# 参考1

## **基础**

在我们每个人的家中可能都会几张退化的老照片，有时候上面不小心在上面弄上了点污渍或者是画了几笔。你有没有想过要修复这些照片呢？我们可以使用笔刷工具轻易在上面涂抹两下，但这没用，你只是用白色笔画取代了黑色笔画。此时我们就要求助于图像修补技术了。这种技术的基本想法很简单：使用坏点周围的像素取代坏点，这样它看起来和周围像素就比较像了。

为了实现这个目的，科学家们已经提出了好几种算法，OpenCV 提供了其中的两种。这两种算法都可以通过使用函数 cv2.inpaint() 来实施。  
第一个算法是根据 Alexandru\_Telea 在 2004 发表的文章实现的。它是基于快速行进算法的。以图像中一个要修补的区域为例。算法从这个区域的边界开始向区域内部慢慢前进，首先填充区域边界像素。它要选取待修补像素周围的一个小的邻域，使用这个邻域内的归一化加权和更新待修复的像素值。权重的选择是非常重要的。对于靠近带修复点的像素点，靠近正常边界像素点和在轮廓上的像素点给予更高的权重。当一个像素被修复之后，使用快速行进算法（FMM）移动到下一个最近的像素。FMM 保证了靠近已知（没有退化的）像素点的坏点先被修复，这与手工启发式操作比较类似。可以通过设置标签参数为 cv2.INPAINT\_TELEA 来使用此算法。  
第二个算法是根据 Bertalmio,Marcelo,Andrea\_L.Bertozzi, 和 Guillermo\_Sapiro在 2001 年发表的文章实现的。这个算法是基于流体动力学并使用了偏微分方程。基本原理是启发式的。它首先沿着正常区域的边界向退化区域的前进（因为边界是连续的，所以退化区域非边界与正常区域的边界应该也是连续的）。它通过匹配待修复区域中的梯度向量来延伸等光强线（isophotes，由灰度值相等的点练成的线）。为了实现这个目的，作者是用来流体动力学中的一些方法。  
  
完成这一步之后，通过填充颜色来使这个区域内的灰度值变化最小。可以通过设置标签参数为 cv2.INPAINT\_NS 来使用此算法。

## 实例：pic-repairdemo1.py

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

# 参考2

[Python+opencv：图像修复\_图像修复 csdn-CSDN博客](https://blog.csdn.net/hzblucky1314/article/details/130517174)

# Python+opencv：图像修复

**简介**：OpenCV 是一个开源的计算机视觉库，它包含了许多图像处理和计算机视觉算法。使用 OpenCV 进行图像修复主要依赖于传统的图像处理技术。

**OpenCV 图像修复方法及其原理：**

**1、去噪**：图像去噪是消除图像中的噪声，提高图像质量的过程。OpenCV 提供了多种去噪算法，如高斯滤波、中值滤波、双边滤波和非局部均值去噪等。这些算法通过平滑图像来消除噪声，同时尽量保持图像的边缘和细节。

**2、色彩平衡**：在老照片中，色彩可能会随着时间推移而逐渐失去平衡。OpenCV 可以通过直方图均衡化、对比度拉伸等技术来调整图像的色彩平衡。这些方法通过改变图像的亮度和颜色分布，使其看起来更自然和鲜艳。

**3、插值和超分辨率**：插值算法用于放大图像，增加其分辨率。OpenCV 提供了多种插值方法，如最近邻插值、双线性插值、双三次插值等。这些方法通过在像素之间插入新的像素值来扩展图像。在超分辨率任务中，OpenCV 可以使用诸如卷积稀疏编码（SCSR）等技术来提高图像的分辨率。

