

1. open CV

1. 使用 Open CV 读取、显示和写入图像

读取、显示和写入图像是图像处理和计算机视觉的基础。即使在裁剪、调整大小、旋转或应用不同的滤镜来处理图像时，也需要首先读取图像。

1. `imread()` 帮助我们读取图像
2. `imshow()` 在窗口中显示图像
3. `imwrite()` 将图像写入文件目录

读取图像

要读取图像，请使用 OpenCV 中的函数。语法如下：`imread()`

```
imread(filename, flags)
```

它需要两个参数：

1. 第一个参数是图像名称，它需要文件的完全限定路径名。
2. 第二个参数是一个可选标志，用于指定应如何表示图像。OpenCV为此标志提供了几个选项，但最常见的选项包括：
 - `cv2.IMREAD_UNCHANGED` 或 -1
 - `cv2.IMREAD_GRAYSCALE` 或 0
 - `cv2.IMREAD_COLOR` 或 1

显示图像

在 OpenCV 中，您可以使用该函数显示图像。语法如下：`imshow()`

```
imshow(window_name, image)
```

此函数还接受两个参数：

1. 第一个参数是将在窗口上显示的窗口名称。
2. 第二个参数是要显示的图像。

要一次显示多个图像，请为要显示的每个图像指定新的窗口名称。

该函数旨在与 `waitKey()` 和 `destroyAllWindows()` / `destroyWindow()` 函数一起使用。

该函数是键盘绑定函数。`waitKey()`

- 它需要一个参数，即将显示窗口的时间（以毫秒为单位）。
- 如果用户在此时间段内按任意键，程序将继续。
- 如果传递 0，程序将无限期地等待击键。
- 还可以将该功能设置为检测特定的击键，例如键盘上的 Q 键或 ESC 键，从而更明确地告诉哪个键应触发哪种行为。

该函数会破坏我们创建的所有窗口。如果需要销毁特定窗口，请提供该确切的窗口名称作为参数。使用还会从系统主内存中清除窗口或图像。下面的代码示例显示了如何使用该函数来显示读入的图像。

代码

```
# import the cv2 library
import cv2

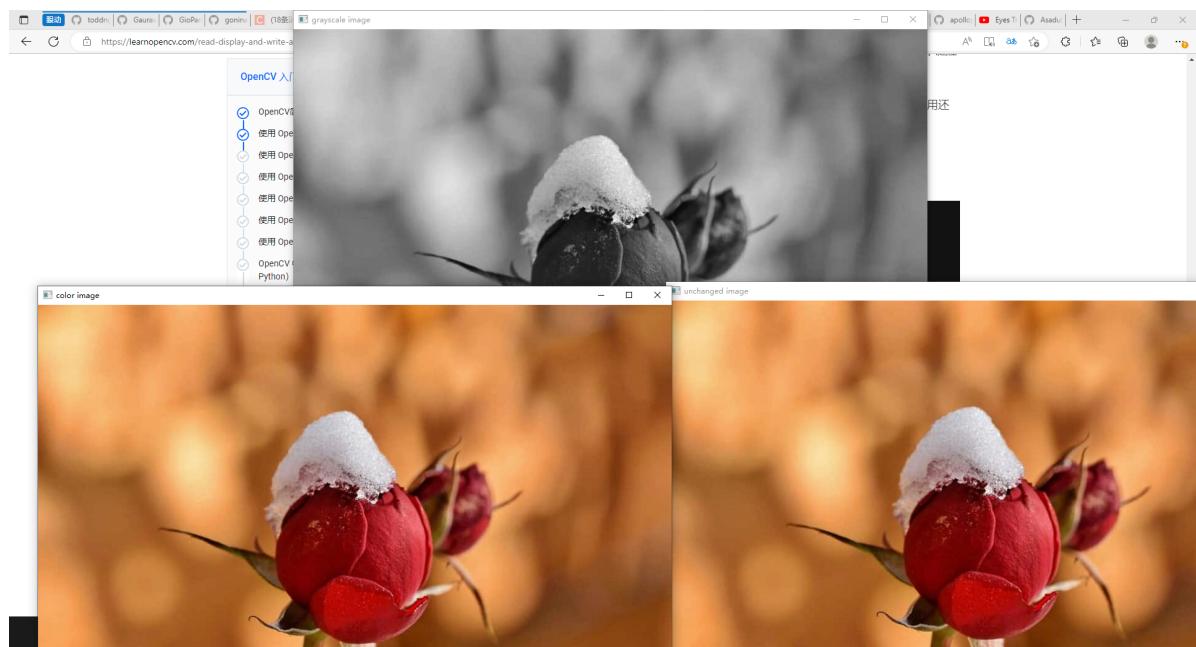
# The function cv2.imread() is used to read an image.
img_color = cv2.imread('openCV\\test.png',cv2.IMREAD_COLOR)
img_grayscale = cv2.imread('openCV\\test.png',cv2.IMREAD_GRAYSCALE)
img_unchanged = cv2.imread('openCV\\test.png',cv2.IMREAD_UNCHANGED)

#Displays image inside a window
cv2.imshow('color image',img_color)
cv2.imshow('grayscale image',img_grayscale)
cv2.imshow('unchanged image',img_unchanged)

# Waits for a keystroke
cv2.waitKey(0)

# Destroys all the windows created
cv2.destroyAllWindows()
```

效果



2. 使用 OpenCV 读取和编写视频

1. `cv2.VideoCapture` - 创建一个视频捕获对象，这将有助于流式传输或显示视频。
2. `cv2.VideoWriter` - 将输出视频保存到目录。
3. 此外，我们还讨论了其他所需的函数，例如 `cv2.imshow()`, `cv2.waitKey()` 以及 `get()` 用于读取视频元数据的方法，例如帧高度，宽度，fps等。

```
VideoCapture(path, apiPreference)
```

#第一个参数是视频文件的文件名/路径。第二个是可选参数，指示首选项。

现在我们有一个视频捕获对象，我们可以使用该 `isOpened()` 方法来确认视频文件已成功打开。该 `isOpened()` 方法返回一个布尔值，指示视频流是否有效。否则，将收到一条错误消息。错误消息可能暗示很多事情。其中之一是整个视频已损坏，或者某些帧已损坏。假设视频文件已成功打开，我们可以使用该 `get()` 方法检索与视频流关联的重要元数据。请注意，此 `get()` 方法不适用于网络摄像头。该方法从[此处](#)记录的选项枚举列表中获取单个参数。在下面的示例中，我们提供了数值 5 和 7，它们对应于帧速率（`CAP_PROP_FPS`）和帧计数（`CAP_PROP_FRAME_COUNT`）。可以提供数值或名称。

```
vid_capture.read()
```

#该方法返回一个元组，其中第一个元素是布尔值，下一个元素是实际的视频帧。当第一个元素为 `True` 时，它表示视频流包含要读取的帧

读取视频

代码

```
import cv2
# 创建一个视频捕获对象，在本例中我们是从文件中读取视频
vid_capture = cv2.VideoCapture('openCV\\test.mp4')
#vid_capture = cv2.VideoCapture('openCV\\test%02d.png')
if (vid_capture.isOpened() == False):
    print("Error opening the video file")
# 读取fps和帧数
else:
    # 获取帧率信息
    # 也可以用CAP_PROP_FPS替换5，它们是枚举
    fps = vid_capture.get(5)
    print('Frames per second : ', fps, 'FPS')

    # 得到帧数
    # 你也可以用CAP_PROP_FRAME_COUNT替换7，它们是枚举
    frame_count = vid_capture.get(7)
    print('Frame count : ', frame_count)

while(vid_capture.isOpened()):
    # vid_capture.read()方法返回一个元组，第一个元素是bool类型
    # 第二个是框架
    ret, frame = vid_capture.read()
    if ret == True:
        cv2.imshow('Frame',frame)
        # 20是以毫秒为单位的，试着增加这个值，比如50，然后观察
        key = cv2.waitKey(20)

        if key == ord('q'):
            break
    else:
        break
# 释放视频捕获对象
vid_capture.release()
cv2.destroyAllWindows()
```

读取图像序列

```
vid_capture = cv2.VideoCapture('openCV\\test%02d.png')#%02d表示两位数序列命名约定（例如test01.png、test02.png、test03.png等）。
```

从网络摄像头读取视频

从网络摄像头读取视频流也与上面讨论的示例非常相似。这一切都归功于 OpenCV 中视频捕获类的灵活性，为了方便起见，它有几个重载函数，可以接受不同的输入参数。无需为视频文件或图像序列指定源位置，只需提供视频捕获设备索引，如下所示。

- 如果您的系统具有内置网络摄像头，则相机的设备索引将为 0
- 如果有多个摄像机连接到系统，则与每个附加摄像机关联的设备索引将递增（例如 1，2 等）。

```
vid_capture = cv2.VideoCapture(0, cv2.CAP_DSHOW)#标志CAP_DSHOW。这是一个可选参数，因此不是必需的。只是另一个视频捕获 API 首选项，它是通过视频输入直接显示的缩写。
```



编写视频

现在让我们来看看如何编写视频。就像视频阅读一样，我们可以编写来自任何来源（视频文件、图像序列或网络摄像头）的视频。要写入视频文件：

- 使用该 `height.get()` 和 `width.get()` 方法来获取视频帧的宽度和高度。
- 初始化视频捕获对象（如前面部分所述），以使用前面所述的任何源将视频流读入内存。
- 创建视频编写器对象。
- 使用视频编写器对象将视频流保存到磁盘。

```
# 使用get()方法获取帧大小信息
frame_width = int(vid_capture.get(3))
frame_height = int(vid_capture.get(4))
frame_size = (frame_width, frame_height)
fps = 20
```

- 在本例中，您将通过指定 `CAP_PROP_FRAME_WIDTH` (3) 和 `CAP_PROP_FRAME_HEIGHT` (4) 来检索帧宽度和高度。在将视频文件写入磁盘时，您将在下面进一步使用这些尺寸。

为了编写视频文件，您需要首先从类创建一个视频编写器对象，如下面的代码所示。`Videowriter()`

```
Videowriter(filename, apiPreference, fourcc, fps, frameSize[, isColor])  
#filename: 输出视频文件的路径名  
#apiPreference: API 后端标识符  
#fourcc: 编解码器的 4 个字符代码，用于压缩帧（fourcc）  
#fps: 创建的视频流的帧速率  
#frame_size: 视频帧的大小  
#isColor: 如果不为零，编码器将期望并编码颜色帧。否则，它将适用于灰度帧（该标志目前仅在Windows 上受支持）。
```

一个特殊的便利函数用于检索四字符编解码器，该编解码器需要作为视频编写器对象的第二个参数。

```
cv2.outputVideowriter()
```

```
Videowriter_fourcc('M', 'J', 'P', 'G') in Python.  
Videowriter::fourcc('M', 'J', 'P', 'G') in C++.
```

视频编解码器指定如何压缩视频流。它将未压缩的视频转换为压缩格式，反之亦然。要创建 AVI 或 MP4 格式，请使用以下 fourcc 规范：

```
.AVI: cv2.Videowriter_fourcc('M', 'J', 'P', 'G')  
.MP4: cv2.Videowriter_fourcc(*'XVID')
```

接下来的两个输入参数指定帧速率（以 FPS 为单位）和帧大小（宽度、高度）。

```
# 初始化视频写入器对象  
output = cv2.Videowriter('Resources/output_video_from_file.avi',  
cv2.Videowriter_fourcc('M', 'J', 'P', 'G'), 20, frame_size)
```

代码

```
import cv2  
# 创建一个视频捕获对象，在本例中我们是从文件中读取视频  
vid_capture = cv2.VideoCapture('openCV\\test.mp4')  
# 使用get()方法获取帧大小信息  
frame_width = int(vid_capture.get(3))  
frame_height = int(vid_capture.get(4))  
frame_size = (frame_width, frame_height)  
fps = 20  
# 初始化视频写入器对象  
output = cv2.Videowriter('output_video_from_file.avi',  
cv2.Videowriter_fourcc('M', 'J', 'P', 'G'), 20, frame_size)  
while(vid_capture.isOpened()):  
    # vid_capture.read()方法返回一个元组，第一个元素是bool类型 第二个是框架  
    ret, frame = vid_capture.read()  
    if ret == True:  
        # 将框架写入输出文件  
        output.write(frame)  
    else:  
        print('Stream disconnected')  
        break  
# 释放对象
```

```
vid_capture.release()  
output.release()
```

结果：

The screenshot shows a VS Code interface with a dark theme. On the left is a tree view of a Python project named 'pythonProject'. Inside the 'PYTHONPROJECT' folder, there are several files: 'Demo01.py', 'Demo02_1.py' (which is the active file), 'Demo02_1.ipynb', 'Eyes-Position-Estimator-Mediapipe', 'openCV', 'tempCodeRunnerFile.py', and some image files ('test.mp4', 'test01.png', 'test02.png', 'test03.png', 'output_video_from_file.avi', 'output21.mp4'). The code in 'Demo02_1.py' is as follows:

```
import cv2  
# 创建一个视频捕获对象，在本例中我们是从文件中读取视频  
vid_capture = cv2.VideoCapture('openCV\\test.mp4')  
frame_width = int(vid_capture.get(3))  
frame_height = int(vid_capture.get(4))  
frame_size = (frame_width, frame_height)  
fps = 20  
  
# 初始化视频写入器对象  
output = cv2.VideoWriter('output_video_from_file.avi', cv2.VideoWriter_fourcc('M','J','P','G'), 20, frame_size)  
while(vid_capture.isOpened()):  
    # Vid_capture.read()方法返回一个元组，第一个元素是bool类型 第二个是框架  
  
    ret, frame = vid_capture.read()  
    if ret == True:  
        # 将框架写入输出文件  
        output.write(frame)  
    else:  
        print('Stream disconnected')  
        break  
  
# 释放对象  
vid_capture.release()
```

Below the code editor, the terminal window shows the command run: `python -u "d:\VScodeProject\pythonProject\openCV\Demo02_1.py"`. The output indicates the stream is disconnected.

