收藏 | 16個OpenCV函數開始你的計算機視覺之旅

OpenCV與人工智能深度學習 2022-03-04 08:05

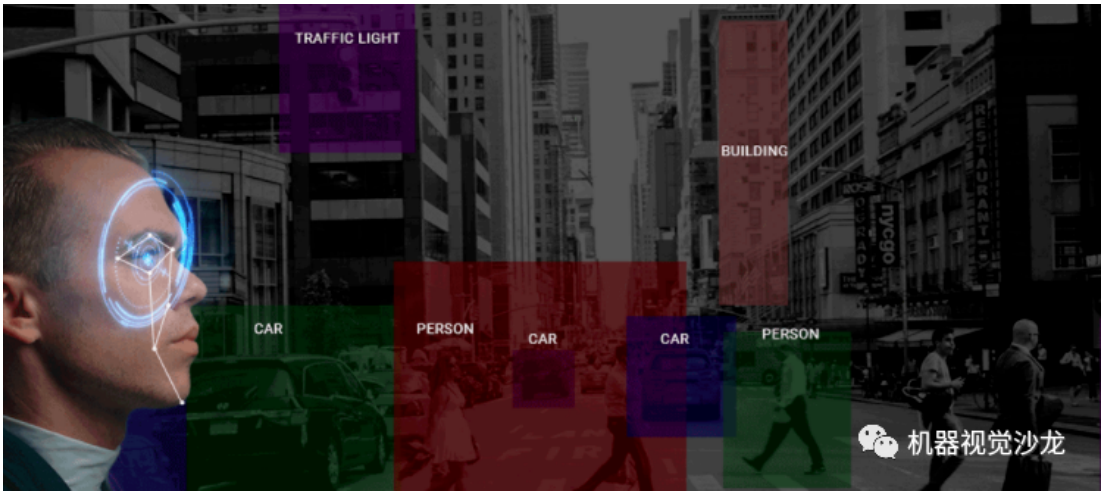
收錄於學科

#計算機視覺 22 #圖像處理 33

轉載自 | AIRX社區

XR開發者或數據科學家在過渡到計算機視覺時面臨著一些常見的挑戰，

1.我們如何收集圖像數據集？圖像有不同的形狀和大小
2.數據獲取中一直存在的問題。在建立計算機模型之前，我們應該收集更多視覺圖像嗎？
3.深入研究對建立計算機視覺模型學習使用學習技術嗎？我們可以不學習嗎？
4.我們可以在自己的機器上建立計算機模型嗎？不是每個人都可以使用GPU和TPU！



目錄

2. 將OpenCV用於計算機視覺任務？
3. 讀取，寫入和顯示圖像
4. 色彩改變空間
5. 調整圖像大小
6. 影像旋轉
7. 圖片翻譯
8. 簡單的影像值
9. 醫療服務價值
10. 圖像分割（分水嶺算法）
11. 按位
12. 標記檢測
13. 圖像過濾
14. 影像影像
15. 尺度不變特徵變換（SIFT）
16. 加速的強大功能（SURF）
17. 特徵匹配
18. 人臉檢測

什麼是CV

在打開簡歷之前，我將快速解釋一下什麼。是我們的學習性。

發現我們看了一條上一條貓的腿圖。一個複雜的步驟，包括特徵提取（特徵檢測、特徵等）、特徵分類等。

可以預計這些是當前行業中最廣泛的領域之一。那麼，未來 2-4 年，你會考慮當你有熱門的職位空缺時，你準備好利用機會了嗎？請花點時間？重命名計算機視覺時，或者？我們每天都在使用，使用手機功能識別應用程序產品，使用手機，點擊自動駕駛汽車等，一些會在。

關於OpenCV

OpenCV庫最初是自己擁有它的一個研究項目。就其功能而言，是目前最大的計算機視覺庫！用於OpenCV包含250個算法的可免費商業實現和學術目標。於澳大利亞語言的接口，包括 Python，Java 和 C++。

OpenCV的第1.0版於2006年發布，發布一個OpenCV社區迅猛發展。

現在，讓我們將注意力轉移到 OpenCV 的背後的上層 OpenCV 的功能！

讀取，寫入和顯示圖像

機器使用數字來查看和處理一切，包括圖像和文本。你如何把轉換成數字？，！

在服務器的圖像中，我展示了一個顏色的圖像值，其中每個人只包含一個位置的黑色強度。

注意，彩色圖像可能有某些可能的值。

讀和寫這個圖像是開放CV的任何計算機視覺項目。使函數非常簡單。

```
#import the libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline
#reading the image
image = cv2.imread('index.png')
image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
#plotting the image
plt.imshow(image)
#saving image
cv2.imwrite('test_write.jpg',image)
```