**4、图像修补**：当图像中有遮挡、划痕或缺陷时，可以使用图像修补技术来填充缺失的区域。OpenCV 提供了诸如 inpaint 函数等修补算法。这些算法通过利用周围像素的信息来估计缺失区域的像素值，从而恢复图像。

**5、锐化**：锐化是一种增强图像细节和边缘的技术。OpenCV 提供了多种锐化滤波器，如拉普拉斯滤波、高通滤波等。这些滤波器通过突出图像的高频信息来提高图像的清晰度。

这些传统图像处理方法在某些情况下可能有效，但它们往往无法应对复杂的图像损坏情况。对于这些任务，深度学习技术（如生成对抗网络，GAN）可能会提供更好的修复效果。

**历史攻略：**

[python：彩色照转黑白照](http://mp.weixin.qq.com/s?__biz=Mzg5NzEwMzkxNw==&mid=2247484454&idx=1&sn=bb69bcbd2b2fd152977d0054a8106686&chksm=c077a564f7002c72d1234b57dc60860f258e4f9a4a28523364cced2688143cce4acaffec2157&scene=21#wechat_redirect)

**安装依赖库：**

pip install opencv-python

**案例源码**：app.py

# -\*- coding: utf-8 -\*-

# time: 2023/4/26 18:58

# file: main.py

# 公众号: 玩转测试开发

import cv2

import numpy as np

def color\_restoration(image\_path):

# 读取图像

image = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

# 将图像从 BGR 转换为 LAB

lab\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2Lab)

# 将 LAB 图像拆分为单独的通道

l\_channel, a\_channel, b\_channel = cv2.split(lab\_image)

# 对每个通道应用自适应直方图均衡化

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

l\_channel = clahe.apply(l\_channel)

a\_channel = clahe.apply(a\_channel)

b\_channel = clahe.apply(b\_channel)

# 将处理后的通道重新组合为 LAB 图像

lab\_image = cv2.merge((l\_channel, a\_channel, b\_channel))

# 将图像从 LAB 转换回 BGR

result\_image = cv2.cvtColor(lab\_image, cv2.COLOR\_Lab2BGR)

return result\_image

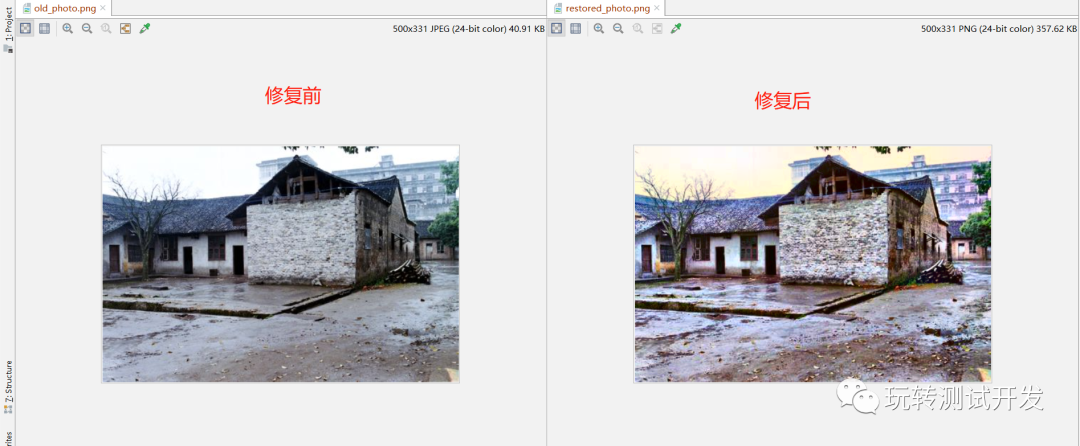
if \_\_name\_\_ == "\_\_main\_\_":

input\_image\_path = "old\_photo.png" # 替换为您的老照片路径

output\_image\_path = "restored\_photo.png" # 替换为恢复后的照片路径

restored\_image = color\_restoration(input\_image\_path)

cv2.imwrite(output\_image\_path, restored\_image)

**前后对比：**  


**注意事项**：微软有一个名为 “Bringing Old Photos Back to Life” 的开源项目。该项目使用深度学习技术（特别是生成对抗网络，GAN）进行老照片修复。它可以处理各种问题，如磨损、划痕、皱纹、褪色等，并能够自动修复老照片，使其看起来更清晰、更自然。

**项目GitHub仓库链接：**

https://github.com/microsoft/Bringing-Old-Photos-Back-to-Life

要运行此项目，您需要安装所需的依赖库（如 PyTorch、OpenCV 等），然后按照GitHub上的说明运行。这个项目非常适合用于修复老照片，可以帮助您将珍贵的回忆恢复到最佳状态。