3. 使用 OpenCV 调整图像大小

调整图像大小时：

- 如果您也想在调整大小的图像中保持相同的宽高比，请务必记住图像的原始纵横比（即宽度与高度）。
- 减小图像的大小将需要对像素进行重新采样。
- 增加图像的大小需要重建图像。这意味着您需要插值新像素

```
# 读取图片  
image = cv2.imread('image.jpg')
```

```
# 获取原始的高度和宽度、通道数  
h,w,c = image.shape  
print("Original Height and Width:", h,"x", w)
```

这里需要注意的重要一点是，OpenCV以格式输出 $height * width * channels$ 图像的形状，而其他一些图像处理库则以宽度,高度的形式输出。

调整函数语法大小

```
resize(src, dsize[src,dst[dsize fx, fy, interpolation])
//源图像。
//调整大小的图像的所需大小 dsize
/*src: 它是必需的输入图像，它可以是一个带有输入图像路径的字符串（例如：“test_image.png”）。
dsize: 它是输出图像的所需大小，它可以是新的高度和宽度。
fx: 沿水平轴的比例因子。
fy: 沿垂直轴的比例因子。
interpolation: 它为我们提供了调整图像大小的不同方法的选项。*/

```

通过指定宽度和高度调整大小

```
# 设置行和列
# 使用新的宽度和高度缩小图像
down_width = 300
down_height = 200
down_points = (down_width, down_height)
resize_down = cv2.resize(image, down_points, interpolation= cv2.INTER_LINEAR)
```

使用比例因子调整大小

```
# 通过指定两个缩放因子，将图像放大1.2倍
scale_up_x = 1.2
scale_up_y = 1.2
# 将图像按比例缩小0.6倍，指定一个比例因子。
scale_down = 0.6

scaled_f_down = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_LINEAR)
scaled_f_up = cv2.resize(image, None, fx= scale_up_x, fy= scale_up_y,
interpolation= cv2.INTER_LINEAR)
```

使用不同的插值方法调整大小

不同的插值方法用于不同的调整大小目的。

`INTER_AREA`:使用像素面积关系进行重采样。这最适合于减小图像的大小(缩小)。当用于放大图像时，它使用该方法:`INTER_NEAREST`

`INTER_CUBIC`:它使用双立方插值来调整图像的大小。在调整大小和插值新像素时，该方法作用于图像的 4×4 相邻像素。然后取16个像素的加权平均值来创建新的插值像素。

`INTER_LINEAR`:这个方法有点类似于插值。但与之不同的是，它使用 2×2 相邻像素来获得插值像素的加权平均值。

`INTER_NEAREST`:该方法使用最近邻概念进行插值。这是最简单的方法之一，只使用图像中的一个相邻像素进行插值。

```
# 用不同的插值方法将图像缩小0.6倍
res_inter_nearest = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_NEAREST)
res_inter_linear = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_LINEAR)
res_inter_area = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_AREA)
```

```
# 将图像在横轴上连接以作比较
vertical=np.concatenate((res_inter_nearest, res_inter_linear, res_inter_area),
axis =0)
# 显示图像按任意键继续
cv2.imshow('Inter Nearest :: Inter Linear :: Inter Area', vertical)
```

代码

```
import cv2
import numpy as np

# 使用imread函数读取图像
image = cv2.imread('opencv\car.jpg')
cv2.imshow('Original Image', image)

# 使用新的宽度和高度缩小图像
down_width = 300
down_height = 200
down_points = (down_width, down_height)
resized_down = cv2.resize(image, down_points, interpolation= cv2.INTER_LINEAR)

# 使用新的宽度和高度来放大图像
up_width = 600
up_height = 400
up_points = (up_width, up_height)
resized_up = cv2.resize(image, up_points, interpolation= cv2.INTER_LINEAR)
# 显示图片
cv2.imshow('Resized Down by defining height and width', resized_down)
cv2.imshow('Resized Up image by defining height and width', resized_up)

# 用不同的插值方法将图像缩小0.6倍
scale_up_x = 1.2
scale_up_y = 1.2
scale_down = 0.6
scaled_f_down = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_LINEAR)
scaled_f_up = cv2.resize(image, None, fx= scale_up_x, fy= scale_up_y,
interpolation= cv2.INTER_LINEAR)
res_inter_nearest = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_NEAREST)
res_inter_linear = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_LINEAR)
res_inter_area = cv2.resize(image, None, fx= scale_down, fy= scale_down,
interpolation= cv2.INTER_AREA)
# 将图像在横轴上连接以作比较
```

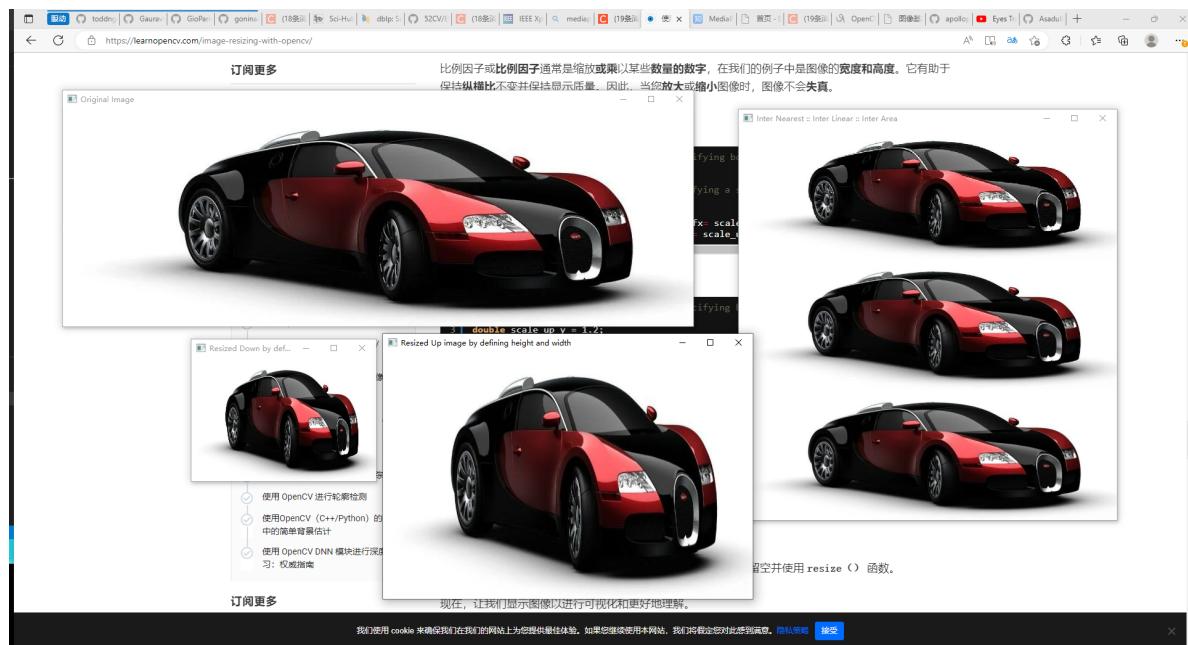
```

vertical= np.concatenate((res_inter_nearest, res_inter_linear, res_inter_area),
axis = 0)
cv2.imshow('Inter Nearest :: Inter Linear :: Inter Area', vertical)
cv2.waitKey()

#按任意键关闭窗口
cv2.destroyAllWindows()

```

结果图



4. 使用 OpenCV 裁剪图像

在 Python 中，您可以使用与 NumPy 数组切片相同的方法来裁剪图像。要对数组进行切片，您需要指定第一个维度和第二个维度的开始和结束索引。

- 第一个维度始终是图像的行数或高度。
- 第二个维度是列数或图像的宽度。

```
cropped = img[start_row:end_row, start_col:end_col]
```

加载高度和宽度以指定需要裁剪较小修补程序的范围。为此，请使用 Python 中的函数。现在，使用两个循环进行裁剪：

1. 一个用于宽度范围
2. 另一个用于高度范围

```

import cv2
import numpy as np
img = cv2.imread('openCV\\test.png')
image_copy = img.copy()
imgheight=img.shape[0]
imgwidth=img.shape[1]
#M、N计算为imgheight、imgwidth 1/3 应该切成9张
M = 227

```

```

N = 341
x1 = 0
y1 = 0
for y in range(0, imgheight, M):#M 为步长
    for x in range(0, imgwidth, N):#N 为步长
        if (imgheight - y) < M or (imgwidth - x) < N:
            break
        y1 = y + M
        x1 = x + N
        # 检查补丁的宽度或高度是否超过图像的宽度或高度
        if x1 >= imgwidth and y1 >= imgheight:
            x1 = imgwidth - 1
            y1 = imgheight - 1
            #剪成MXN大小的小块
            tiles = image_copy[y:y+M, x:x+N]
            #将每个补丁保存到文件目录中

cv2.imwrite('openCV\saved_patches\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)
elif y1 >= imgheight: # 当补丁高度超过图像高度时
    y1 = imgheight - 1
    #剪成MXN大小的小块
    tiles = image_copy[y:y+M, x:x+N]
    #将每个补丁保存到文件目录中

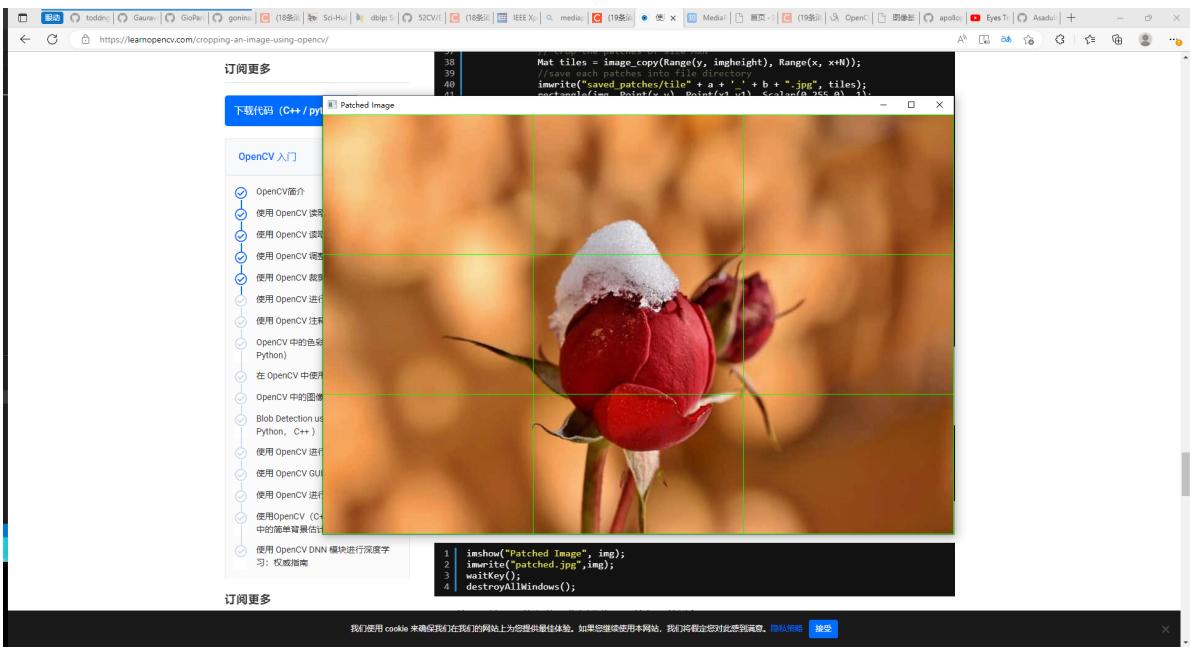
cv2.imwrite('openCV\saved_patches\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)
elif x1 >= imgwidth: # 当补丁宽度超过图像宽度时
    x1 = imgwidth - 1
    #剪成MXN大小的小块
    tiles = image_copy[y:y+M, x:x+N]
    #将每个补丁保存到文件目录中

cv2.imwrite('openCV\saved_patches\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)
else:
    #剪成MXN大小的小块
    tiles = image_copy[y:y+M, x:x+N]
    #将每个补丁保存到文件目录中

cv2.imwrite('openCV\saved_patches\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)
#保存完整的图像到文件目录
cv2.imshow("Patched Image",img)
cv2.imwrite("patched.jpg",img)
cv2.waitKey()
cv2.destroyAllWindows()