默认情况下，imread函数以BGR（蓝绿色红色）格式读取图像。我们可以使用imread函数中的额外标志来读取不同格式的图像：

- cv2.IMREAD_COLOR： 加载彩色图像的默认标志
- cv2.IMREAD_GRAYSCALE： 以灰度格式加载图像
- cv2.IMREAD_UNCHANGED： 以给定格式（包括Alpha通道）加载图像。Alpha通道存储透明度信息，Alpha通道的值越高，像素越不透明。

改变色彩空间

颜色空间是一种协议，用于以一种使颜色易于复制的方式表示颜色。我们知道灰度图像有单个像素值，而彩色图像每个像素包含3个值——红、绿、蓝通道的强度。

大多数计算机视觉用例处理RGB格式的图像。然而，像视频压缩和设备独立存储这样的应用程序严重依赖于其他颜色空间，比如颜色-饱和度-值或HSV颜色空间。

正如你所理解的，RGB图像是由不同颜色通道的颜色强度组成的，即强度和颜色信息混合在RGB颜色空间中，而在HSV颜色空间中，颜色和强度信息是相互分离的。这使得HSV颜色空间对光线变化更加健壮。OpenCV默认以BGR格式读取给定的图像。因此，在使用OpenCV读取图像时，需要将图像的颜色空间从BGR更改为RGB。让我们看看怎么做:

```
#import the required libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline
image = cv2.imread('index.jpg')
#converting image to Gray scale
gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
#plotting the grayscale image
plt.imshow(gray_image)
#converting image to HSV format
hsv_image = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
#plotting the HSV image
plt.imshow(hsv_image)
```

调整图像大小

机器学习模型在固定大小的输入下工作。同样的想法也适用于计算机视觉模型。我们用于训练模型的图像必须具有相同的尺寸。

现在，如果我们通过从各种来源抓取图像来创建自己的数据集，这可能会成为问题。这就是调整图像大小的功能。

使用OpenCV可以轻松地放大和缩小图像。当我们需要将图像转换为模型的输入形状时，此操作对于训练深度学习模型很有用。OpenCV支持不同的插值和下采样方法，这些参数可以由以下参数使用：

1.INTER_NEAREST： 最近邻居插值
2.INTER_LINEAR： 双线性插值
3.INTER_AREA： 使用像素面积关系进行重采样
4.INTER_CUBIC： 在4×4像素邻域上的双三次插值
5.INTER_LANCZOS4：在8×8邻域内进行Lanczos插值

OpenCV的调整大小功能默认情况下使用双线性插值。

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#reading the image
```

```
image = cv2.imread('index.jpg')
#converting image to size (100,100,3)
smaller_image = cv2.resize(image,(100,100),inerpolation='linear')
#plot the resized image
plt.imshow(smaller_image)
```

影像旋转

“你需要大量数据来训练深度学习模型”。大多数深度学习算法在很大程度上取决于数据的质量和数量。但是，如果你没有足够大的数据集怎么办？并非所有人都能负担得起手动收集和标记图像的费用。

假设我们正在建立一个图像分类模型来识别图像中存在的动物。因此，下面显示的两个图像都应归类为“狗”：

但是，如果没有对第二幅图像进行训练，该模型可能会发现很难将其归类为狗。那么我们应该怎么做呢?

让我来介绍一下数据扩充技术。该方法允许我们生成更多的样本来训练我们的深度学习模型。数据扩充利用现有的数据样本，通过应用旋转、缩放、平移等图像操作来生成新的数据样本。这使我们的模型对输入的变化具有鲁棒性，并导致更好的泛化。

旋转是最常用和最容易实现的数据扩充技术之一。顾名思义，它包括以任意角度旋转图像，并为其提供与原始图像相同的标签。想想你在手机中旋转图像以达到一定角度的次数——这基本上就是这个功能的作用。