# 参考3

**[OpenCV实战]34 使用OpenCV进行图像修复**

本文将描述一类称为图像修复的区域填充算法。想象一下找一张旧的家庭照片。你扫描它，它看起来很棒，除了一些划痕。当然，你可以在photoshop中加载照片并修复划痕。除此之外可以编写10行代码以使用[OpenCV](https://so.csdn.net/so/search?q=OpenCV&spm=1001.2101.3001.7020)中的修复算法来解决问题。

**1 什么是图像修复**

图像修复是计算机视觉中的一类算法，其目标是填充图像或视频内的区域。该区域使用二进制掩模进行标识，填充通常根据需要填充的区域边界信息来完成。图像修复的最常见应用是恢复旧的扫描照片。它还用于删除图像中的小的不需要的对象。

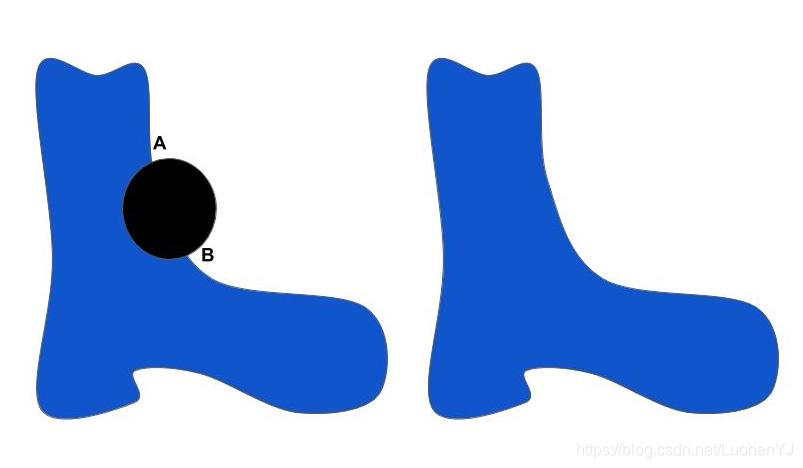
在本节中，我们将简要讨论在OpenCV中实现的两种修复算法。

**1.1 INPAINT\_NS : Navier-Stokes based Inpainting**

该方法于2001年发表在题为Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting的论文。论文见：

<http://www.math.ucla.edu/~bertozzi/papers/cvpr01.pdf>

有时我觉得计算机视觉领域是一个来自其他领域的移民领域，如电子工程，计算机科学，物理和数学。他们将自己的想法带到现场，以非常有趣和独特的方式解决同样的问题。电气工程师可以将图像看作2D信号，并应用信号处理理论来解决计算机视觉问题。另一方面，数学家可以将图像看作连通图并使用图论解决计算机视觉问题。因此，为流体动力学开发的理论也可以用于计算机视觉，这并不奇怪。在下图中，我们的目标是填充暗区并获得一个看起来像右边的图像。

​

wAAACH5BAEKAAAALAAAAAABAAEAAAICRAEAOw==

我们如何填补这个黑色区域？我们想要的一个约束是边缘进入点A应该连接边缘离开点B。我们可能想要的另一个约束是连接A和B的曲线右边的区域应该是白色，而左边的区域应该是蓝色的。

以上两个约束基本上要求：保留渐变（即边缘特征）和继续在平滑区域中传播颜色信息。

作者建立了一个偏微分方程（PDE）来更新具有上述约束的区域内的图像强度。

**1.2 INPAINT\_TELEA : Fast Marching Method based**

该方法基于论文An Image Inpainting Technique Based on the Fast Marching Method。论文作者Alexandru Telea。论文见：

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

该方法实现使用不同的技术解决了相同的约束。作者不使用图像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补绘。已知的邻域像素和梯度用于估计要修复的像素的颜色。

**1.3 方法比较与函数实现**

根据理论和论文，基于Navier-Stokes的修复应该更慢，并且倾向于产生比fast marching method的方法更模糊的结果。在实践中，我们没有发现这种情况。INPAINT\_NS在我们的测试中产生了更好的结果，速度也略高于INPAINT\_TELEA。