```

效果图



使用裁剪的一些有趣的应用程序

1. 您可以使用裁剪从图像中提取感兴趣区域，并丢弃不需要使用的其他部分。
2. 您可以从图像中提取补丁以训练基于补丁的神经网络。

5. 使用 OpenCV 进行图像旋转和转换

创建 2D 旋转矩阵的语法

```
getRotationMatrix2D(center, angle, scale)
#center: 输入图像的旋转中心
#angle: 以度为单位的旋转角度
#scale: 根据提供的值向上或向下缩放图像的各向同性比例因子
```

旋转三步操作：

1. 首先，需要获得旋转中心。这通常是您尝试旋转的图像的中心。
2. 接下来，创建 2D 旋转矩阵。OpenCV 提供了我们上面讨论的功能。
3. 最后，使用在上一步中创建的旋转矩阵将仿射变换应用于图像。OpenCV 中的 warpAffine() 函数可以完成这项工作。

```
warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])
```

#src: 图片源
#M: 转换矩阵
#dsize: 输出图像的大小
#dst: 输出图像
#flags: 插值方法的组合，如INTER_LINEAR或INTER_NEAREST
#borderMode: 像素外推法
#borderValue: 在常量边框的情况下使用的值，默认值为 0

代码

```
import cv2

# 读取图像
image = cv2.imread('openCV\\car.jpg')
```

```

# 把高度和宽度除以2得到图像的中心
height, width = image.shape[:2]
# 获取图像的中心坐标，创建二维旋转矩阵
center = (width/2, height/2)

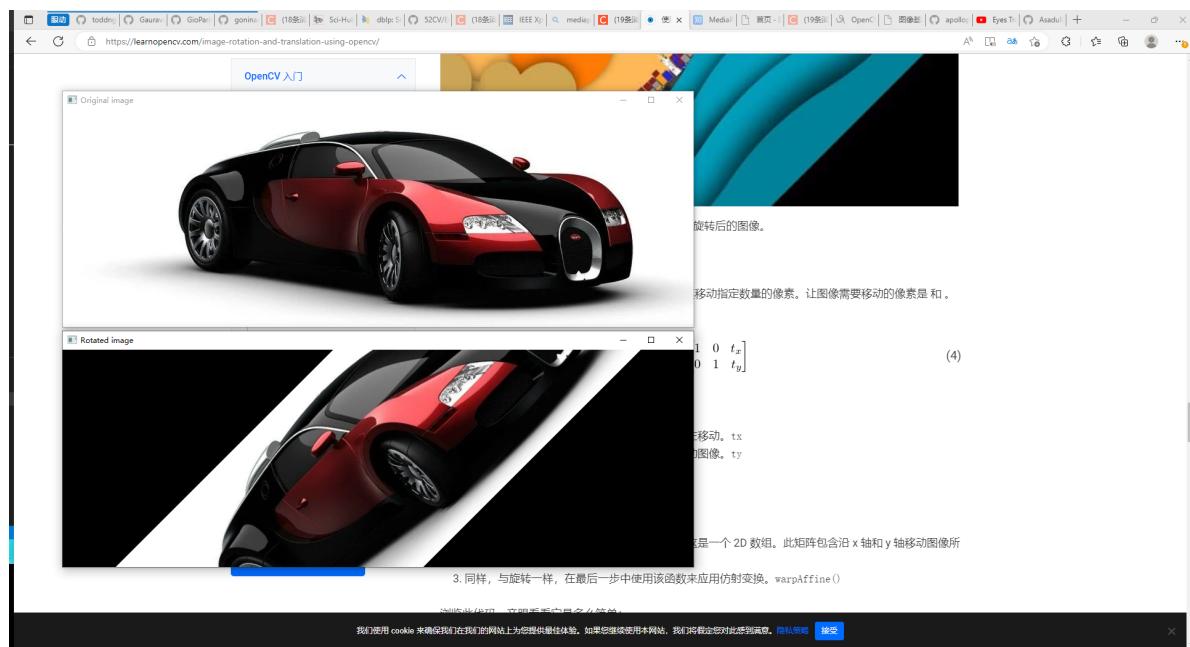
# 使用cv2.getRotationMatrix2D()来获得旋转矩阵
rotate_matrix = cv2.getRotationMatrix2D(center=center, angle=45, scale=1)

# 使用cv2.warpAffine旋转图像
rotated_image = cv2.warpAffine(src=image, M=rotate_matrix, dsize=(width,
height))

cv2.imshow('Original image', image)
cv2.imshow('Rotated image', rotated_image)
# 无限等待，按键盘上的任意键退出
cv2.waitKey(0)
# 将旋转后的图像保存
cv2.imwrite('rotated_image.jpg', rotated_image)

```

效果



使用OpenCV平移图像

在计算机视觉中，图像的平移意味着沿x轴和y轴将其移动指定数量的像素。让图像需要移动的像素是 `tx` 和 `ty`。然后可以定义一个平移矩阵。

`M:txty`

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

现在，在移动图像和值时，您应该记住几点。

- `tx` 提供正值会将图像向右移动，负值会将图像向左移动。
- 同样，`ty` 正值将向下移动图像，而负值将向上移动图像。

按照以下步骤使用 OpenCV 翻译图像：

- 首先，读取图像并获得其宽度和高度。
- 接下来，就像旋转一样，创建一个转换矩阵，这是一个 2D 数组。此矩阵包含沿 x 轴和 y 轴移动图像所需的信息。
- 同样，与旋转一样，在最后一步中使用 `warpAffine()` 函数来应用仿射变换。

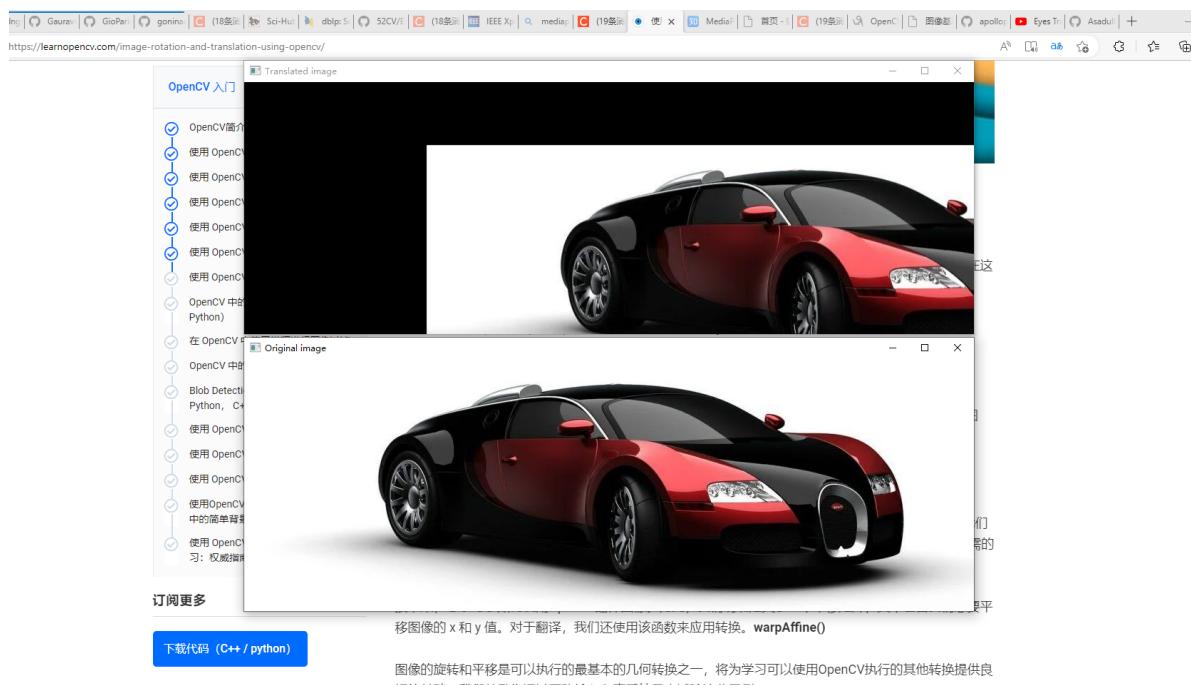
代码

```

import cv2
import numpy as np
# 读取图像
image = cv2.imread('openCV\\car.jpg')
# 把高度和宽度除以2得到图像的中心
height, width = image.shape[:2]
# 获取用于转换的tx和ty值
# 指定所选的任何值
tx, ty = width / 4, height / 4
# 使用tx和ty创建转换矩阵，它是一个NumPy数组
translation_matrix = np.array([
    [1, 0, tx],
    [0, 1, ty]
], dtype=np.float32)
# 平移
translated_image = cv2.warpAffine(src=image, M=translation_matrix, dsize=(width,
height))
# 展示原图和翻平移图
cv2.imshow('Translated image', translated_image)
cv2.imshow('Original image', image)
cv2.waitKey(0)
# 保存
cv2.imwrite('translated_image.jpg', translated_image)

```

效果



6. 使用 OpenCV 注释图像

- 向演示中添加信息
- 在检测到对象时在对象周围绘制边界框
- 突出显示具有不同颜色的像素以进行图像分割

一旦你学会了注释图像，注释视频帧似乎也一样容易。这是因为视频中的每一帧都表示为一个图像。

6.1画一条线

```
line(image, start_point, end_point, color, thickness)
#image: 图像源。
#start_point: 起点坐标
#end_point: 终点坐标
#color: 线段颜色
#thickness: 线段的粗细
```

6.2画一个圆

```
circle(image, center_coordinates, radius, color, thickness)
#image: 图像源。
#center_coordinates: 圆心坐标
#radius: 半径
#color: 线段颜色
#thickness: 线段的粗细
```

6.3绘制实心圆

当 `circle` 函数属性: `thickness` 置为 `-1` 时即可填充此圆圈。

6.4绘制矩形

```
rectangle(image, start_point, end_point, color, thickness)
#image: 图像源。
#start_point: 起点（左上角）坐标
#end_point: 终点（右下角）坐标
#color: 线段颜色
#thickness: 线段的粗细
```

6.5绘制椭圆

```
ellipse(image, centerCoordinates, axesLength, angle, startAngle, endAngle,
color, thickness)
#image: 图像源。
#centerCoordinates: 中心坐标
#axesLength: 长、短轴长度
#angle: 角度
#startAngle: 开始角度
#endAngle: 结束角度
#color: 线段颜色
#thickness: 线段的粗细
```

6.6绘制半椭圆

1. 只绘制蓝色椭圆的一半
2. 将垂直红色椭圆更改为水平红色椭圆，该椭圆为半填充

为此，请进行以下更改：

- 将蓝色椭圆 `endAngle` 设置为 180 度
- 将红色椭圆的 `angle` 从 90 更改为 0
- 将红色椭圆的 `startAngle` 和 `endAngle` 分别指定为 0 和 180
- 将红色椭圆的 `thickness` 指定为负数

6.7添加文本

```
putText(image, text, org, font, fontScale, color)
#image: 图像源
#text: 注释图像的实际文本字符串
#org: 文本字符串左上角的起始位置
#font: 字体样式
    #FONT_HERSHEY_SIMPLEX      = 0,
    #FONT_HERSHEY_PLAIN        = 1,
    #FONT_HERSHEY_DUPLEX       = 2,
    #FONT_HERSHEY_COMPLEX      = 3,
    #FONT_HERSHEY_TRIPLEX     = 4,
    #FONT_HERSHEY_COMPLEX_SMALL = 5,
    #FONT_HERSHEY_SCRIPT_SIMPLEX = 6,
    #FONT_HERSHEY_SCRIPT_COMPLEX = 7,
    #FONT_ITALIC                = 16
#fontScale: 字体比例
#color: 字体颜色
```

