使用OpenCV在Python中進行圖像操作

原創 磐慰慰 深度學習與計算機視覺 2022-02-13 22:04

介紹

眾所周知，OpenCV是一個用於計算機視覺和圖像操作的免費開源庫。

OpenCV 是用C++ 編寫的，並且有數千種優化的算法和函數用於各種圖像操作。很多現實生活中的操作都可以使用OpenCV 來解決。例如視頻和圖像分析、實時計算機視覺、對象檢測、鏡頭分析等。

許多公司、研究人員和開發人員為OpenCV 的創建做出了貢獻。使用OpenCV 很簡單，而且OpenCV 配備了許多工具和功能。讓我們使用OpenCV 來執行有趣的圖像操作並查看結果。

形態變換

形態變換是基於形狀變換圖像的圖像處理方法。這個過程有助於區域形狀的表現和刻畫。這些轉換使用應用於輸入圖像的結構元素，並生成輸出圖像。

形態學操作有多種用途，包括從圖像中去除噪聲、定位圖像中的強度凹凸和孔洞，以及連接圖像中的不同元素。

有兩種主要的形態學變換類型：腐蝕和膨脹。

腐蝕

腐蝕是為了減小前景對象的大小而執行的形態學操作。異物的邊界被慢慢腐蝕。腐蝕在圖像編輯和轉換中有許多應用，腐蝕會縮小圖像像素。對象邊界上的像素也被刪除。

腐蝕的實現在Python 中很簡單，可以在內核的幫助下實現。

讓我們開始使用Python 中的代碼來實現腐蝕。

首先，我們導入Open CV 和Numpy。

```
import  cv2
import  numpy  as  np
```

現在我們讀取圖像。

```
image = cv2.imread( "image1.jpg" )
```

圖片:

我們創建了一個執行腐蝕操作所需的內核，並使用內置的OpenCV 函數實現它。

```
# Creating kernel
 kernel = np.ones(( 5 ,  5 ), np.uint8)
# Using cv2.erode() method
 image_erode = cv2.erode(image, kernel)
```

現在，我們保存文件並查看。

```
filename =  'image_erode1.jpg'
# Using cv2.imwrite() method
# Saving the image
 cv2.imwrite(filename, image_erode)
```

圖片：

正如我們所看到的，圖像現在被腐蝕了，銳度和邊緣都減少了，圖像變得模糊了。腐蝕可用於隱藏或刪除圖像的某些部分或隱藏圖像中的信息。

讓我們嘗試不同類型的腐蝕。

```
kernel2 = np.ones(( 3 ,  3 ), np.uint8)
image_erode2 = cv2.erode(image, kernel2, cv2.BORDER_REFLECT)
```

現在，我們保存圖像文件。

```
filename = 'image_erode2.jpg'
# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, image_erode2)
```

图片：

现在，让我们看看什么是膨胀。

膨胀

膨胀过程与腐蚀相反。图像膨胀时，前景对象不是缩小，而是扩大。图像里的东西在边界附近扩张，并形成一个膨胀的物体。

图像中的明亮区域在膨胀后往往会“发光”，这通常会导致图像增强。因此，膨胀用于图像校正和增强。

让我们使用 Python 代码实现 Dilation。

```
kernel3 = np.ones((5,5), np.uint8)
image_dilation = cv2.dilate(image, kernel, iterations=1)
```

现在，我们保存图像。

```
filename = 'image_dilation.jpg'
# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, image_dilation)
```

图片：

正如我们所见，图像现在更亮，强度更高。

创建边框

为图像添加边框非常简单，我们的手机图库应用程序或编辑应用程序可以非常快速地完成。但是，现在让我们使用 Python 为图像创建边框。

```
## Using cv2.copyMakeBorder() method
image_border1 = cv2.copyMakeBorder(image, 25, 25, 10, 10, cv2.BORDER_CONSTANT, No
```

现在，让我们保存图像。

```
filename = 'image_border1.jpg'
# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, image_border1)
```

图片：

在这里，我们为图像添加了一个简单的黑色边框。现在，让我们尝试一些镜像边框。

```
#making a mirrored border
image_border2 = cv2.copyMakeBorder(image, 250, 250, 250, 250, cv2.BORDER_REFLECT)
```

现在，我们保存图像。

```
filename = 'image_border2.jpg'
# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, image_border2)
```

图片：

这很有趣，它看起来像是奇异博士的镜子维度中的东西。

让我们试试别的。

```
#making a mirrored border
image_border3 = cv2.copyMakeBorder(image, 300, 250, 100, 50, cv2.BORDER_REFLECT)
```

现在，我们保存图像。

```
filename = 'image_border3.jpg'
# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, image_border3)
```