在OpenCV中，使用函数inpaint实现了修复算法。函数接口如下：

C++:

void inpaint( const Mat& src, const Mat& inpaintMask, Mat& dst, double inpaintRange, int flags );

wAAACH5BAEKAAAALAAAAAABAAEAAAICRAEAOw==

Python:

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

wAAACH5BAEKAAAALAAAAAABAAEAAAICRAEAOw==

Src：源图像

inpaintMask：二进制掩码，指示要修复的像素。

Dst：结果图像

inpaintRadius：表示修复的半径

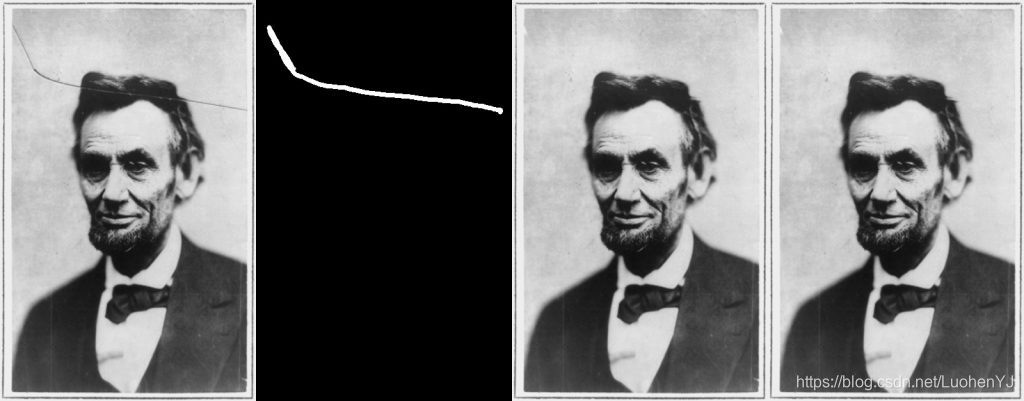
flags : 修复算法，主要有INPAINT\_NS (Navier-Stokes based method) or INPAINT\_TELEA (Fast marching based method)

**2 结果与代码**

**2.1 结果**

让我们来看看对林肯总统的历史形象进行修复的结果。这张照片背后有一段引人入胜的历史，我从维基百科借来的：

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，加德纳要求总统最后一个姿势。他把相机拉得更近，拍了一张林肯头部，肩膀和胸部的照片。神秘的玻璃板破裂。加德纳小心翼翼地将它带到了他的黑暗房间，并且能够制作一张印刷品，但在林肯的脸上有一个不祥的裂缝。在这个印刷品完全破碎并被弃用。但这种印刷品，即O-118，至今仍然存在。多年来，许多人将这一裂缝与10周后等待林肯的刺客子弹的象征性预言联系在一起。

​

wAAACH5BAEKAAAALAAAAAABAAEAAAICRAEAOw==

修复结果：上图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，第四个图像是INPAINT\_NS的结果。

让我们来看一个更复杂的例子。我们已经在一个花园的图像上草草写了很多，但是结果仍然非常引人注目。结果如下：

​

wAAACH5BAEKAAAALAAAAAABAAEAAAICRAEAOw==

上图中，左：带有潦草文字的原始图像。中：使用INPAINT\_TELEA方法修复，右：使用INPAINT\_NS。

**2.2 代码**

所有代码见：

<https://github.com/luohenyueji/OpenCV-Practical-Exercise>

两种算法修复效果都还不错，但是都需要事先准备修复模板的掩模mask,也就是inpaintMask 这个参数。例子里面用鼠标在图片上划线，划线的结果就是mask，而真正应用的时候需要事先设计好这个mask。例子程序中在划线确定mask后，不同按键有不同效果。按t选择INPAINT\_TELEA处理，按n选择INPAINT\_NS处理，按r查看原图，按ESC退出。