代码

```
import cv2
img = cv2.imread('openCV\sample.jpg')
# 显示图象
cv2.imshow('Original Image', img)
cv2.waitKey(0)
# 如果图像为空，打印错误消息
if img is None:
    print('Could not read image')

# 画直线
imageLine = img.copy()
# 从A点画到B点
pointA = (200, 80)
pointB = (450, 80)
cv2.line(imageLine, pointA, pointB, (255, 255, 0), thickness=3,
lineType=cv2.LINE_AA)
cv2.imshow('Image Line', imageLine)
cv2.waitKey(0)

# 画圆
# 复制图像
```

```
imageCircle = img.copy()
# 确定圆心
circle_center = (415,190)
# 定义圆的半径
radius =100
# 使用circle()函数绘制一个圆
cv2.circle(imageCircle, circle_center, radius, (0, 0, 255), thickness=3,
lineType=cv2.LINE_AA)
# 显示结果
cv2.imshow("Image Circle",imageCircle)
cv2.waitKey(0)

#画实心圆
# 复制图像
imageFilledCircle = img.copy()
# 定义圆心
circle_center = (415,190)
# 定义圆的半径
radius =100
# 在输入图像上绘制填充圆
cv2.circle(imageFilledCircle, circle_center, radius, (255, 0, 0), thickness=-1,
lineType=cv2.LINE_AA)
# 显示输出图像
cv2.imshow('Image with Filled Circle',imageFilledCircle)
cv2.waitKey(0)

#画矩形
# 复制图像
imageRectangle = img.copy()
# 定义矩形的起点和终点
start_point =(300,115)
end_point =(475,225)
# 画矩形
cv2.rectangle(imageRectangle, start_point, end_point, (0, 0, 255), thickness= 3,
lineType=cv2.LINE_8)
#显示输出
cv2.imshow('imageRectangle', imageRectangle)
cv2.waitKey(0)

#画椭圆
# 复制图像
imageEllipse = img.copy()
#定义椭圆中心点
ellipse_center = (415,190)
#定义椭圆的长轴和短轴
axis1 = (100,50)
axis2 = (125,50)
#水平
cv2.ellipse(imageEllipse, ellipse_center, axis1, 0, 0, 360, (255, 0, 0),
thickness=3)
#垂直
cv2.ellipse(imageEllipse, ellipse_center, axis2, 90, 0, 360, (0, 0, 255),
thickness=3)
#显示输出
cv2.imshow('ellipse Image',imageEllipse)
cv2.waitKey(0)

#画半椭圆
```

```

#复制图像
halfEllipse = img.copy()
#定义半椭圆的中心
ellipse_center = (415,190)
#定义轴点
axis1 = (100,50)
#绘制不完全椭圆
cv2.ellipse(halfEllipse, ellipse_center, axis1, 0, 180, 360, (255, 0, 0),
thickness=3)
# 填充椭圆
cv2.ellipse(halfEllipse, ellipse_center, axis1, 0, 0, 180, (0, 0, 255),
thickness=-2)
#显示输出
cv2.imshow('halfEllipse',halfEllipse)
cv2.waitKey(0)

#复制图像
imageText = img.copy()
#图像上的文本
text = 'I am a Happy dog!'
#org:文本的位置
org = (50,350)
#在输入图像上写入文本
cv2.putText(imageText, text, org, fontFace = cv2.FONT_HERSHEY_COMPLEX, fontScale
= 1.5, color = (250,225,100))
#显示带有文本的输出图像
cv2.imshow("Image Text",imageText)
cv2.waitKey(0)

cv2.destroyAllWindows()

```

总结

用几何形状和文本注释图像是一种强大的交流方式。它有助于放大图像上的信息。在图像由各种计算机视觉算法处理后，图像几乎总是被注释为叠加结果（例如，在对象检测模型检测到的对象周围绘制边界框）。

7. OpenCV 中的色彩空间

```

cv2.cvtColor(input_image, flag)
#input_image:需要转换的图片
#flag: 转换的类型
#return :颜色空间转换后的图片矩阵

```

转化类型

转换类型	Opencv2.x	Opencv3.x
RGB<-->BGR	CV_BGR2BGRA、CV_RGB2BGRA、CV_BGRA2RGBA、CV_BGR2BGR、CV_BGRA2BGR	COLOR_BGR2BGRA,COLOR_RGB2BGRA、COLOR_BGRA2RGB,COLOR_BGR2BGR、COLOR_BGRA2BGR
RGB<-->GRAY	CV_RGB2GRAY、CV_GRAY2RGB、CV_RGBA2GRAY、CV_GRAY2GRBA	COLOR_RGB2GRAY,COLOR_GRAY2RGB、COLOR_RGBA2GRAY,COLOR_GRAY2GRBA
RGB<-->HSV	CV_BGR2HSV、CV_RGB2HSV、CV_HSV2BGR、CV_HSV2RGB	COLOR_BGR2HSV,COLOR_RGB2HSV、COLOR_HSV2BGR、COLOR_HSV2RGB
RGB<-->YCrCb JPEG(或 YCC)	CV_RGB2YCrCb、CV_YCrCb2BGR、CV_YCrCb2RGB (可以用 YUV 代替 YCrCb)	COLOR_RGB2YCrCb,COLOR_RGB2YCrCb、COLOR_YCrCb2BGR、COLOR_YCrCb2RGB (可以用 YUV 代替 YCrCb)
RGB <-->CIE XYZ	CV_BGR2XYZ,CV_RGB2XYZ, CV_XYZ2BGR, CV_XYZ2RGB	COLOR_BGR2XYZ,COLOR_RGB2XYZ、 COLOR_XYZ2BGR, COLOR_XYZ2RGB
RGB<-->HLS	CV_BGR2HLS,CV_RGB2HLS, CV_HLS2BGR, CV_HLS2RGB	COLOR_BGR2HLS,COLOR_RGB2HLS、 COLOR_HLS2BGR, COLOR_HLS2RGB
RGB<-->CIE L*a*b	CV_BGR2Lab,CV_RGB2Lab, CV_Lab2BGR, CV_Lab2RGB	COLOR_BGR2Lab,COLOR_RGB2Lab、 COLOR_Lab2BGR, COLOR_Lab2RGB
RGB<-->CIE L*a*b	CV_BGR2Luv,CV_RGB2Luv, CV_Luv2BGR, CV_Luv2RGB	COLOR_BGR2Luv,COLOR_RGB2Luv、 COLOR_Luv2BGR, COLOR_Luv2RGB
Bay-->RGB	CV_BayerBG2BGR,CV_BayerGB2BGR, CV_BayerRG2BGR,CV_BayerGR2BGR, CV_BayerBG2RGB,CV_BayerGB2RGB, CV_BayerRG2RGB,CV_BayerGR2RGB	COLOR_BayerBG2BGR,COLOR_BayerGB2BGR、 COLOR_BayerRG2BGR,COLOR_BayerGR2BGR、 COLOR_BayerBG2RGB,COLOR_BayerGB2RGB、 COLOR_BayerRG2RGB,COLOR_BayerGR2RGB

8.在 OpenCV 中使用卷积进行图像过滤

8.1图像处理中的卷积核简介

在图像处理中，卷积核是用于过滤图像的 2D 矩阵。卷积核也称为卷积矩阵，通常是一个正方形的 $M \times N$ 矩阵，其中 M 和 N 都是奇整数（例如 3×3 、 5×5 、 7×7 等）。

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (1)$$

3×3 二维卷积核

这种内核可用于对图像的每个像素执行数学运算，以实现所需的效果（如模糊或锐化图像）。但是为什么要模糊图像呢？这里有两个重要原因：

- 因为它减少了图像中某些类型的噪点。因此，模糊通常称为平滑。
- 要去除分散注意力的背景，可以有意模糊图像的某些部分，就像在移动设备摄像头上的“人像”模式下所做的那样。

作为计算机视觉中的基本处理技术，使用内核过滤图像还有更多应用。

8.2如何使用卷积核锐化或模糊图像

源图像的滤波是通过将核与图像进行卷积来实现的。简单地说，图像与核的卷积表示的是核与图像中相应元素之间的一种简单的数学运算。

- 假设内核的中心位于图像中的特定像素 (p) 上。
- 然后将内核中每个元素的值（在本例中为 1）与源图像中相应的像素元素（即其像素强度）相乘。
- 现在，将这些乘法的结果相加并计算平均值。
- 最后，将像素 (p) 的值替换为您刚刚计算的平均值。

使用上述 3×3 卷积核对源图像中的每个像素执行此操作后，生成的过滤图像将显示为模糊。这是因为使用此卷积核的卷积操作具有平均效果，这往往会平滑或模糊图像。您很快就会亲眼看到内核中各个元素的值如何决定过滤的性质。例如，通过更改卷积核元素的值，还可以实现锐化效果。这个概念很简单，但功能非常强大，因此被用于许多图像处理管道。

8.3 将恒等卷积核应用于 OpenCV 中的图像

在我们描述如何实现模糊和锐化卷积核之前，我们先了解一下恒等核。恒等卷积核是一个方阵，其中中间元素为 1，所有其他元素为零，如下所示。

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (2)$$

单位矩阵的特别之处在于，将其与任何其他矩阵相乘将返回原始矩阵。现在演示如何将这个恒等卷积核与 OpenCV 过滤函数一起使用。在第一个示例中，我们将使用上面的恒等核来证明过滤操作使原始图像保持不变。

- 读取测试图像
- 使用 3×3 NumPy 数组定义标识卷积核
- 使用 OpenCV 中的 `filter2D()` 函数执行线性滤波操作
- 使用 `imshow()` 显示原始图像和过滤图像
- 使用 `imwrite()` 将过滤后的图像保存到磁盘

```
filter2D(src, ddepth, kernel)
#src:源图像
#ddepth:结果图像的深度。值为-1 表示最终图像也将具有与源图像相同的深度
#kernel: 卷积核
```

8.4 使用自定义 2D 卷积核模糊图像

接下来，将演示如何模糊图像。将定义一个自定义卷积核，并使用 OpenCV 中的 `filter2D()` 函数对源图像应用过滤操作。

首先定义一个 5×5 卷积核，仅由卷积核组成。我们还将卷积核除以 25。因为在使用 2D 卷积矩阵对图像应用任何卷积之前，需要确保所有值都已归一化。这是通过将卷积核的每个元素除以卷积核中的元素数（在本例中为 25）来完成的。这可确保所有值保持在 [0, 1] 范围内。

现在使用 `filter2D()` 函数来过滤图像。`filter2D()` 可用于卷积图像，具有任何用户定义的卷积核。

```
"""
应用模糊卷积核
"""

kernel2 = np.ones((5, 5), np.float32) / 25
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel2)

cv2.imshow('original', image)
cv2.imshow('Kernel Blur', img)

cv2.waitKey()
cv2.imwrite('blur_kernel.jpg', img)
cv2.destroyAllWindows()
```

8.5 使用 OpenCV 的内置函数模糊图像

还可以使用 `opencv` 的内置 `blur()` 函数对图像进行模糊处理。本质上是一个方便的功能，使用它来模糊图像，你不需要专门定义内核。只需使用 `ksize` 输入参数指定内核大小，如下面的代码所示。然后，模糊函数将在内部创建一个 5×5 模糊内核，并将其应用于源图像。

```
"""
使用 blur() 函数应用模糊
"""

img_blur = cv2.blur(src=image, ksize=(5,5)) # 使用模糊函数模糊一个图像，其中ksize是内核大小

cv2.imshow('original', image)
cv2.imshow('blurred', img_blur)

cv2.waitKey()
cv2.imwrite('blur.jpg', img_blur)
cv2.destroyAllWindows()
```

8.6 在 OpenCV 中将高斯模糊应用于图像

该技术使用高斯滤波器，该滤波器执行加权平均值，而不是第一个示例中描述的均匀平均值。在这种情况下，高斯模糊根据像素值与内核中心的距离对像素值进行加权。离中心较远的像素对加权平均值的影响较小。以下代码使用 `opencv` 中的 `GaussianBlur()` 函数卷积图像。

```
GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])
#src: 源图像
#ksize: 高斯核的大小
#sigmaX 和 sigmaY, 它们都设置为 0。这些是 X（水平）和 Y（垂直）方向上的高斯核标准差。sigmaY 的默认设置为零。如果只是将 sigmaX 设置为零，则根据内核大小（分别为宽度和高度）计算标准偏差。还可以将每个参数的大小显式设置为大于零的正值
```

