東南大學 SOUTHEAST UNIVERSITY 人工智能学院

专业技能实训

授课教师：王洪松

邮箱：hongsongwang@seu.edu.cn

其它图像处理

颜色查找表

- 颜色查找表就是一种像素值映射的查找表。
- 调用LUT 函数可以获得最快的速度，这是因为OpenCV库可以通过英特尔线程架构启用多线程。

cv2.LUT(src, lut, dst=None)
src：原始图像。
lut：256维的向量，[0, 255]对应颜色值的查找表。如果输入src是多通道的，而查表是单通道的，则多通道使用的是同一个查找表。

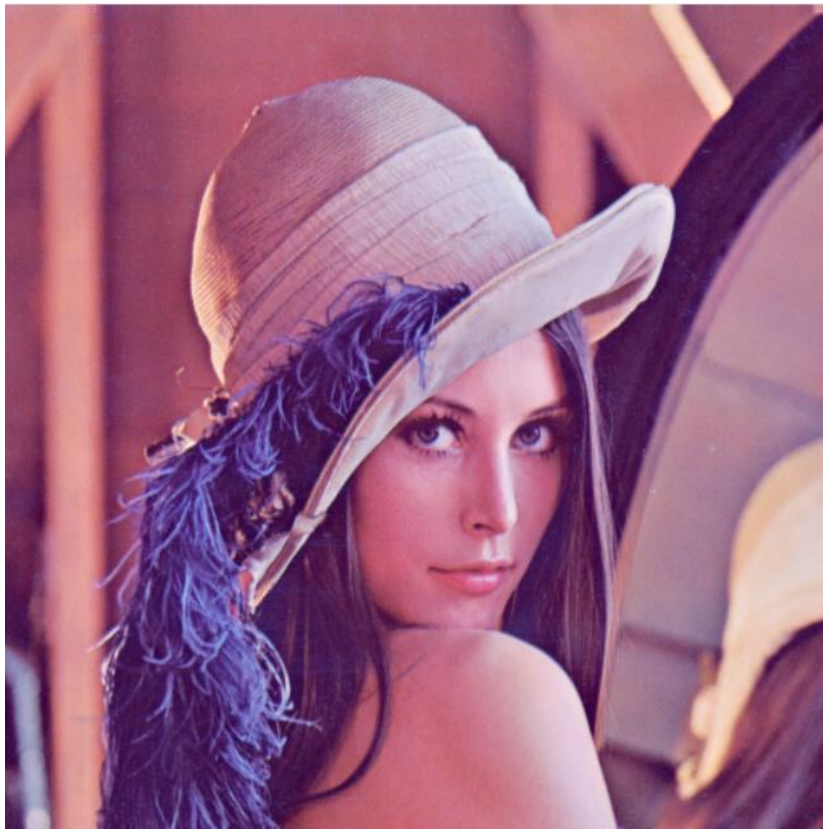
例如，增加图像亮度用颜色查找表实现：

```
table = np.array([i * brightness_factor for i in range
(0,256)]).clip(0,255).astype('uint8’ )
dst = cv2.LUT(img, table)
```

颜色查找表应用：风格变换

```
from scipy.interpolate import UnivariateSpline

def LookupTable(x, y):
  spline = UnivariateSpline(x, y)
  return spline(range(256))
#winter effect
def Winter(img):
    increaseLookupTable = LookupTable([0, 64, 128, 256],
[0, 80, 160, 256])
    decreaseLookupTable = LookupTable([0, 64, 128, 256],
[0, 50, 100, 256])
    blue_channel, green_channel,red_channel = cv2.split(img)
    red_channel = cv2.LUT(red_channel,
decreaseLookupTable).astype(np.uint8)
    blue_channel = cv2.LUT(blue_channel,
increaseLookupTable).astype(np.uint8)
    win= cv2.merge((blue_channel, green_channel, red_channel))
    return win
```



图像修补

- 使用区域邻域恢复图像中的选定区域。
- 用修补技术去除照片中小的噪音或划痕。

cv2.inpaint(src, mask, radius, flags)
mask：掩码图像，非零像素表示需要修复的区域。
radius：算法考虑的每个修复点的圆形邻域的半径。
flags：可以是cv2.INPAINT_TELEA或cv2.INPAINT_NS。