图片：

强度变换

通常，由于各种原因，图像会发生强度变换。这些是在空间域中直接在图像像素上完成的。图像阈值处理和对比度处理等操作是使用强度转换完成的。

对数变换

对数变换是一种强度变换操作，其中图像中的像素值被替换为它们的对数值。

对数变换用于使图像变亮或增强图像，因为它将图像中较暗的像素扩大到较高的像素值。

让我们实现对数变换。

```
# Apply log transform.
c = 255/(np.log(1 + np.max(image)))
log_transformed = c * np.log(1 + image)
```

```
# Specify the data type.
log_transformed = np.array(log_transformed, dtype = np.uint8)
```

现在，我们保存图像。

```
cv2.imwrite('log_transformed.jpg', log_transformed)
```

图片：

图像变得非常明亮。

线性变换

我们将对图像应用分段线性变换。这种变换也是在空间域上完成的。此方法用于为特定目的修改图像。它被称为分段线性变换，因为它只有一部分是线性的。最常用的分段线性变换是对比拉伸。

通常，如果在低光照条件下单击图像并且周围照明不佳，则生成的图像对比度较低。对比度拉伸会增加图像中强度级别的范围，并且对比度拉伸函数会单调增加，从而保持像素强度的顺序。

现在，让我们实现对比度拉伸。

```
def pixelVal(pix, r1, s1, r2, s2):
    if (0 <= pix and pix <= r1):
        return (s1 / r1)*pix
    elif (r1 < pix and pix <= r2):
        return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
    else:
        return ((255 - s2)/(255 - r2)) * (pix - r2) + s2
# Define parameters.
r1 = 70
s1 = 0
r2 = 140
s2 = 255
# Vectorize the function to apply it to each value in the Numpy array.
pixelVal_vec = np.vectorize(pixelVal)
# Apply contrast stretching.
contrast_stretch = pixelVal_vec(image, r1, s1, r2, s2)
# Save edited image.
cv2.imwrite('contrast_stretch.jpg', contrast_stretch)
```

图片：

在这里，图像得到了改善，并且可以观察到更高的对比度。

去噪彩色图像

去噪信号或图像意味着去除不必要的信号和信息以获得有用的信号和信息。去噪以去除不需要的噪声，并更好地分析和处理图像。

让我们用 Python 对彩色图像进行去噪。

```
denoised_image = cv2.fastNlMeansDenoisingColored(image, None, 15, 8, 8, 15)
```

现在，我们保存图像。

```
# Save edited image.
cv2.imwrite('denoised_image.jpg', denoised_image)
```

图片：

我们可以看到很多想要的东西，比如背景和天空已经被删除了。

使用直方图分析图像

在任何形式的分析中，直方图都是必不可少的视觉效果。图像的直方图是理解全局描述的一种令人兴奋的方式，直方图可用于对图像进行定量分析。图像直方图表示图像中灰度级的出现。

我们可以使用直方图来了解数字图像的像素强度分布，也可以使用直方图来了解主色。

让我们绘制一个直方图。

```
from matplotlib import pyplot as plt
histr = cv2.calcHist([image],[0],None,[256],[0,256])
plt.plot(histr)
```

输出：


```
# alternative way to find histogram of an image
plt.hist(image.ravel(),256,[0,256])
plt.show()
```

输出：

该图显示了图像上 0 到 255 颜色范围内的像素数。我们可以看到，所有类型的颜色都有良好的分布。

现在，让我们将图像转换为黑白并生成直方图。

```
grey_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
histogram = cv2.calcHist([grey_image], [0], None, [256], [0, 256])
plt.plot(histogram, color='k')
```

输出：

这个分布和之前的分布有很大的不同。这主要是因为图像被转换为灰度，然后进行分析。

现在，我们执行颜色直方图。

```
for i, col in enumerate(['b', 'g', 'r']):
    hist = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(hist, color = col)
    plt.xlim([0, 256])
plt.show()
```

输出：

我们可以看到蓝色和绿色的像素数量远高于红色。这很明显，因为图像中有很多蓝色和绿色区域。

所以我们可以看到，绘制图像直方图是理解图像强度分布的好方法。

我们看了一些令人兴奋的计算机视觉应用。

检查代码：https://github.com/prateekmaj21/Image-Processing-Tasks/tree/main/OPEN%20CV%20Image%20Operations

☆ END ☆

如果看到这里，说明你喜欢这篇文章，请转发、点赞。微信搜索「uncle_pn」，欢迎添加小编微信「woshicver」，每日朋友圈更新一篇高质量博文。

↓掃描二維碼添加小編↓

喜歡此內容的人還喜歡

基於OpenCV的特定區域提取

小白學視覺

Opencv實現透視形變

小白學視覺

基於python和OpenCV構建智能停車系統

小白學視覺