8.7 在 OpenCV 中对图像应用中值模糊

使用OpenCV中的`medianBlur()` 函数应用中位数模糊。在中值模糊中，源图像中的每个像素都替换为内核区域中图像像素的中值。

```
medianBlur(src, ksize)
#src: 源图像。
#ksize: 内核大小，它必须是奇数正整数
```

8.8. 使用自定义 2D 卷积内核锐化图像

首先定义一个自定义 2D 内核，然后使用 `filter2D()` 函数将卷积操作应用于图像。

```
"""
使用卷积核应用锐化
"""

kernel3 = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
sharp_img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel3)
```

```
cv2.imshow('Original', image)
cv2.imshow('Sharpened', sharp_img)

cv2.waitKey()
cv2.imwrite('sharp_image.jpg', sharp_img)
cv2.destroyAllWindows()
```

8.9 在 OpenCV 中对图像应用双边滤波

虽然模糊是减少图像中噪点的有效方法，但通常不希望模糊整个图像，因为重要的细节和锋利的边缘可能会丢失。在这种情况下，双边过滤可以更加方便。

- 此技术有选择地应用滤镜来模糊邻域中类似强度的像素。尽可能保留锋利的边缘。
- 它不仅允许控制滤镜的空间大小，还允许控制相邻像素在过滤输出中的包含程度。这是根据其颜色强度的变化以及与过滤像素的距离来完成的。

双边滤波本质上是对图像应用2D高斯（加权）模糊，同时还考虑了相邻像素强度的变化，以最小化近边缘的模糊。这意味着卷积核的形状实际上取决于每个像素位置的本地图像内容。

这是一个具体的例子。假设过滤图像中靠近边缘的区域。一个简单的高斯模糊滤镜会模糊边缘，因为它位于过滤区域附近（靠近高斯滤镜的中心）。但是双边滤波器可以感知边缘，因为它也考虑了像素强度的差异。因此，它将为跨越边缘的像素计算出低得多的权重，从而减少它们对过滤区域的影响。强度更均匀的区域模糊得更重，因为它们与强边缘无关。

OpenCV 提供了 `bilateralFilter()` 函数来过滤图像

```
bilateralFilter(src, d, sigmaColor, sigmaSpace)
#src:源图像。
#d:用于过滤的像素邻域的直径。
#sigmaColor:颜色强度分布
#sigmaSpace :空间分布的标准偏差。

#sigmaColor参数定义了一维高斯分布，它指定了可以容忍像素强度差异的程度。
#sigmaSpace参数在x和y方向上定义了内核的空间范围(就像前面描述的高斯模糊过滤器一样)。
```

过滤图像中像素的最终（加权）值是其空间和强度权重的乘积。因此

- 相似且接近过滤像素的像素将产生影响
- 远离滤波像素的像素几乎没有影响（由于空间高斯）
- 具有不同强度的像素几乎没有影响（由于颜色强度高斯），即使它们靠近卷积核的中心。

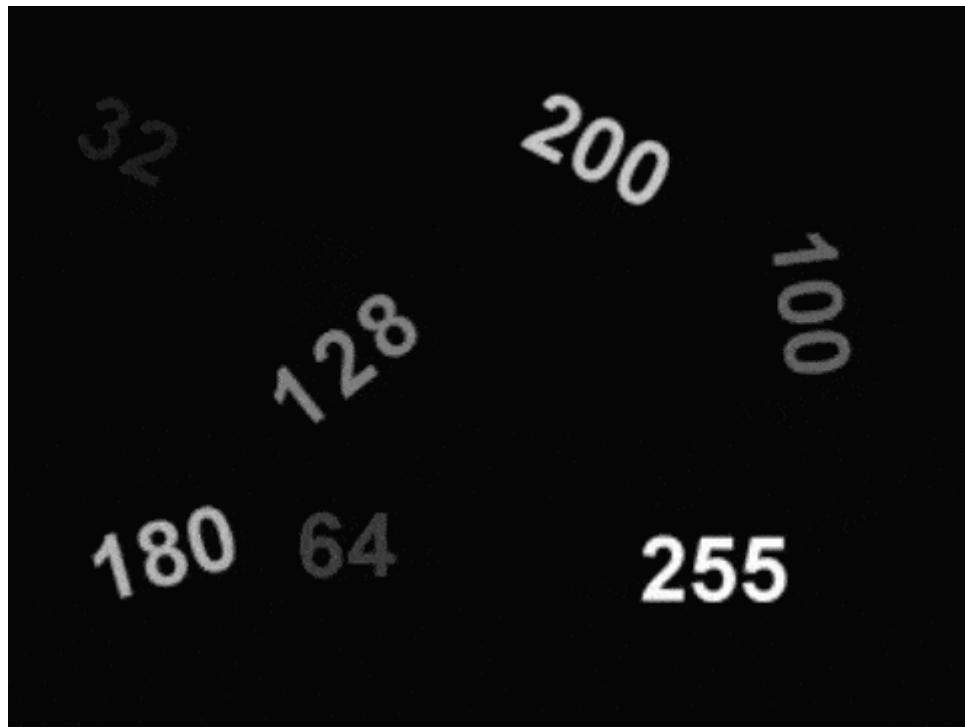
```
"""
应用双边过滤
"""

bilateral_filter = cv2.bilateralFilter(src=image, d=9, sigmaColor=75,
                                         sigmaSpace=75)

cv2.imshow('Original', image)
cv2.imshow('Bilateral Filtering', bilateral_filter)

cv2.waitKey(0)
cv2.imwrite('bilateral_filtering.jpg', bilateral_filter)
cv2.destroyAllWindows()
```

9. OpenCV 中的图像阈值



当图像在动画中循环时，将看到原始图像的阈值版本，其中：

- 所有数字看起来都是完全白色的（即它们的灰度值为 255）。
- 可以看到数字“5”，它在原始图像中存在但不可见，仅仅是因为它的灰度值为 5。

实际上，**原始图像中的所有数字都有一个等于数字值的灰度值**。因此，“255”是最亮的，“5”是最暗的。
回想一下，灰度强度范围从纯黑色（0）到纯白色（255）。

因此，读取阈值图像中的数字比读取原始图像中的数字要容易得多。即使是文本识别算法也发现处理阈值图像比原始图像更容易。因此，阈值在计算机视觉中有许多应用，并且通常在许多处理管道的初始阶段执行。有几种类型的阈值算法。让我们在这里关注“全局”阈值。

9.1 全局阈值

什么是“全局”阈值？当阈值规则平均应用于图像中的每个像素，并且阈值是固定的时，这些操作称为全局操作。

全局阈值算法将源图像（src）和阈值（thresh）作为输入，并通过比较源像素位置（x, y）处的像素强度与阈值来生成输出图像（dst）。如果 `src(x, y) > thresh`，则 `dst(x, y)` 被分配一些值。否则，`dst(x, y)` 被分配一些其他值。

最简单的全局阈值设置形式称为**二进制阈值**。

- 除了源图像（src）和阈值（thresh）之外，它还需要另一个称为最大值（ maxValue ）的输入参数。
- 在每个像素位置（x, y）处，将该位置的像素强度与阈值 thresh 进行比较。

如果 `src(x, y)` 大于 `thresh`，阈值操作会将目标图像像素的值设置为 `maxValue`。否则，它会将其设置为 0，如下面的伪代码所示。`dst(x, y)`

```
# 简单的阈值函数伪代码
if src(x,y) > thresh
    dst(x,y) = maxValue
else
    dst(x,y) = 0
```

阈值算法各不相同，基于应用于 `src(x, y)` 的不同**阈值规则**来获取 `dst(x, y)`。

9.1. 二进制阈值 (THRESH_BINARY)

阈值规则

```
# 二进制阈值
if src(x,y) > thresh
    dst(x,y) = maxValue
else
    dst(x,y) = 0
```

9.2. 逆二进制阈值 (THRESH_BINARY_INV)

```
# 逆二进制阈值
if src(x,y) > thresh
    dst(x,y) = 0
else
    dst(x,y) = maxValue
```

9.3. 截断阈值 (THRESH_TRUNC)

在这种类型的阈值中：

- 如果源像素值大于阈值，则目标像素设置为阈值 (`thresh`)。
- 否则，将其设置为源像素值
- `最大值` 被忽略

阈值规则

```
# 截断阈值
if src(x,y) > thresh
    dst(x,y) = thresh
else
    dst(x,y) = src(x,y)
```

9.4. 阈值为零 (THRESH_TOZERO)

在这种类型的阈值中，

- 如果源像素值大于阈值，则目标像素值设置为相应源的像素值。
- 否则，将其设置为零
- `最大值` 被忽略

```
# 阈值为零
if src(x,y) > thresh
    dst(x,y) = src(x,y)
else
    dst(x,y) = 0
```

9.5. 倒置阈值为零 (THRESH_TOZERO_INV)

在倒置阈值为零时，

- 如果源像素值大于阈值，则目标像素值设置为零。
- 否则，将其设置为源像素值
- 最大值 被忽略

```
# 倒置阈值为零
if src(x,y) > thresh
    dst(x,y) = 0
else
    dst(x,y) = src(x,y)
```

10. 使用OpenCV进行斑点检测

什么是斑点

斑点是图像中一组连接的像素，它们共享一些公共属性（例如灰度值）。在上图中，暗连接区域是斑点，斑点检测的目标是识别和标记这些区域

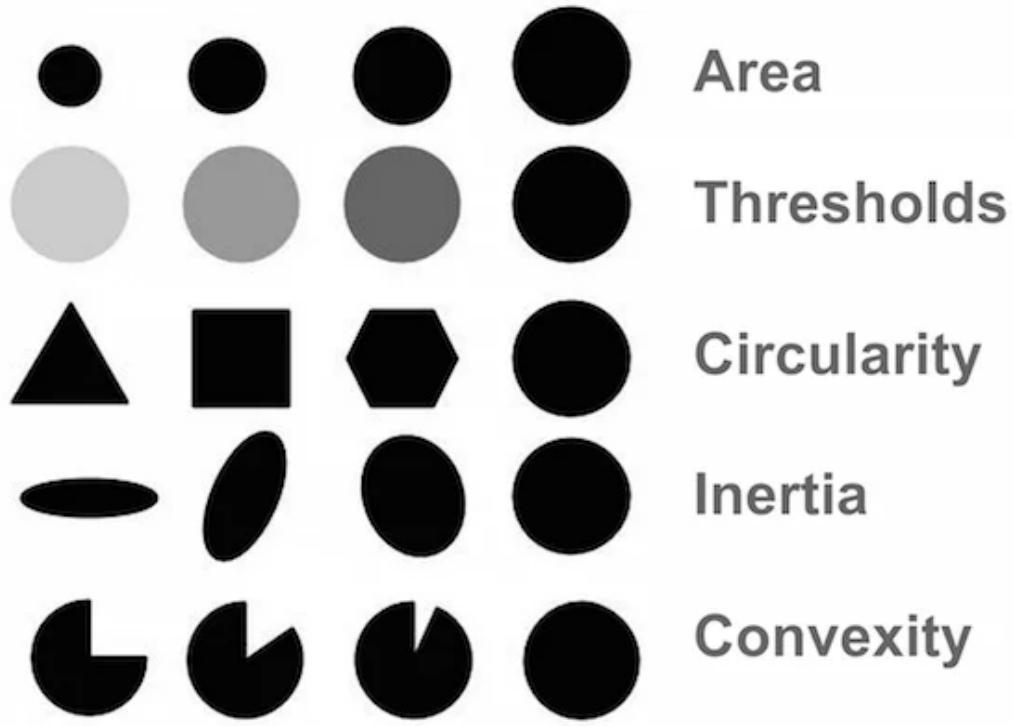
斑点检测如何工作

顾名思义，SimpleBlobDetector基于下面描述的相当简单的算法。该算法由参数控制（下面以粗体显示），并具有以下步骤。向下滚动以了解参数的设置方式。

1. **阈值**：通过使用从 `minThreshold` 开始的阈值对源图像进行阈值设置阈值，将源图像转换为多个二进制图像。这些阈值按阈值递增，直到**最大阈值**。所以第一个阈值是 `minThreshold`，第二个是 `minThreshold + thresholdStep`，第三个是 `minThreshold + 2 x thresholdStep`，依此类推。
2. **分组**：在每个二进制图像中，连接的白色像素被分组在一起。我们称这些为二进制 `blob`。
3. **合并**：计算二进制图像中二进制 `blob` 的中心，并合并位于 `minDistBetweenBlob` 更近的 `blob`。
4. **中心和半径计算**：将计算并返回新合并 `blob` 的中心和半径。

按颜色、大小和形状过滤斑点

可以设置 SimpleBlobDetector 的参数来过滤我们想要的 blob 类型。



1. 颜色 (Color) :