```
#importing the required libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
image = cv2.imread('index.png')
rows,cols = image.shape[:2]
#(col/2,rows/2) is the center of rotation for the image
# M is the cordinates of the center
M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
dst = cv2.warpAffine(image,M,(cols,rows))
plt.imshow(dst)
```

图片翻译

图像平移是一种几何变换，它将图像中每个对象的位置映射到最终输出图像中的新位置。平移操作之后，输入图像中位置（x，y）处的对象将移动到新位置（X，Y）：

X = x + dx
Y = y + dy

在此，dx和dy是沿不同维度的各自平移。图像平移可用于为模型增加平移不变性，因为通过翻译，我们可以更改图像中对象的位置，从而使模型具有更多多样性，从而导致更好的泛化性。

以下面的图片为例。即使图像中没有完整的鞋子，模型也应该能够将其归类为鞋子。

此转换功能通常在图像预处理阶段使用。查看下面的代码，看看它在实际情况下如何工作：

```
#importing the required libraries
```

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
#reading the image
image = cv2.imread('index.png')
#shifting the image 100 pixels in both dimensions
M = np.float32([[1,0,-100],[0,1,-100]])
dst = cv2.warpAffine(image,M,(cols,rows))
plt.imshow(dst)
```


简单图像阈值

阈值化是一种图像分割方法。它将像素值与阈值进行比较，并相应地进行更新。OpenCV支持阈值的多种变化。可以这样定义一个简单的阈值函数：

如果Image（x，y）> threshold，则Image（x，y）= 1
否则，Image（x，y）= 0

阈值只能应用于灰度图像。图像阈值化的一个简单应用就是将图像分为前景和背景。

```
#importing the required libraries
import numpy as np
```

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
#here 0 means that the image is loaded in gray scale format
gray_image = cv2.imread('index.png',0)
ret,thresh_binary = cv2.threshold(gray_image,127,255,cv2.THRESH_BINARY)
ret,thresh_binary_inv = cv2.threshold(gray_image,127,255,cv2.THRESH_BINARY_INV)
ret,thresh_trunc = cv2.threshold(gray_image,127,255,cv2.THRESH_TRUNC)
ret,thresh_tozero = cv2.threshold(gray_image,127,255,cv2.THRESH_TOZERO)
ret,thresh_tozero_inv = cv2.threshold(gray_image,127,255,cv2.THRESH_TOZERO_INV)
#DISPLAYING THE DIFFERENT THRESHOLDING STYLES
names = ['Oiriginal
Image','BINARY','THRESH_BINARY_INV','THRESH_TRUNC','THRESH_TOZERO','THRESH_TOZERO_I
images =
gray_image,thresh_binary,thresh_binary_inv,thresh_trunc,thresh_tozero,thresh_tozero
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(names[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```


自适应阈值

在自适应阈值的情况下，对于图像的不同部分使用不同的阈值。此功能可为光照条件变化的图像提供更好的结果，因此称为“自适应”。Otsu的二值化方法为整个图像找到最佳阈值。它适用于双峰图像（直方图中具有2个峰的图像）。

```
#import the libraries
import numpy as np
```

```
import matplotlib.pyplot as plt
import cv2
%matplotlib inline
#ADAPTIVE THRESHOLDING
gray_image = cv2.imread('index.png',0)
ret,thresh_global = cv2.threshold(gray_image,127,255,cv2.THRESH_BINARY)
#here 11 is the pixel neighbourhood that is used to calculate the threshold value
thresh_mean =
cv2.adaptiveThreshold(gray_image,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,1
thresh_gaussian =
cv2.adaptiveThreshold(gray_image,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINA
names = ['Original Image','Global Thresholding','Adaptive Mean Threshold','Adaptive
Gaussian Thresholding']
images = [gray_image,thresh_global,thresh_mean,thresh_gaussian]
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(names[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```


图像分割（分水岭算法）

图像分割是将图像中的每个像素分类为某个类的任务。例如，将每个像素分类为前景或背景。图像分割对于从图像中提取相关部分非常重要。

分水岭算法是一种经典的图像分割算法。它将图像中的像素值视为地形。为了找到对象边界，它以初始标记作为输入。然后，该算法开始从标记中泛洪盆地，直到标记在对象边界处相遇。

假设我们有多个盆地。现在，如果我们用不同颜色的水填充不同的盆地，那么不同颜色的交点将为我们提供对象边界。

```
#importing required libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
#reading the image
image = cv2.imread('coins.jpg')
#converting image to grayscale format
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
#apply thresholding
ret,thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
#get a kernel
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel,iterations = 2)
#extract the background from image
sure_bg = cv2.dilate(opening,kernel,iterations = 3)
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret,sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_bg)
ret,markers = cv2.connectedComponents(sure_fg)
markers = markers+1
markers[unknown==255] = 0
markers = cv2.watershed(image,markers)
image[markers==-1] = [255,0,0]
```