具体代码如下：

C++：

**#include "pch.h"**

**#include <opencv2/opencv.hpp>**

**#include <opencv2/photo.hpp>**

**#include <iostream>**

**using namespace cv;**

**using namespace std;**

**// Declare Mat objects for original image and mask for inpainting**

**Mat img, inpaintMask;**

**// Mat object for result output**

**Mat res;**

**Point prevPt(-1, -1);**

**// onMouse function for Mouse Handling**

**// Used to draw regions required to inpaint**

**// 调用鼠标事件**

**static void onMouse(int event, int x, int y, int flags, void\*)**

**{**

**if (event == EVENT\_LBUTTONUP || !(flags & EVENT\_FLAG\_LBUTTON))**

**prevPt = Point(-1, -1);**

**else if (event == EVENT\_LBUTTONDOWN)**

**prevPt = Point(x, y);**

**else if (event == EVENT\_MOUSEMOVE && (flags & EVENT\_FLAG\_LBUTTON))**

**{**

**Point pt(x, y);**

**if (prevPt.x < 0)**

**prevPt = pt;**

**line(inpaintMask, prevPt, pt, Scalar::all(255), 5, 8, 0);**

**line(img, prevPt, pt, Scalar::all(255), 5, 8, 0);**

**prevPt = pt;**

**imshow("image", img);**

**imshow("image: mask", inpaintMask);**

**}**

**}**

**int main()**

**{**

**string filename = "./image/flower-garden.jpg";**

**// Read image in color mode 读图**

**img = imread(filename);**

**Mat img\_mask;**

**// Return error if image not read properly**

**if (img.empty())**

**{**

**cout << "Failed to load image: " << filename << endl;**

**return 0;**

**}**

**namedWindow("image");**

**// Create a copy for the original image 复制原图像**

**img\_mask = img.clone();**

**// Initialize mask (black image)**

**inpaintMask = Mat::zeros(img\_mask.size(), CV\_8U);**

**// Show the original image**

**imshow("image", img);**

**//调用鼠标在图像上画圈**

**setMouseCallback("image", onMouse, NULL);**

**for (;;)**

**{**

**char c = (char)waitKey();**

**//按t选择INPAINT\_TELEA处理**

**if (c == 't')**

**{**

**// Use Algorithm proposed by Alexendra Telea**

**inpaint(img, inpaintMask, res, 3, INPAINT\_TELEA);**

**imshow("Inpaint Output using FMM", res);**

**}**

**//按n选择INPAINT\_NS处理**

**if (c == 'n')**

**{**

**// Use Algorithm proposed by Bertalmio et. al.**

**inpaint(img, inpaintMask, res, 3, INPAINT\_NS);**

**imshow("Inpaint Output using NS Technique", res);**

**}**

**//按r查看原图**

**if (c == 'r')**

**{**

**inpaintMask = Scalar::all(0);**

**img\_mask.copyTo(img);**

**imshow("image", inpaintMask);**

**}**

**//按ESC退出**

**if (c == 27)**

**{**

**break;**

**}**

**}**

**return 0;**

**}**

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreBlack.png

wAAACH5BAEKAAAALAAAAAABAAEAAAICRAEAOw==

Python：