首先需要设置`filterByColor = 1`。设置`blobColor = 0`来选择较暗的斑点，设置`blobColor = 255`来选择较浅的斑点。

2按大小 (Area) :

通过设置参数`filterByArea = 1`，以及`minArea`和`maxArea`的适当值，可以根据大小过滤blob。例如，设置`minArea = 100`将过滤掉所有小于100像素的斑点。

3. 圆形度(Circularity) :

这只是测量斑点与圆的接近程度。例如，正六边形比正方形具有更高的圆度。要按循环度进行筛选，请将“按循环性筛选”= 1。然后为**最小圆度**和**最大圆度**设置适当的值。循环性定义为

$$\frac{4\pi Area}{(perimeter)^2}$$

这意味着圆的圆度为 1，正方形的圆度为 0.785，依此类推。

3.凸性(Convexity)

凸度定义为（斑点的面积/凸包的面积）。现在，形状的凸包是完全包围形状的最紧密的凸形状。要按凸度过滤，请将 `filterByConvexity = 1`，然后设置 `0 ≤ 最小凸性 ≤ 1` 和 `最大凸性 (≤ 1)`

4. 惯性比 (Inertia Ratio)

这衡量的是形状的细长程度。例如，对于圆，此值为 1，对于椭圆，该值介于 0 和 1 之间，对于直线，此值为 0。要按惯性比过滤，请将 `filterByInertia = 1` 设置为 1，并适当地将 `0 ≤ 最小惯性比 ≤ 1 ≤ 最大惯性比 (≤ 1)`。

```
# 设置SimpleBlobDetector参数。
params = cv2.SimpleBlobDetector_Params()

# 改变阈值
params.minThreshold = 10;
```

```
params.maxThreshold = 200;

#按区域过滤。
params.filterByArea = True
params.minArea = 1500

#圆形过滤
params.filterByCircularity = True
params.minCircularity = 0.1

#凹凸过滤
params.filterByConvexity = True
params.minConvexity = 0.87

#惯性过滤
params.filterByInertia = True
params.minInertiaRatio = 0.01

#创建一个带有参数的检测器
ver = (cv2.__version__).split('.')
if int(ver[0]) < 3 :
    detector = cv2.SimpleBlobDetector(params)
else :
    detector = cv2.SimpleBlobDetector_create(params)
```

11.使用 OpenCV 进行边缘检测

边缘检测是一种图像处理技术，用于识别对象的边界（边缘）或图像中的区域。边缘是与图像相关的最重要的功能之一。我们通过图像的边缘了解图像的底层结构。因此，计算机视觉处理管道在应用中广泛使用边缘检测。

如何检测边缘？

边缘的特征是像素强度的突然变化。为了检测边缘，我们需要在相邻像素中寻找这种变化。

OpenCV中可用的两种重要边缘检测算法的使用：[Sobel边缘检测](#) 和 [Canny边缘检测](#)。

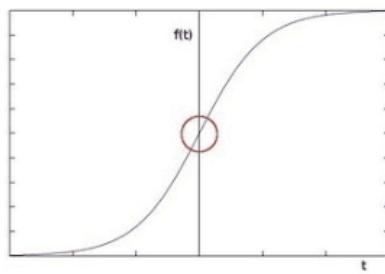
11.1读取图片

将彩色图像读取为灰度图像，因为不需要颜色信息来检测边缘。读取图像后，使用 `GaussianBlur()` 该功能对其进行模糊处理。这样做是为了减少图像中的噪点。在边缘检测中，必须计算像素强度的数值导数，这通常会导致边缘“噪声”。换句话说，图像中相邻像素（尤其是近边缘）的强度可能会波动很大，从而产生不代表我们正在寻找的主要边缘结构的边缘。**模糊可平滑边缘附近的强度变化，从而更容易识别图像中的主要边缘结构。**

11.2索贝尔边缘检测

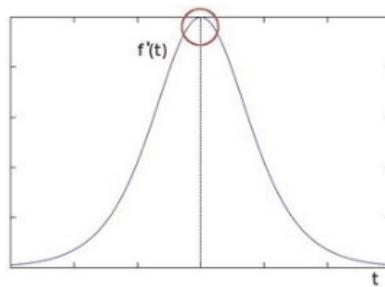
Sobel边缘检测是使用最广泛的边缘检测算法之一。Sobel 运算符检测像素强度突然变化的边缘，如下图所示。

Sobel边缘检测是使用最广泛的边缘检测算法之一。Sobel 运算符检测像素强度突然变化的边缘，如下图所示。



像素强度作为 t 的函数 ([来源](#))

当我们绘制强度函数的一阶导数时，强度的上升更加明显。



像素强度作为 t 函数的一阶导数 ([来
源](#))

上图表明，在梯度高于特定阈值的区域可以检测到边缘。此外，导数的突然变化也会揭示像素强度的变化。考虑到这一点，我们可以使用 3×3 卷积核近似导数。我们使用一个卷积核来检测 X 方向像素强度的突然变化，另一个内核在 Y 方向上。

[使用 OpenCV | 进行边缘检测学习公开简历# \(learnopencv.com\)](#)

OpenCV 应用 Sobel 边缘检测的语法：

```
Sobel(src, ddepth, dx, dy)
#src: 源图像
#ddepth: 指定输出图像的精度,
#dx 和 dy : 指定导数在每个方向上的顺序。例如:
#如果 dx=1 且 dy=0, 我们计算 x 方向的一阶导数 Sobel 图像。
#如果 dx=1 且 dy=1, 我们计算两个方向的一阶导数 Sobel 图像
```

11.3.Canny 边缘检测

Canny 边缘检测是当今使用的最流行的边缘检测方法之一，因为它非常强大和灵活。该算法本身遵循从图像中提取边缘的三阶段过程。添加图像模糊，这是减少噪声的必要预处理步骤。这使得它成为一个四阶段的过程，其中包括：

1. 降噪
2. 计算图像的强度梯度
3. 伪边缘的抑制
4. 滞后阈值

1.降噪

原始图像像素通常会导致边缘噪声，因此在计算边缘之前减少噪声非常重要。在Canny边缘检测中，高斯模糊滤镜用于基本上删除或最小化可能导致不良边缘的不必要的细节。

2.计算图像的强度梯度

图像被平滑（模糊）后，它会用 Sobel 内核进行水平和垂直过滤。然后，这些过滤操作的结果将用于计算每个像素的强度梯度幅度 (G) 和方向 (Θ)，如下所示。

$$G = \sqrt{G_x^2 + G_y^2} \quad (7)$$

$$\Theta = \tan^{-1} \left(\frac{G_x}{G_y} \right) \quad (8)$$

然后将渐变方向四舍五入到最接近的 45 度角。

3.伪边缘的抑制

在减少噪声并计算强度梯度后，此步骤中的算法使用一种称为边缘非最大抑制的技术来过滤掉不需要的像素（实际上可能不构成边缘）。为此，在正和负梯度方向上将每个像素与其相邻像素进行比较。

4.滞后阈值

在Canny边缘检测的最后一步中，将梯度幅度与两个阈值进行比较，一个阈值小于另一个阈值。

- 如果梯度幅度值高于较大的阈值，则这些像素与强边相关联，并包含在最终边图中。
- 如果梯度幅度值低于较小的阈值，则会抑制像素，并从最终边图中排除像素。
- 所有其他像素，其梯度幅度介于这两个阈值之间，被标记为“弱”边（即它们成为最终边图中的候选者）。
- 如果“弱”像素连接到与强边关联的像素，则它们也会包含在最终的边图中。

OpenCV 应用Canny边缘检测的语法：

```
Canny(image, threshold1, threshold2)
#image:源图片
#滞后处理的第一个阈值。
#滞后处理的第二个阈值。
```

在性能方面，Canny 边缘检测产生最佳结果，因为它不仅使用 Sobel 边缘检测，还使用非最大抑制和滞后阈值。这为如何在算法的最后阶段识别和连接边提供了更大的灵活性。

12.OpenCV GUI 中的鼠标和跟踪栏

1.使用鼠标注释图像

OpenCV 提供鼠标事件检测功能来检测各种鼠标操作，如左键单击和右键单击。

设置回调函数

首先定义一个特殊的“回调”函数，该函数将在命名窗口中显示的图像上绘制一个矩形。仅在检测到某些用户界面事件时调用，这种类型的函数被称为“回调”函数。在这里，这个特定的函数与鼠标事件相关联，因此将其定义为 `MouseCallback` 函数。命名为 `drawRectangle()`。无需为函数指定任何输入参数，当用户与鼠标交互时，这些参数将自动填充

定义此鼠标回调函数后，我们继续执行以下操作：

- 使用 `imread()` 从磁盘读取源图片
- 保存图片的副本，使用 `copy()`
- 创建命名窗口

调用 `setMouseCallback()` 来注册我们上面定义的鼠标回调函数，以及 `drawRectangle()` 来注册命名窗口中发生的鼠标事件。

setMouseCallback() 函数的一般语法

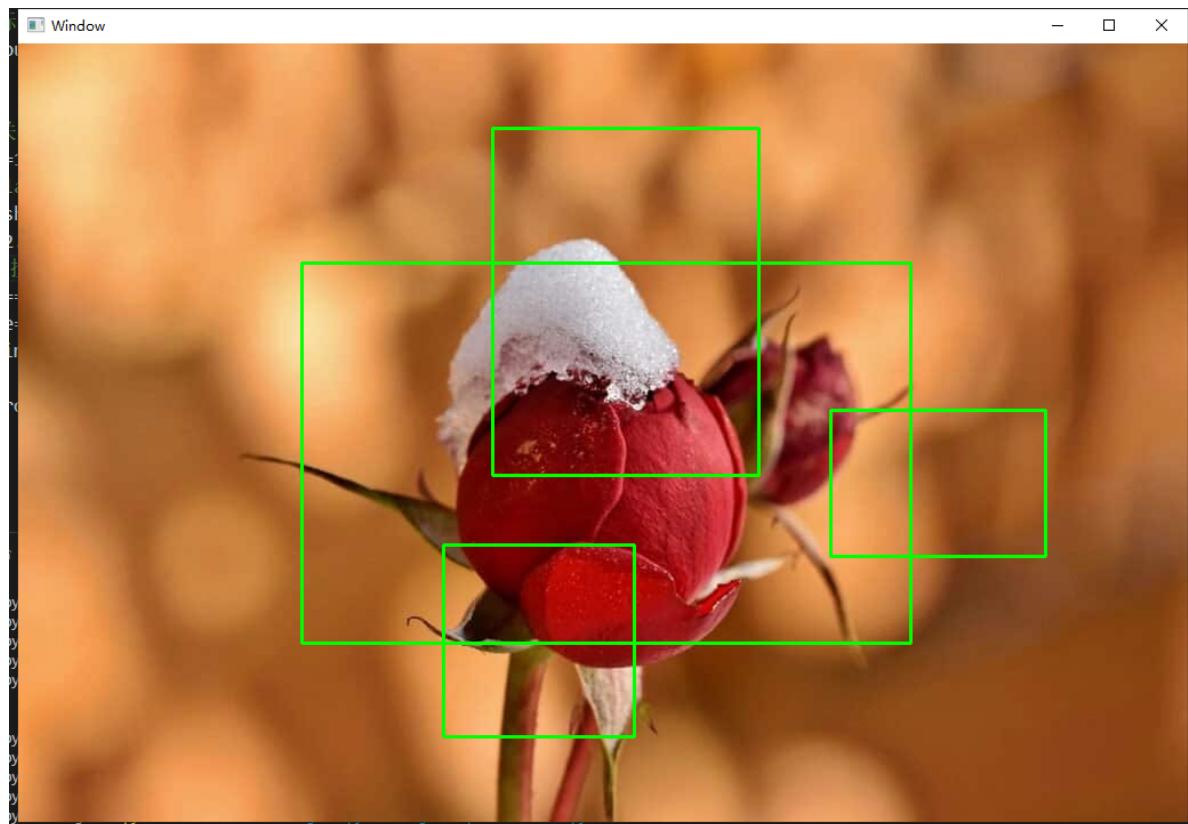
```
cv2.setMouseCallback(winname, onMouse, userdata)
```

#**winname**: 窗口的名称
#**onMouse**: 鼠标事件的回调函数
#**userdata**: 传递给回调的可选参数

在最后一步中，我们需要创建一个显示循环，允许用户与命名窗口进行交互。因此，在下面的代码中，我们：

- 创建一个持续显示图像的 `while` 循环，直到用户按 `q` 键（ASCII 代码：113）退出应用程序脚本。
- 在循环中为用户提供清除所有先前批注的功能。用户可以将命名窗口中的图像重置为我们在原始图像中读取时所做的副本。这可以通过键盘 `c`（ASCII 代码：99）来实现。
- 当用户退出循环时，我们使用 `destroyAllWindows()` 销毁窗口。