```
plt.imshow(sure_fg)
```


换位运算

按位运算包括AND，OR，NOT和XOR。你可能在编程课上还记得它们！在计算机视觉中，当我们有一个遮罩图像并且想要将该遮罩应用于另一个图像以提取感兴趣区域时，这些操作非常有用。

```
#import required libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline
#read the image
image = cv2.imread('coins.jpg')
#apply thresholdin
ret,mask = cv2.threshold(sure_fg,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
#apply AND operation on image and mask generated by thrresholding
final = cv2.bitwise_and(image,image,mask = mask)
#plot the result
plt.imshow(final)
```

在上图中，我们可以看到使用分水岭算法计算的输入图像及其分割蒙版。此外，我们应用了按位“与”运算以从图像中删除背景并从图像中提取相关部分。

边缘检测

边缘是图像中图像亮度急剧变化或不连续的点。这种不连续通常对应于：

- 深度不连续
- 表面取向不连续
- 材料特性的变化
- 场景照明的变化

边缘是图像的非常有用的功能，可以用于不同的应用程序，例如图像中的对象分类和定位。甚至深度学习模型也会计算边缘特征，以提取有关图像中存在的对象的信息。

边缘与轮廓不同，因为它们与对象无关，而是表示图像像素值的变化。边缘检测可用于图像分割甚至图像锐化。

```
#import the required libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
#read the image
image = cv2.imread('coins.jpg')
```

```
#calculate the edges using Canny edge algorithm
edges = cv2.Canny(image,100,200)
#plot the edges
plt.imshow(edges)
```


图像过滤

在图像过滤中，使用像素的邻近值更新像素值。但是这些值首先如何更新?

嗯，有多种更新像素值的方法，例如从邻居中选择最大值，使用邻居的平均值等。每种方法都有其自己的用途。例如，将邻域中的像素值平均用于图像模糊。

高斯滤波还用于图像模糊，该模糊基于相邻像素与所考虑像素的距离为它们赋予不同的权重。

对于图像过滤，我们使用内核。内核是具有不同形状的数字矩阵，例如3 x 3、5 x 5等。内核用于计算带有图像一部分的点积。在计算像素的新值时，内核中心与像素重叠。相邻像素值与内核中的相应值相乘。将计算出的值分配给与内核中心一致的像素。

```
#importing the required libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
image = cv2.imread('index.png')
#using the averaging kernel for image smoothening
averaging_kernel = np.ones((3,3),np.float32)/9
filtered_image = cv2.filter2D(image,-1,kernel)
plt.imshow(dst)
#get a one dimensional Gaussian Kernel
gaussian_kernel_x = cv2.getGaussianKernel(5,1)
gaussian_kernel_y = cv2.getGaussianKernel(5,1)
#converting to two dimensional kernel using matrix multiplication
gaussian_kernel = gaussian_kernel_x * gaussian_kernel_y.T
#you can also use cv2.GaussianBLurring(image,(shape of kernel),standard deviation)
instead of cv2.filter2D
filtered_image = cv2.filter2D(image,-1,gaussian_kernel)
plt.imshow()
```

在上面的输出中，右侧的图像显示了在输入图像上应用高斯核的结果。我们可以看到原始图像的边缘被抑制了。具有不同sigma值的高斯核被广泛用于计算我们图像的高斯差。这是特征提取过程中的重要步骤，因为它可以减少图像中存在的噪点。

影像轮廓

轮廓是点或线段的闭合曲线，代表图像中对象的边界。轮廓实质上是图像中对象的形状。
与边缘不同，轮廓不是图像的一部分。相反，它们是与图像中对象形状相对应的点和线段的抽象集合。
我们可以使用轮廓来计算图像中对象的数量，根据对象的形状对对象进行分类，或者从图像中选择特定形状的对象。

```
#importing the required libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
image = cv2.imread('shapes.png')
#converting RGB image to Binary
gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray_image,127,255,0)
#calculate the contours from binary image
im,contours,hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
with_contours = cv2.drawContours(image,contours,-1,(0,255,0),3)
```