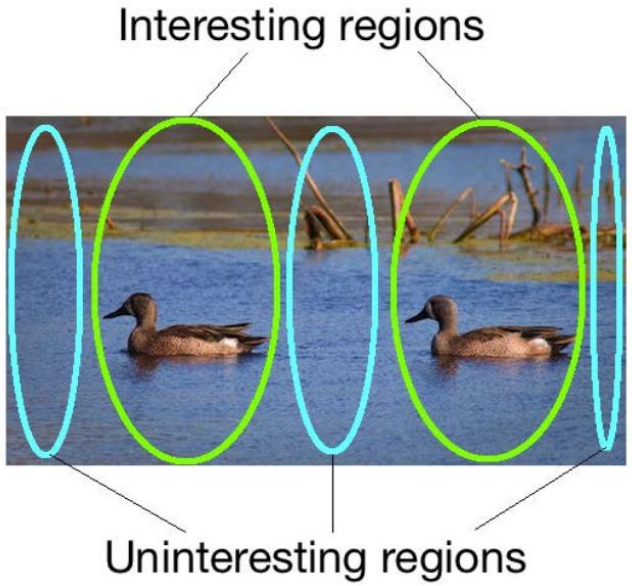
掩码图像可以人工绘制或者基于修补区域的特点编程生成。

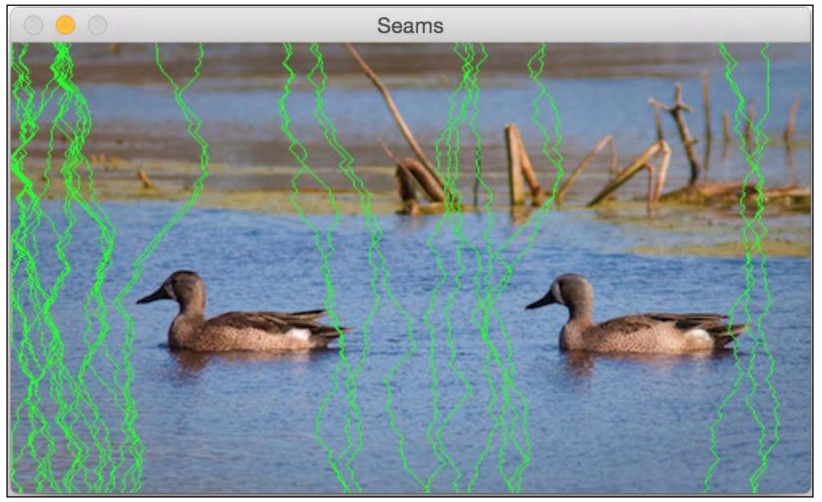
```
img = cv2.imread('messi_2.jpg’ )
mask = cv2.imread('mask2.png',0)
dst  = cv2.inpaint(img,mask,3,
cv2.INPAINT_TELEA)
```