效果



2. 使用跟踪栏调整图像大小

- `scaleFactor` 用于缩放图像的回调函数。
- `最大缩放` 是跟踪栏位置将记录的最大值。最好将 100 作为最大值，然后我们可以使用跟踪栏位置直接将某些内容缩放为百分比。

使用 `imread()` 读取一个图像，并创建一个命名窗口。在 `namedWindow()` 函数中，我们传递 `WINDOW_AUTOSIZE` 标志。这很重要，因为它可以让我们调整图像大小。

我们现在准备讨论回调函数，当用户与跟踪栏交互时将调用它。定义跟踪栏回调函数与定义鼠标回调函数非常相似。只需符合 [TrackbarCallback](#) 函数签名，然后将该回调函数与跟踪栏关联。

- 使用函数签名定义回调函数，如下所示，用于 Python 和 C++。
- 和以前一样，随心所欲地命名函数。在本例中，我们将其命名为 `scaleImage()`。
- 在 Python 代码中，从 `args[0]` 中检索跟踪栏位置，范围为 0 到 100。
- 根据跟踪栏位置计算**比例因子**，并将该**比例因子**输入到 `resize()` 函数以调整图像大小。
- 在 Python 代码中，无需为图像指定任何数据类型。
- 要创建一个使用我们的回调函数的跟踪栏，我们需要调用 `createTrackbar()` 函数

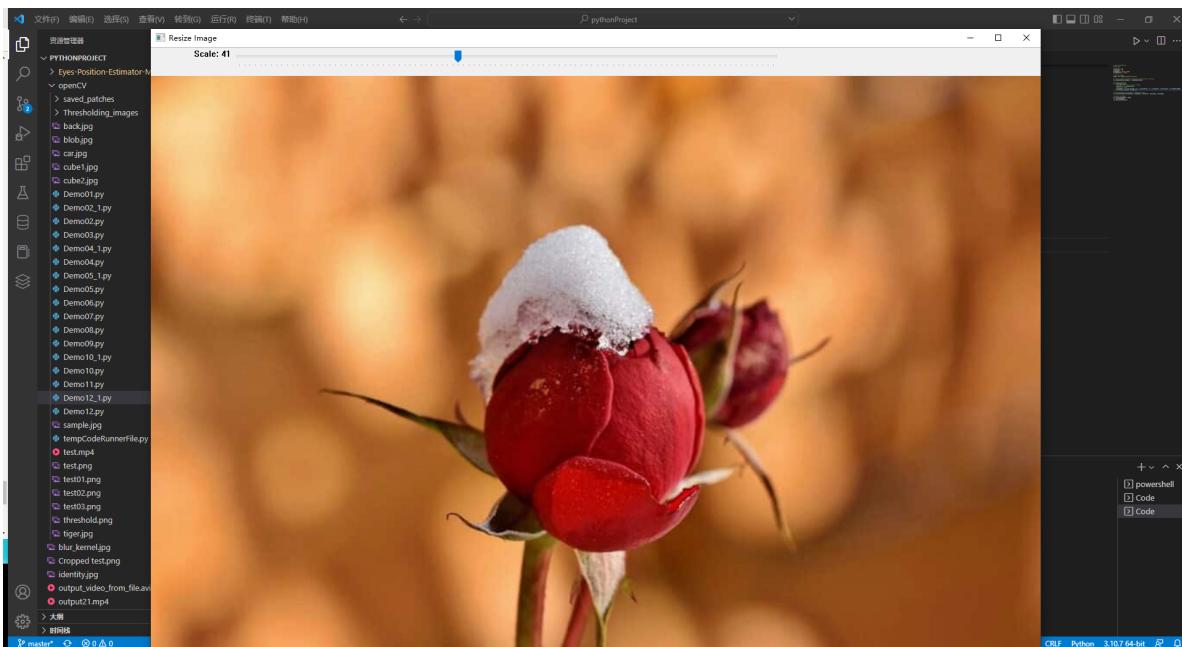
`createTrackbar()` 函数语法

```
cv2.createTrackbar( trackbarName, windowName, value, count, onChange)
```

#**trackbarName**: 创建的跟踪栏的名称。
#**windowName**: 创建的跟踪栏的父窗口的名称。
#**value**: 滑块的默认位置。这是可选的。
#**count**: 滑块将转到什么值。
#**onChange**: 回调函数。

最后，我们只需调用即可在命名窗口中显示图像。后跟，参数为零，无限期地显示窗口。`imshow()`
`waitKey(0)`

效果



13. 使用 OpenCV 进行轮廓检测

使用轮廓检测，我们可以检测物体的边界，并在图像中轻松定位它们。它通常是在许多应用的第一步，例如图像前景提取、简单图像分割、检测和识别。

13.1轮廓在计算机视觉中的应用

移动侦测 在监控视频中，运动检测技术具有多种应用，从室内外安全环境、交通控制、体育活动期间的行为检测、无人看管物体的检测，甚至视频压缩。

无人看管的物体检测：公共场所的任何无人看管的物体通常被视为可疑物体。

背景/前景分割:将图像的背景替换为另一个图像，您需要执行图像前景提取（类似于图像分割）。使用等值线是一种可用于执行分割的方法。

13.2什么是等高线

当我们连接对象边界上的所有点时，我们得到一个轮廓。通常，特定轮廓是指具有相同颜色和强度的边界像素。OpenCV使得在图像中查找和绘制轮廓变得非常容易。它提供了两个简单的基本功能：

1. `findContours()`
2. `drawContours()`

此外，它还具有两种不同的轮廓检测算法：

1. `CHAIN_APPROX_SIMPLE`
2. `CHAIN_APPROX_NONE`

13.3在 OpenCV 中检测和绘制轮廓的步骤

OpenCV使这成为一个相当简单的任务。只需按照以下步骤操作：

1.读取图像并将其转换为灰度格式

读取图像并将图像转换为灰度格式。将图像转换为灰度非常重要，因为它为下一步准备了图像。将图像转换为单通道灰度图像对于阈值设置非常重要，而阈值检测算法又是轮廓检测算法正常工作所必需的。

2.应用二进制阈值

查找轮廓时，首先始终对灰度图像应用二进制阈值或Canny边缘检测。在这里，我们将应用二进制阈值。这会将图像转换为黑白，突出显示感兴趣的对象，使轮廓检测算法变得容易。阈值设置将图像中对象的边框完全变为白色，所有像素都具有相同的强度。该算法现在可以从这些白色像素中检测对象的边框。

注： 值为 0 的黑色像素被视为背景像素并被忽略。

3.查找轮廓

使用 `findContours()` 函数检测图像中的轮廓。

`findContours()` 函数语法

```
findContours(image, mode, method)
#上一步获得的二进制输入图像
#这是轮廓检索模式。默认: RETR_TREE, 这意味着算法将从二进制图像中检索所有可能的轮廓。
#这定义了轮廓近似方法。将使用 CHAIN_APPROX_NONE。虽然比CHAIN_APPROX_SIMPLE稍慢, 但使用这种方法来存储所有轮廓点。
```

4.在原始 RGB 图像上绘制轮廓。

识别轮廓后，使用 `drawContours()` 函数将轮廓叠加在原始 RGB 图像上。

drawContours () 函数语法

```
drawContours(image, contours, contourIdx, color, thickness)
```

#image: 在其上绘制轮廓的输入的 RGB 图像。

#contours: 表示从 `findContours()` 函数获得的轮廓。

#contourIdx: 等值线点的像素坐标列在获得的等值线中。使用此参数，可以指定此列表中的索引位置，以准确指示要绘制的轮廓点。提供负值将绘制所有等值线点。

#color: 这表示要绘制的等值线点的颜色。

#thickness: 这是等高点的厚度。

13.4 使用CHAIN_APPROX_SIMPLE绘制轮廓

```
#使用cv2.CHAIN_APPROX_SIMPLE检测二值图像上的轮廓。
contours1, hierarchy1 = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
#在'CHAIN_APPROX_SIMPLE'的原始图像上绘制轮廓
image_copy1 = image.copy()
cv2.drawContours(image_copy1, contours1, -1, (0, 255, 0), 2, cv2.LINE_AA)
#查看结果
cv2.imshow('Simple approximation', image_copy1)
cv2.waitKey(0)
cv2.imwrite('contours_simple_image1.jpg', image_copy1)
cv2.destroyAllWindows()
```

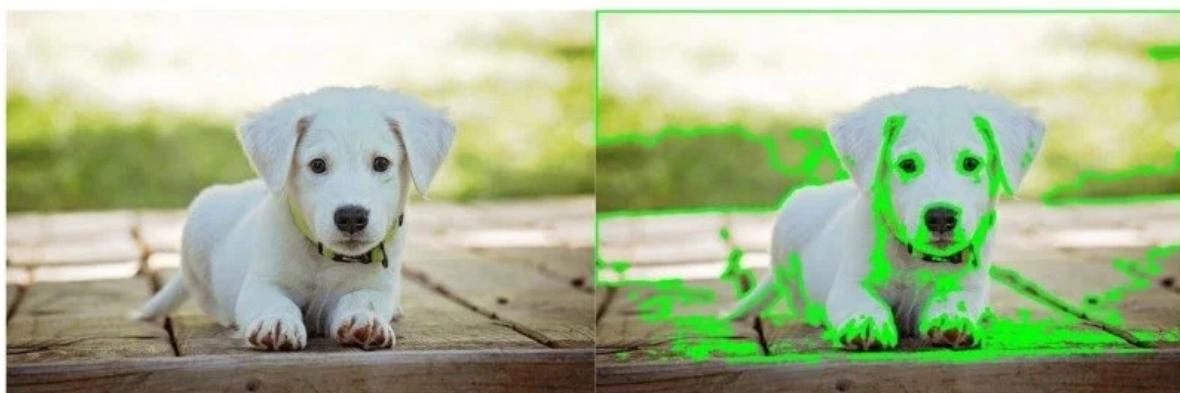
13.5 不同的轮廓检索技术

[使用 OpenCV 进行轮廓检测 \(Python/C++\) \(learnopencv.com\)](#)

OpenCV中有四种轮廓检索技术，即 `RETR_LIST`，`RETR_EXTERNAL` 和 `RETR_CCOMP` 以及前面使用的 `RETR_TREE`。

13.6 局限性

当图像中的对象与其背景形成强烈对比时，可以清楚地识别与每个对象关联的轮廓。但是，如果有一个图像，如下面的图 16它不仅有一个明亮的物体（小狗），还有一个与感兴趣对象（小狗）具有相同值（亮度）的背景。右侧图像中的轮廓甚至不完整。此外，背景区域中还有多个不需要的轮廓突出。



当图像中物体的背景充满线条时，轮廓检测也可能失败。

- OpenCV中的轮廓检测算法在图像具有深色背景和明确定义的兴趣对象时效果很好。
- 但是，当输入图像的背景杂乱无章或像素强度与感兴趣对象相同时，算法的表现就不那么好了。

14. 使用 OpenCV DNN 模块进行深度学习

通过使用OpenCV DNN模块，我们可以对图像和视频进行基于深度学习的计算机视觉推理。***让我们看一下它支持的所有功能。** * 有趣的是，我们能想到的大多数深度学习和计算机视觉任务都得到了支持。以下列表将让我们很好地了解这些功能。

1. 图像分类。
2. 对象检测。
3. 图像分割。
4. 文本检测和识别。
5. 姿势估计。
6. 深度估计。
7. 人和人脸验证和检测。
8. 行人重识别，也被称为 ReID(Person Re-identification)，是希望能够利用计算机视觉的算法来进行跨摄像头的追踪，也就是找到不同摄像头下的同一个人，这也是图像检索的一个子问题

15. 字体

```
img = cv.putText(img, text, org, fontFace, fontScale, color, thickness=1,
lineType= 8, bottomLeftOrigin=False)
#img:图片 text:文本 org:文字坐标
# fontFace:声明的字体类型
    # FONT_HERSHEY_SIMPLEX = 0
    #FONT_HERSHEY_PLAIN = 1
    #FONT_HERSHEY_DUPLEX = 2
    #FONT_HERSHEY_COMPLEX = 3
    #FONT_HERSHEY_TRIPLEX = 4
    #FONT_HERSHEY_COMPLEX_SMALL = 5
    #FONT_HERSHEY_SCRIPT_SIMPLEX = 6
    #FONT_HERSHEY_SCRIPT_COMPLEX = 7
#fontScale:字体范围因子
#color:字体颜色
#thickness:字体厚度
#lineType: 行宽
    # cv2.LINE_4 、 cv2.LINE_8 和 cv2.LINE_AA
#bottomLeftOrigin:如果 bottomLeftOrigin = False 则为左上角, 否则为左下角
```