**import numpy as np**

**import cv2 as cv**

**# OpenCV Utility Class for Mouse Handling**

**class Sketcher:**

**def \_\_init\_\_(self, windowname, dests, colors\_func):**

**self.prev\_pt = None**

**self.windowname = windowname**

**self.dests = dests**

**self.colors\_func = colors\_func**

**self.dirty = False**

**self.show()**

**cv.setMouseCallback(self.windowname, self.on\_mouse)**

**def show(self):**

**cv.imshow(self.windowname, self.dests[0])**

**cv.imshow(self.windowname + ": mask", self.dests[1])**

**# onMouse function for Mouse Handling**

**def on\_mouse(self, event, x, y, flags, param):**

**pt = (x, y)**

**if event == cv.EVENT\_LBUTTONDOWN:**

**self.prev\_pt = pt**

**elif event == cv.EVENT\_LBUTTONUP:**

**self.prev\_pt = None**

**if self.prev\_pt and flags & cv.EVENT\_FLAG\_LBUTTON:**

**for dst, color in zip(self.dests, self.colors\_func()):**

**cv.line(dst, self.prev\_pt, pt, color, 5)**

**self.dirty = True**

**self.prev\_pt = pt**

**self.show()**

**def main():**

**print("Usage: python inpaint <image\_path>")**

**print("Keys: ")**

**print("t - inpaint using FMM")**

**print("n - inpaint using NS technique")**

**print("r - reset the inpainting mask")**

**print("ESC - exit")**

**# Read image in color mode**

**img = cv.imread("./Lincoln.jpg")**

**# If image is not read properly, return error**

**if img is None:**

**return**

**# Create a copy of original image**

**img\_mask = img.copy()**

**# Create a black copy of original image**

**# Acts as a mask**

**inpaintMask = np.zeros(img.shape[:2], np.uint8)**

**# Create sketch using OpenCV Utility Class: Sketcher**

**sketch = Sketcher('image', [img\_mask, inpaintMask], lambda : ((255, 255, 255), 255))**

**while True:**

**ch = cv.waitKey()**

**if ch == 27:**

**break**

**if ch == ord('t'):**

**# Use Algorithm proposed by Alexendra Telea: Fast Marching Method (2004)**

**# Reference: https://pdfs.semanticscholar.org/622d/5f432e515da69f8f220fb92b17c8426d0427.pdf**

**res = cv.inpaint(src=img\_mask, inpaintMask=inpaintMask, inpaintRadius=3, flags=cv.INPAINT\_TELEA)**

**cv.imshow('Inpaint Output using FMM', res)**

**if ch == ord('n'):**

**# Use Algorithm proposed by Bertalmio, Marcelo, Andrea L. Bertozzi, and Guillermo Sapiro: Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting (2001)**

**res = cv.inpaint(src=img\_mask, inpaintMask=inpaintMask, inpaintRadius=3, flags=cv.INPAINT\_NS)**

**cv.imshow('Inpaint Output using NS Technique', res)**

**if ch == ord('r'):**

**img\_mask[:] = img**

**inpaintMask[:] = 0**

**sketch.show()**

**print('Completed')**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

**cv.destroyAllWindows()**

**https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreBlack.png**

**3 参考资料**

<https://www.learnopencv.com/image-inpainting-with-opencv-c-python/>



# 参考4

[opencv-python 图像修复 - 寒水浮云 - 博客园 (cnblogs.com)](https://www.cnblogs.com/libai123456/p/17618201.html)

# [opencv-python 图像修复](https://www.cnblogs.com/libai123456/p/17618201.html)

opencv中的图像修复很简单，用相邻像素替换这些坏标记，使其看起来跟周围颜色一样。

图像修复函数是：inpaint(src,inpaintmask,inpaintradius,flags)

参数说明：

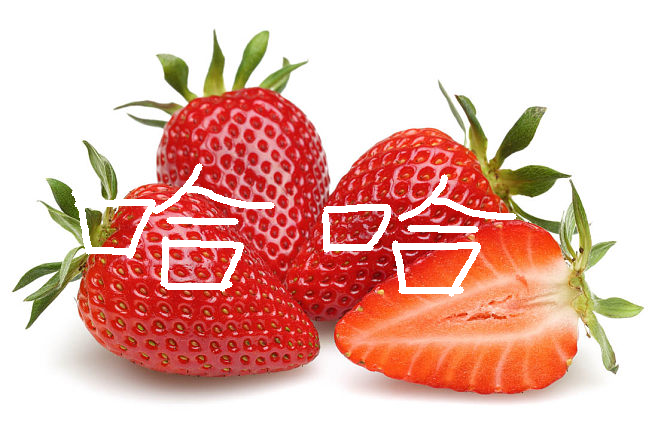
inpaintmask：图像掩码，单通道图像，大小和原图像一致，inpaintmask图像上除了需要修复的部分，其他地方全是0。

