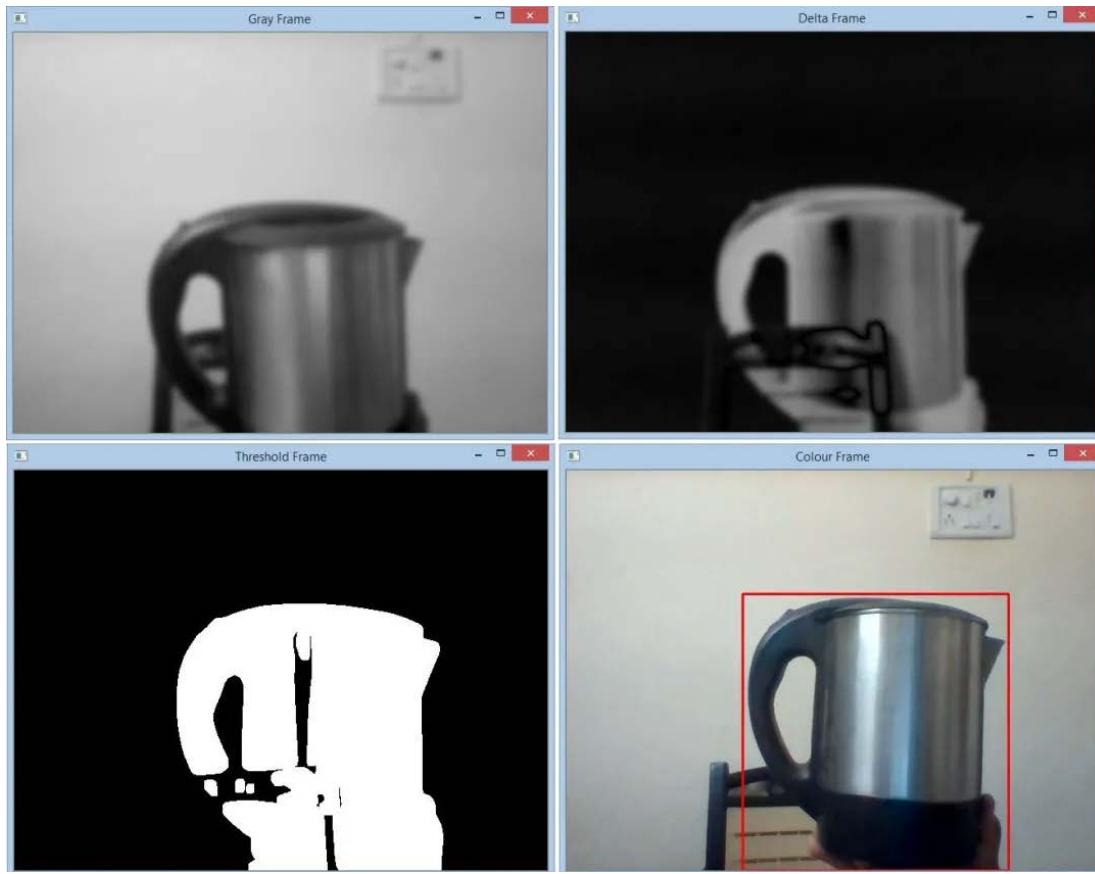
48     key=cv2.waitKey(1)
49     if key==ord('q'):
50         if status==1:
51             time_stamp.append(datetime.now())
52         break
53 #while Loop ends here

```

我们同时需要在按下“Q”的同时捕获最后一个时间戳，因为这将帮助程序结束从摄像机捕获视频的过程，并生成时间数据。

下面是使用该应用程序生成的实际图像输出。第一个图像表示基准帧的4个帧类型，第二个图像表示带有对象的帧的4种类型的帧。你能比较一下区别吗？





Frame with a detected object

第十一步：生成时间数据

```

55 print(status_list)
56
57 for i in range(0,len(time_stamp),2):
58     df=df.append({"Start":time_stamp[i],"End":time_stamp[i+1]},ignore_index=True)
59
60 df.to_csv("All_Time_Stamp.csv")
61 video.release()
62 cv2.destroyAllWindows()

```

到目前为止，所有的时间戳都存储在pandas的data-frame变量中。为了从生成的数据中获得更多信息，我们将把data-frame变量导出到本地磁盘的csv文件中。

All_Time_Stamp.csv - Microsoft Excel						
	A	B	C	D	E	F
1		Start	End			
2	0	31-03-2020 20:08:30	31-03-2020 20:11:25			
3	1	31-03-2020 20:13:19	31-03-2020 20:14:10			
4	2	31-03-2020 20:14:24	31-03-2020 20:14:33			
5						

请不要忘记释放视频变量，因为它在内存中占用了不少空间。同时销毁所有窗口以避免出现不必要的错误

这就是生成的csv的样子。正如我们所看到的那样，在程序结束之前，这个对象已经被检测了3次。您可以查看开始时间和结束时间，并计算对象在摄影机前面的时间。

这个应用程序还不够令人兴奋吗？这个应用程序是不是远离了典型的无聊编程？物联网爱好者甚至可以把这个程序部署到树莓派服务器Raspberry Pi上，并创造奇迹！

如果本文对小伙伴有帮助，希望可以在文末来个“一键三连”。

分享

赞 107

在看 128

交流群

欢迎加入公众号读者群一起和同行交流，目前有**SLAM、三维视觉、传感器、自动驾驶、计算摄影、检测、分割、识别、医学影像、GAN、算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：“昵称+学校/公司+研究方向”，例如：“张三 + 上海交大 + 视觉SLAM”。**请按照格式备注，否则不予通过**。添加成功后会根据研究方向邀请进入相关微信群。**请勿在群内发送广告**，否则会请出群，谢谢理解~





//
//