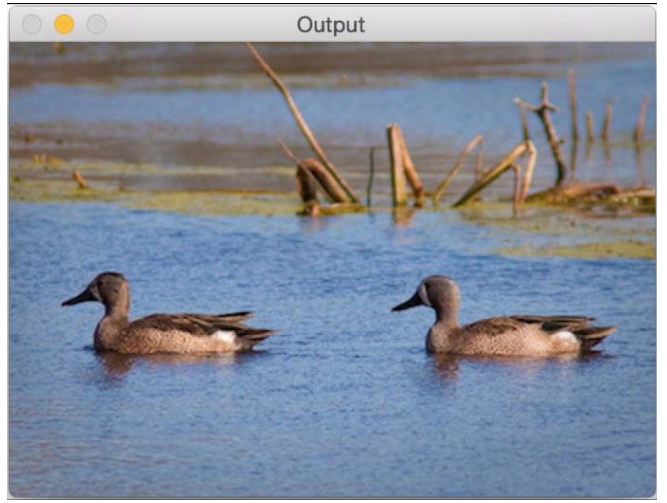
原图和掩码　修复结果

Seam Carving

- 很自然的扩张图像，并且还保持了感兴趣的目标主体的形状比例。
- 使某个对象从图像中消失。





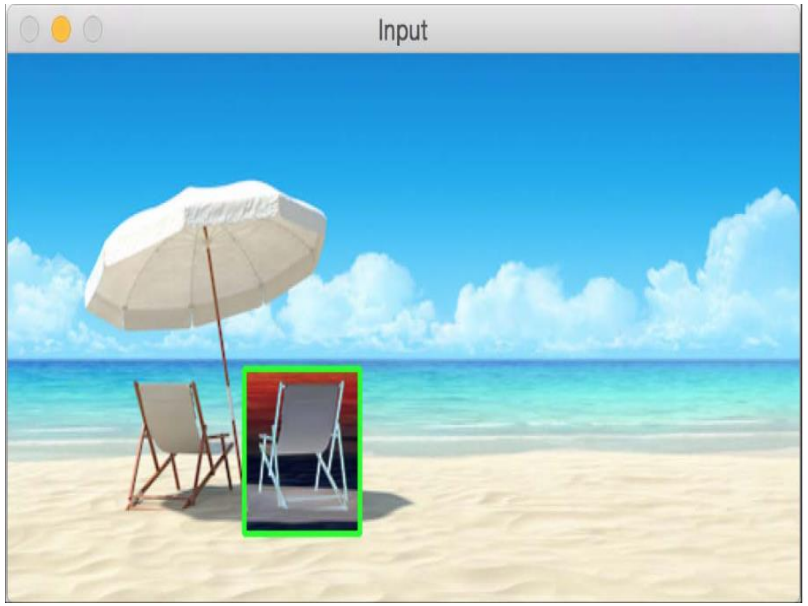


Seam Carving算法是通过移除或者插入累计能量最小的像素线的方式来实现图像缩放。每次在垂直或水平方向上利用动态规划法找到一条能量最小的像素线，通过删除该像素线实现图像缩小，反之插入该像素线实现图像放大。

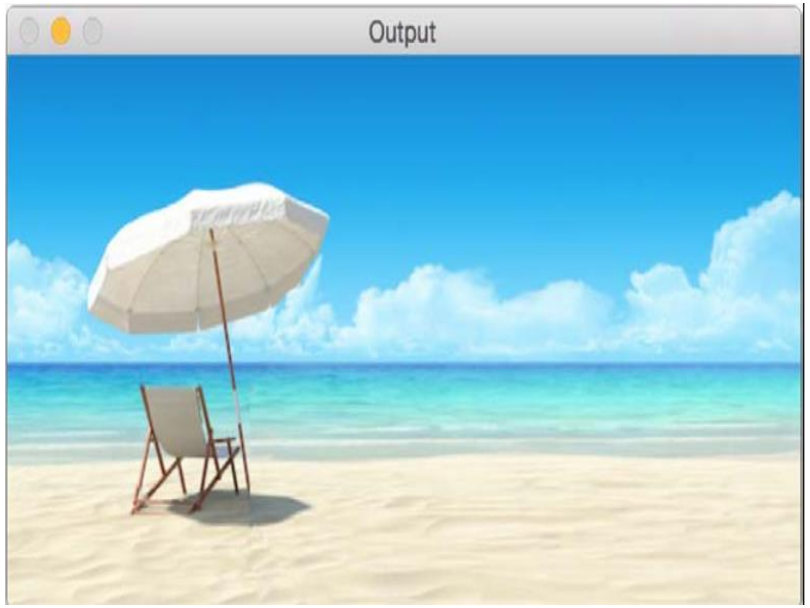
Seam Carving

- 删除图像中的某个对象

原图



选择删除的区域



删除目标后的图像

使用Seam Carving去除物体。首先选择感兴趣的区域，让所有的接缝都穿过这个区域。在我们移除与该区域相关的所有接缝之后，继续添加接缝，直到将图像扩展到其原始宽度。

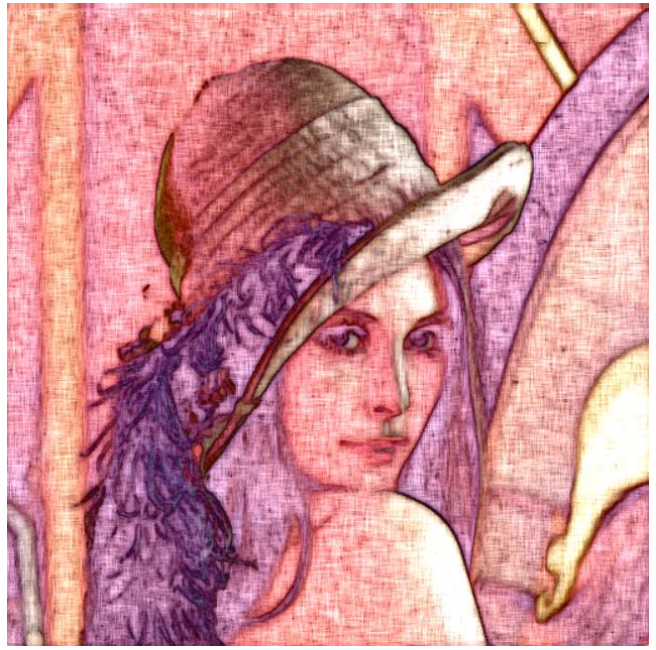
铅笔素描

- 生成类似铅笔的非真实的线条绘制的图像。

dst1_gray, dst1_color = cv2.pencilSketch(img, sigma_s = 50, sigma_r = 0.15, shade_factor = 0.04)

sigma_s：邻近Pixel的数量，范围在0到200之间。
sigma_r：颜色平衡度，数值越大，同颜色的区域就会越大。范围在0到1之间。
shade_factor：图像亮度，值越高图像越亮，范围在0到0.1之间。