```
plt.imshow(with_contours)
```


尺度不变特征变换（SIFT）

关键点是在处理图像时应注意的概念。这些基本上是图像中的关注点。关键点类似于给定图像的特征。

它们是定义图像中有趣内容的位置。关键点很重要，因为无论如何修改图像（旋转，缩小，扩展，变形），我们始终会为图像找到相同的关键点。

尺度不变特征变换（SIFT）是一种非常流行的关键点检测算法。它包括以下步骤：

- 尺度空间极值检测
- 关键点本地化
- 方向分配
- 关键点描述符
- 关键点匹配

从SIFT提取的特征可用于图像拼接，物体检测等应用。以下代码和输出显示了关键点及其使用SIFT计算得出的方向。

```
#import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
%matplotlib inline
#show OpenCV version
print(cv2.__version__)
#read the iamge and convert to grayscale
image = cv2.imread('index.png')
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
#create sift object
sift  = cv2.xfeatures2d.SIFT_create()
#calculate keypoints and their orientation
keypoints,descriptors = sift.detectAndCompute(gray,None)
#plot keypoints on the image
with_keypoints = cv2.drawKeypoints(gray,keypoints)
#plot the image
plt.imshow(with_keypoints)
```


SURF

SURF是SIFT的增强版本。它的工作速度更快，并且对图像转换更健壮。在SIFT中，比例空间使用高斯的拉普拉斯算子近似。什么是高斯的拉普拉斯算子?

拉普拉斯算子是用于计算图像边缘的内核。拉普拉斯核通过近似图像的二阶导数来工作。因此，它对噪声非常敏感。我们通常将高斯核应用于拉普拉斯核之前的图像，因此将其命名为高斯拉普拉斯。

在SURF中，高斯的拉普拉斯算子是使用盒式滤波器（核）来计算的。盒式滤波器的卷积可以针对不同的比例并行进行，这是SURF速度提高的根本原因（与SIFT相比）。

```
#import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
%matplotlib inline
#show OpenCV version
print(cv2.__version__)
#read image and convert to grayscale
image = cv2.imread('index.png')
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
#instantiate surf object
surf  = cv2.xfeatures2d.SURF_create(400)
#calculate keypoints and their orientation
keypoints,descriptors = surf.detectAndCompute(gray,None)
with_keypoints = cv2.drawKeypoints(gray,keypoints)
plt.imshow(with_keypoints)
```


特征匹配

可以匹配使用SIFT或SURF从不同图像中提取的特征，以找到存在于不同图像中的相似对象/图案。OpenCV库支持多种功能匹配算法，例如蛮力匹配，knn特征匹配等。

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
#reading images in grayscale format
image1 = cv2.imread('messi.jpg',0)
image2 = cv2.imread('team.jpg',0)
#finding out the keypoints and their descriptors
keypoints1,descriptors1 = cv2.detectAndCompute(image1,None)
keypoints2,descriptors2 = cv2.detectAndCompute(image2,None)
#matching the descriptors from both the images
bf = cv2.BFMatcher()
```

```
matches = bf.knnMatch(ds1,ds2,k = 2)
#selecting only the good features
good_matches = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])
image3 = cv2.drawMatchesKnn(image1,kp1,image2,kp2,good,flags = 2)
```


在上图中，我们可以看到从原始图像（左侧）提取的关键点与其旋转版本的关键点匹配。这是因为特征是使用SIFT提取的，而SIFT对于此类变换是不变的。

人脸检测

OpenCV支持基于haar级联的对象检测。Haar级联是基于机器学习的分类器，可计算图像中的不同特征（如边缘，线条等）。然后，这些分类器使用多个正样本和负样本进行训练。

OpenCV Github存储库中提供了针对不同对象（如面部，眼睛等）的训练分类器，你还可以针对任何对象训练自己的haar级联。

```
#import required libraries
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#load the classifiers downloaded
face_cascade = cv.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv.CascadeClassifier('haarcascade_eye.xml')
#read the image and convert to grayscale format
img = cv.imread('rotated_face.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
#calculate coordinates
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
for (x,y,w,h) in faces:
    cv.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    #draw bounding boxes around detected features
    for (ex,ey,ew,eh) in eyes:
        cv.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
#plot the image
plt.imshow(img)
#write image
cv2.imwrite('face_detection.jpg',img)
```


參考文章：

https://www.analyticsvidhya.com/2019/03/opencv-functions computer-vision-/本文僅做學術分享博客-python，用，請聯繫刪文。

-結束-

好像有用，麻煩給個贊和在看

收錄於#話題圖像處理 33

下一篇 · （附鏈接）實戰｜如何用YOLOX訓練自己的數據集?

喜歡這個內容的人還喜歡

10年，4600萬台！樹莓派，生日快樂

OpenCV與人工智能深度學習