[OpenCV文本绘制详解 - 掘金\(juejin.cn\)](#)

2. mediapipe

[Home - mediapipe \(google.github.io\)](#)

MediaPipe 主要概念

`MediaPipe` 的核心框架由 `C++` 实现，并提供 `Java` 以及 `Objective C` 等语言的支持。`MediaPipe` 的主要概念包括数据包 (`Packet`)、数据流 (`Stream`)、计算单元 (`Calculator`)、图 (`Graph`) 以及子图 (`Subgraph`)。数据包是最基础的数据单位，一个数据包代表了在某一特定时间节点的数据，例如一帧图像或一小段音频信号；数据流是由按时间顺序升序排列的多个数据包组成，一个数据流的某一特定时间戳 (`Timestamp`) 只允许至多一个数据包的存在；而数据流则是在多个计算单元构成的图中流动。`MediaPipe` 的图是有向的——数据包从数据源 (`Source Calculator` 或者 `Graph Input Stream`) 流入图直至在汇聚结点 (`Sink Calculator` 或者 `Graph Output Stream`) 离开。

solution(解决方案)

- `Face Detection` (人脸检测)
- `Face Mesh` (人脸网格)
- `Iris` (虹膜)
- `Hands` (手)
- `PoseHolistic` (整体姿势)
- `Selfie Segmentation` (自拍人像抠图)
- `Hair Segmentation` (头发分割)
- `Object Detection` (对象检测)
- `Box Tracking` (目标追踪)
- `Instant Motion Tracking` (即时运动跟踪)
- `Objectron (3D ObjectDetection)` (实时3D物体检测)
- `KNIFT (Template-based Feature Matching)` (基于模板的特征匹配)
- `AutoFlip (Saliency-aware Video Cropping)` (Saliency-aware视频裁剪)
- `Dataset Preparation with MediaSequence` (使用MediaSequence准备数据集)
- `YouTube-8M Feature Extraction and ModelInference` (YouTube-8M特征提取与模型推断)
- `Models and Model Cards` (模型和模型卡)

1.开始

Windows

1. 从 [Protobuf GitHub 存储库](#) 下载最新的 `protoc win64 zip`，解压缩文件，然后将 `protoc.exe` 可执行文件复制到首选位置。请确保将位置添加到 Path 环境变量中。
2. 激活 Python 虚拟环境。

```
$ python3 -m venv mp_env && source mp_env/bin/activate
```

3. 在虚拟环境中，转到 `MediaPipe` 存储库目录。
4. 安装所需的 Python 包。

```
(mp_env)mediapipe$ pip3 install -r requirements.txt
```

5. 生成并安装 `MediaPipe` 包。

```
(mp_env)mediapipe$ python3 setup.py install --link-opencv
```

或

```
(mp_env)mediapipe$ python3 setup.py bdist_wheel
```

2. Face Detection

MediaPipe 人脸检测是一种超快人脸检测解决方案，具有 6 个地标和多人脸支持。它基于 [BlazeFace](#)，这是一款专为移动 GPU 推理量身定制的轻量级且性能良好的人脸检测器。该探测器的超实时性能使其能够应用于任何需要精确感兴趣的面部区域作为其他任务特定模型输入的实时取景器体验，例如 3D 面部关键点估计（例如 [MediaPipe 面部网格](#)）、面部特征或表情分类以及面部区域分割。BlazeFace 使用受 [MobileNetV1/V2](#) 启发但又不同的轻量级特征提取网络、一种从[单发多盒检测器 \(SSD\)](#) 修改而来的 GPU 友好锚点方案，以及一种改进的平局解析策略，以替代非最大抑制。

Python Solution API

支持配置选项：

```
model_selection  
min_detection_confidence
```

3. Face Mesh

MediaPipe Face Mesh是一种实时估计468个3D面部地标的解决方案，即使是在移动设备上。它使用机器学习(ML)来推断3D面部表面，只需要一个单一的摄像头输入，而不需要专用的深度传感器。该解决方案利用轻量级模型架构和GPU加速，为实时体验提供了至关重要的实时性能。此外，该解决方案与人脸变换模块捆绑在一起，该模块弥合了人脸地标估计和有用的实时增强现实(AR)应用之间的差距。它建立了一个度量3D空间，并使用人脸地标屏幕位置来估计该空间中的人脸变换。人脸变换数据由常见的三维原语组成，包括人脸姿态变换矩阵和三角人脸网格。在底层，一种称为Procrustes analysis的轻量级统计分析方法被用于驱动一个健壮的、性能良好的、可移植的逻辑。分析运行在CPU上，在ML模型推理之上具有最小的速度/内存占用。

机器学习管道

我们的 ML 管道由两个协同工作的实时深度神经网络模型组成：一个在完整图像上运行并计算人脸位置的检测器，以及一个在这些位置上运行并通过回归预测近似 3D 表面的 3D 人脸地标模型。精确裁剪人脸可大大减少对常见数据增强的需求，例如由旋转、平移和比例变化组成的仿射变换。相反，它允许网络将其大部分容量用于坐标预测精度。此外，在我们的管道中，还可以根据前一帧中识别的人脸特征点生成裁剪，并且只有当地标模型无法再识别人脸存在时，才会调用人脸检测器来重新定位人脸。此策略类似于我们的 [MediaPipe Hands](#) 解决方案中采用的策略，该解决方案使用手掌检测器和手部地标模型。

模型

人脸特征点检测

对于3D面部特征点，我们采用了迁移学习，并训练了一个具有多个目标的网络：该网络同时预测合成渲染数据上的3D地标坐标和注释真实世界数据的2D语义轮廓。由此产生的网络不仅为我们提供了合理的3D地标预测，还针对合成数据以及真实世界数据。

3D 人脸特征点网络检测接收裁剪的视频帧作为输入，而无需额外的深度输入。该模型输出 3D 点的位置，以及人脸在输入中存在并合理对齐的概率。一种常见的替代方法是预测每个检测的 2D 热图，但它不适合深度预测，并且对于如此多的点具有很高的计算成本。我们通过迭代引导和优化预测来进一步提高模型的准确性和鲁棒性。这样，我们可以将数据集扩展到越来越具有挑战性的情况，例如鬼脸、斜角和遮挡。

注意力网格模型

除了面部特征点模型之外，我们还提供了另一个模型，该模型将[注意力](#)应用于语义上有意义的面部区域，从而更准确地预测嘴唇、眼睛和虹膜周围的[特征点](#)，但代价是更多的计算。

Python Solution API

支持配置选项：

- static_image_mode
- max_num_faces
- refine_landmarks
- min_detection_confidence
- min_tracking_confidence

STATIC_IMAGE_MODE

如果设置为 `false`，该解决方案将输入图像视为视频流。它将尝试在第一个输入图像中检测人脸，并在成功检测后进一步定位人脸地标。在随后的图像中，一旦检测到所有 `max_num_faces` 人脸并本地化了相应的人脸标记，它只需跟踪这些标记，而不调用另一个检测，直到失去对任何人脸的跟踪。这减少了延迟，非常适合处理视频帧。如果设置为 `true`，则在每个输入图像上运行人脸检测，非常适合处理一批静态的、可能不相关的图像。默认为 `false`

MAX_NUM_FACES

要检测的最大人脸数量。默认为 `1`。

REFINE_LANDMARKS

是否进一步细化眼睛和嘴唇周围的检测坐标，并通过应用注意力网格模型输出虹膜周围的额外坐标。默认 `false`。

MIN_DETECTION_CONFIDENCE

来自人脸检测模型的最小置信值 (`[0.0, 1.0]`)，使检测被认为是成功的。默认为 `0.5`。

输出

MULTI_FACE_LANDMARKS

被检测/跟踪的人脸集合，其中每个人脸表示为468个人脸坐标的列表，每个坐标由x, y和z组成，x和y分别由图像宽度和高度归一化。Z表示坐标深度，头部中心的深度为原点`[0.0, 1.0]`，Z和x的值越小，坐标离相机越近。z的大小与x的大小大致相同。

4.Iris

广泛的实际应用，包括计算摄影（闪光反射）和增强现实效果（虚拟化身），都依赖于精确跟踪眼睛内的虹膜。由于计算资源有限、光线条件多变以及存在遮挡（例如头发或人眯眼），因此在移动设备上解决这是一项具有挑战性的任务。光圈跟踪还可用于确定相机到用户的公制距离。这可以改进各种用例，从虚拟试戴适当尺寸的眼镜和帽子到根据观看者的距离采用字体大小的辅助功能。通常，使用复杂的专用硬件来计算公制距离，从而限制了可以应用解决方案的设备范围。

MediaPipe 虹膜是一种用于精确虹膜估计的 ML 解决方案，能够使用单个 RGB 相机实时跟踪涉及虹膜、瞳孔和眼睛轮廓的地标，而无需专门的硬件。通过使用虹膜地标，该解决方案还能够确定主体和相机之间的公制距离，相对误差小于 10%。请注意，虹膜跟踪不会推断人们正在看的位置，也不会提供任何形式的身份识别。借助 MediaPipe 框架的交叉平台功能，MediaPipe Iris 可以在大多数现代手机、台式

机/笔记本电脑甚至网络上运行。

5.Hands

概述

感知手的形状和运动的能力可以成为改善各种技术领域和平台用户体验的重要组成部分。例如，它可以构成手语理解和手势控制的基础，还可以在增强现实中将数字内容和信息叠加在物理世界之上。虽然对人们来说很自然，但强大的实时手部感知是一项极具挑战性的计算机视觉任务，因为手经常遮挡自己或彼此（例如手指/手掌遮挡和握手），并且缺乏高对比度模式。

MediaPipe Hands是一种高保真手和手指跟踪解决方案。它采用机器学习（ML）从单个帧推断出一只手的21个3D坐标。虽然当前最先进的方法主要依赖于强大的桌面环境进行推理，但我们的方法在手机上实现了实时性能，甚至可以扩展到多手。我们希望，为更广泛的研发社区提供这种手部感知功能将导致创造性用例的出现，刺激新的应用和新的研究途径。

Python Solution API

支持配置选项：

- static_image_mode
- max_num_hands
- model_complexity
- min_detection_confidence
- min_tracking_confidence

static_image_mode

如果设置为false，则解决方案将输入图像视为视频流。它将尝试在第一个输入图像中检测手，并在成功检测后进一步定位手坐标。在后续的图像中，一旦检测到所有的max_num_hands手并本地化了相应的手标记，它只需跟踪这些标记，而不调用另一个检测，直到失去任何手的跟踪。这减少了延迟，非常适合处理视频帧。如果设置为true，手部检测将在每个输入图像上运行，这非常适合处理一批静态的、可能不相关的图像。默认为false。

MAX_NUM_HANDS

要检测的最大手数。默认为2。

MODEL_COMPLEXITY

手部坐标模型的复杂度：0或1。坐标的准确性和推理延迟通常随着模型的复杂性而上升。默认为1

MIN_DETECTION_CONFIDENCE

来自手部检测模型的最小置信值([0.0,1.0])被认为检测成功。默认为0.5。

MIN_TRACKING_CONFIDENCE:

标志跟踪模型中的最小置信度值([0.0,1.0])被认为是成功跟踪的手部标志，否则将在下一个输入图像上自动调用手部检测。将其设置为较高的值可以增加解决方案的健壮性，但代价是增加延迟。如果static_image_mode为true则忽略，其中手检测只是在每个图像上运行。默认为0.5。

Output

MULTI_HAND_LANDMARKS

被检测/跟踪的手的集合，其中每只手表示为21个手部标记的列表，每个标记由x, y和z组成，x和y分别由图像宽度和高度归一化为[0.0,1.0]。Z表示以手腕处的深度为原点的地标深度，该值越小，地标越接近摄像机。z的大小与x的大小大致相同。

MULTI_HAND_WORLD_LANDMARKS

被检测/跟踪的手的集合，其中每只手都表示为世界坐标中21个手坐标的列表。每个坐标都由x、y和z组成:以米为单位的真实3D坐标，原点位于手部的近似几何中心。

MULTI_HANDEDNESS

被检测/追踪的手的惯用手的集合(即它是左手还是右手)。每只手都由标签和分数组成。label是一个值为"Left"或"Right"的字符串。Score是预测惯用手的估计概率，它总是大于或等于0.5(反惯用手的估计概率为 $1 - \text{Score}$)。

注意，假设输入图像是镜像的，也就是说，用前置/自拍相机拍摄的图像水平翻转。如果不是这样，请交换应用程序中的惯用手输出。