dst1_gray



dst1_color

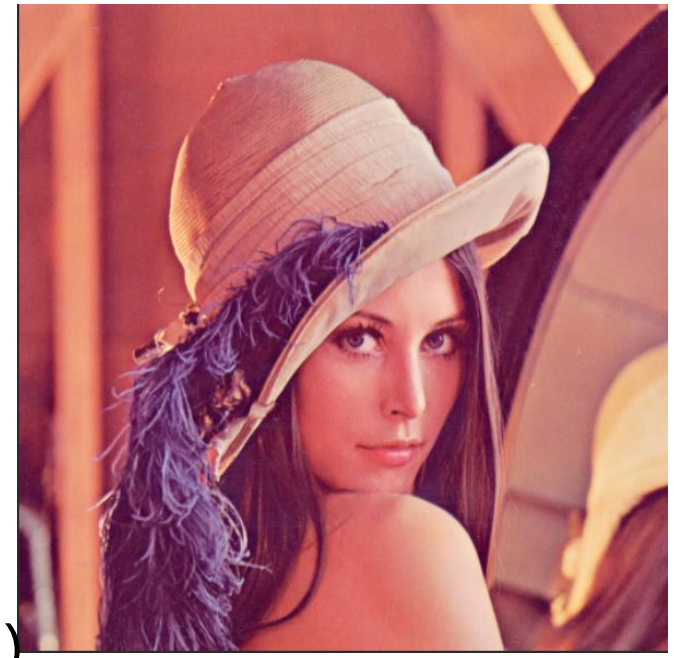
```
img = cv2.imread('LenaRGB.bmp', -1)
dst1_gray, dst1_color = cv2.pencilSketch(img, sigma_s = 50, sigma_r = 0.15, shade_factor = 0.04)
```

图像风格化

- 制作具有各种各样效果的数字图像，而不是关注照片真实性。
- 抽出对比度低的特征，保留对比度高的特征。

```
dst = cv2.stylization(img, sigma_s=200, sigma_r=0.6)
```

sigma_s：邻近Pixel的数量，范围在0到200之间。
sigma_r：颜色平衡度，数值越大，同颜色的区域就会越大。范围在0到1之间。



原图

风格化图像

```
img = cv2.imread('LenaRGB.bmp', -1)
dst = cv2.stylization(img, sigma_s = 50,
sigma_r = 0.15)
```

图像细节增强

- 对图像滤波以增强图像的细节。

dst = cv2.detailEnhance(img, sigma_s=100, sigma_r=0.1)

sigma_s：邻近Pixel的数量，范围在0到200之间。
sigma_r：颜色平衡度，数值越大，同颜色的区域就会越大。范围在0到1之间。

原图

细节增强图像

```
img = cv2.imread('LenaRGB.bmp', -1)
dst = cv2.stylization(img, sigma_s = 50,
sigma_r = 0.15)
```

边缘保持图像平滑

- 保留图片边缘特征做的平滑化处理，可以实现磨皮、美颜的效果。

```
dst = cv2.edgePreservingFilter(img, flags=1, sigma_s = 50, sigma_r = 0.15)
```

flags：滤波参数, 1为递归双边滤波（速度3倍快），2为归一化卷积(对边缘产生锐化)。
sigma_s：邻近Pixel的数量，范围在0到200之间。
sigma_r：颜色平衡度，数值越大，同颜色的区域就会越大。范围在0到1之间。

原图

边缘保持图像平滑

```
img = cv2.imread('LenaRGB.bmp', -1)
dst = cv2.edgePreservingFilter(img,
flags=1, sigma_s = 50, sigma_r = 0.15)
```

专业技能实训：练习题

- 去除图像中的文字
  - 在输入图像的背景区域添加文字。
  - 用inpaint或其它方法去除图像中文字，恢复文字区域的背景。
- 去除图像中的物体
  - 用numpy向量运算改写Seam Carving的Python代码，减少for使用。
  - 用Seam Carving删除图像中的选定物体。
- 放大或缩小图像中的物体
  - 输入图像，用鼠标选择前景目标并分割该目标。
  - 用滚动条控制该目标的大小，推动滚动条，该目标相应的放大或缩小。
  - 目标缩小时，空出的白色背景用Seam Carving或其它有效方法处理。

专业技能实训：练习题

- 用颜色查找表实现图像风格转化
  - 用一个滚动条控制不同的图像风格类型。
  - 用cv2.LUT实现对应的图像风格转化，输出的图像尽量不失真。

- 沿着曲线插入文字
  - 假设原图中沿着某个目标的边缘是个曲线。
  - 用鼠标选择该目标边缘的几个点，用曲线拟合（例如，UnivariateSpline）逼近该边缘。
  - 沿着该逼近的边缘插入文字，文字的方向大致和插入点的切线平行。