inpaintradius：每个点的圆心领域半径。

flags：修复图像的方法。INPAINT\_NS(流体力学法), INPAINT\_TELET(快速行进法)。

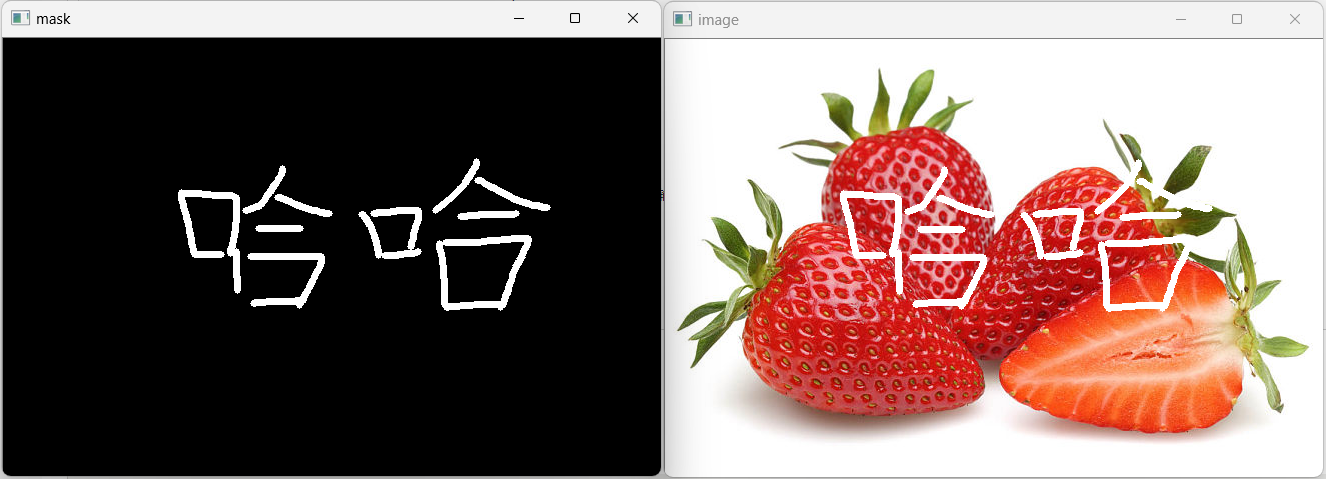
**图像修复案例如下：**

首先根据待修复图片制作mask。



利用鼠标事件，鼠标在待修复图像上面操作，鼠标左键按下拖动画白线，对应mask同时绘制白线（获取图片和mask）：

|  |
| --- |
| **import cv2**  **import numpy as np**    **start\_point=(0,0) #鼠标开始坐标**  **lb\_down = False #鼠标左键按下的标志，bool型**    **def mouse\_event(event,x,y,flags,param):**  **#如果全局变量是int或者str，那么如果想要在函数中对函数变量进行修改，则需要**  **#先在函数内，声明其为global，再进行修改，如果是list或者dict则可以直接修改**  **global start\_point,end\_point,lb\_down**  **if event == cv2.EVENT\_LBUTTONDOWN: #左键按下，更新鼠标坐标，启动按下标志**  **start\_point = (x,y)**  **lb\_down = True**    **elif event == cv2.EVENT\_MOUSEMOVE and lb\_down: #鼠标移动，绘制线**  **cv2.line(img,start\_point,(x,y),(255,255,255),thickness=5)**  **cv2.line(mask,start\_point,(x,y),(255,255,255),thickness=5)**  **start\_point = (x,y) #只要鼠标移动，就更新鼠标的坐标**    **elif event == cv2.EVENT\_LBUTTONUP: #左键释放**    **cv2.line(img,start\_point,(x,y),(255,255,255),thickness=5) #鼠标点击后直接释放鼠标的时候也会绘制一个点**  **cv2.line(mask,start\_point,(x,y),(255,255,255),thickness=5)**  **lb\_down = False**      **cv2.namedWindow('image') #新建窗口，用来进行鼠标操作**  **img = cv2.imread('./strawberry.jpg')**  **mask = np.zeros(img.shape,np.uint8) #创建一个黑色mask图像**    **cv2.setMouseCallback('image',mouse\_event) #设置鼠标回调**    **while True:**  **cv2.imshow('image',img)**  **cv2.imshow('mask',mask)**  **if cv2.waitKey(1)==ord('q'): #waitKey参数不能写0，写0就需要键盘输入才会继续**  **break**    **cv2.destroyAllWindows()** |



利用获取的mask进行图像修复：

|  |
| --- |
| **import cv2**  **import numpy as np**    **img = cv2.imread('./image.png')**  **mask = cv2.imread('mask.png',0)**    **result = cv2.inpaint(img,mask,5,cv2.INPAINT\_TELEA)**    **cv2.imshow('img',np.hstack((img,result)))**  **cv2.waitKey(0)**  **cv2.destroyAllWindows()** |

