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

颜色空间

HSV颜色空间

- 相比于RGB空间，HSV（Hue-Saturation-Value，色调饱和度）颜色空间更符合人们对颜色的描述。
  - H：色相，色度，色彩，也就是我们平时说的颜色。在HSV模型中，用度数描述，其中红色对于0度，绿色对于120度，蓝色对应240度。
  - S：饱和度，色彩的深浅度（0-100%）。
  - V：色调，纯度，色彩的亮度(0-100%)。

当S=1, V=1时，H所代表的任何颜色被称为纯色。
当S=0时，颜色最浅，最浅被描述为灰色（灰色也有亮度，黑色和白色也属于灰色），灰色的亮度由V决定，此时H无意义。
当V=0时，颜色最暗，最暗被描述为黑色，因此此时H(无论什么颜色最暗都为黑色)和S(无论什么深浅的颜色最暗都为黑色)均无意义。

L*a*b*颜色空间

- 从视觉感知均匀的角度，人所感知到的两种颜色的区别程度，应该与这两种颜色在色彩空间中的距离成正比。
- L*a*b*颜色空间是均匀色彩空间模型，它是面向视觉感知的颜色模型。
  - L分量用于表示像素的亮度，取值范围是[0,100]，表示从纯黑到纯白；
  - a分量表示从红色到绿色的范围，取值范围是[-127,127];
  - b分量表示从黄色到蓝色的范围，取值范围是[-127,127]。
  - 在从RGB色彩空间转换到L*a*b*色彩空间之前，需要先将RGB色彩空间的值转换到[0, 1]之间，然后再进行处理。

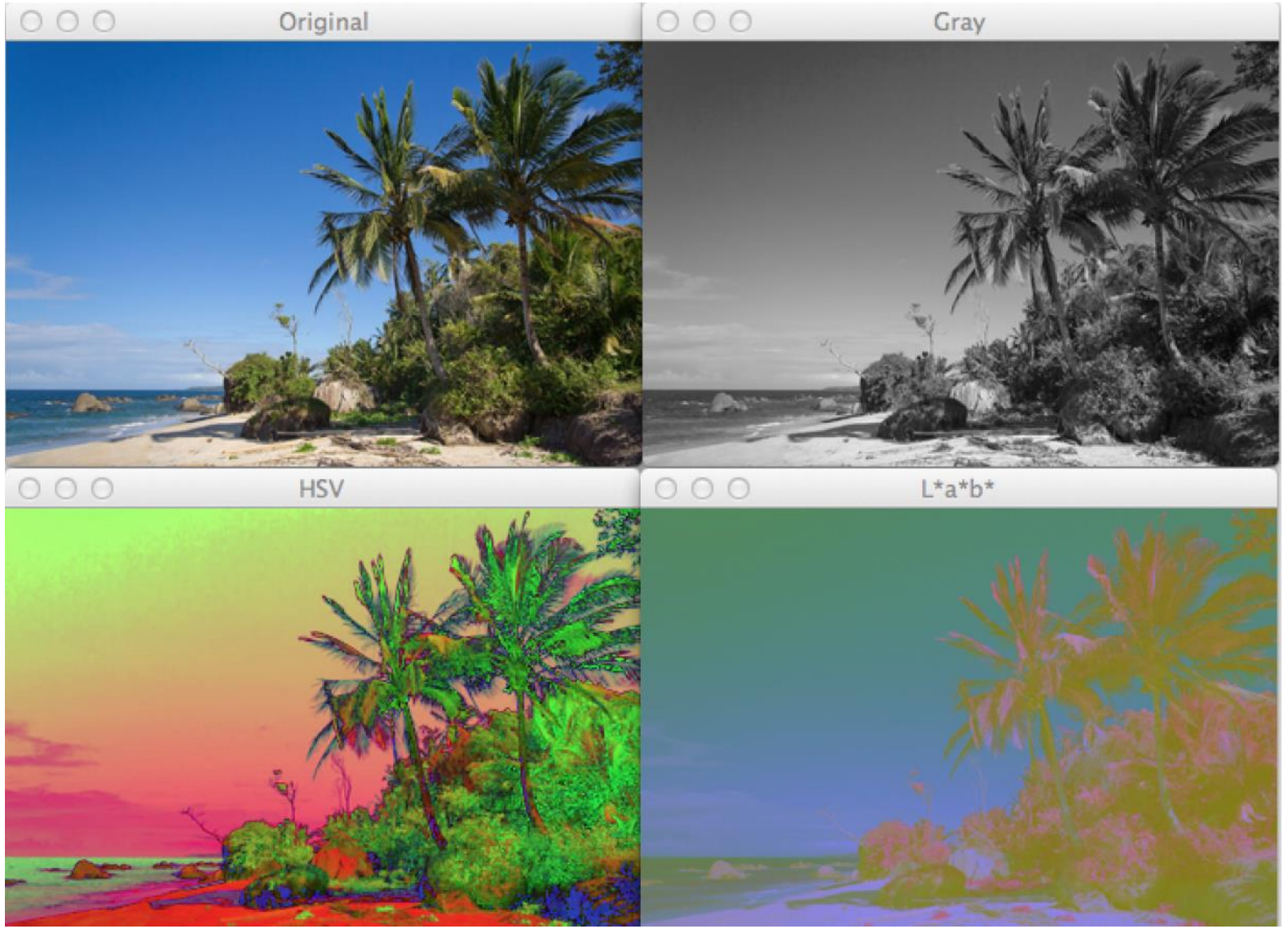
其它颜色空间

- HLS 空间包含的三要素是色调 H （ Hue）、光亮度/明度 L （ Lightness）、饱和度 S （ Saturation）。与 HSV 空间相比，HLS 空间用“光亮度/明度 L（ lightness）” 替换了“亮度 （ Value）”。
- YCrCb 空间，Y 代表光源的亮度， 色度信息保存在 Cr 和 Cb 中，Cr 表示 红色分量信息，Cb 表示蓝色分量信息。
- L*u*v*空间与设备无关，适用于显示器显示和根据加色原理进行组合的场合。该模型中比较强调对红色的表示，对红色的变化比较敏感，但对蓝色的变化不太敏感。
- Bayer 颜色空间，被广泛地应用在 CCD 和 CMOS 相机中。

颜色空间变换

cv2.cvtColor()函数
颜色空间的变换

```
image = cv2.imread(path)
cv2.imshow("Original", image)
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Gray", gray)
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
cv2.imshow("HSV", hsv)
lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
cv2.imshow("L*a*b*", lab)
cv2.waitKey(0)
```

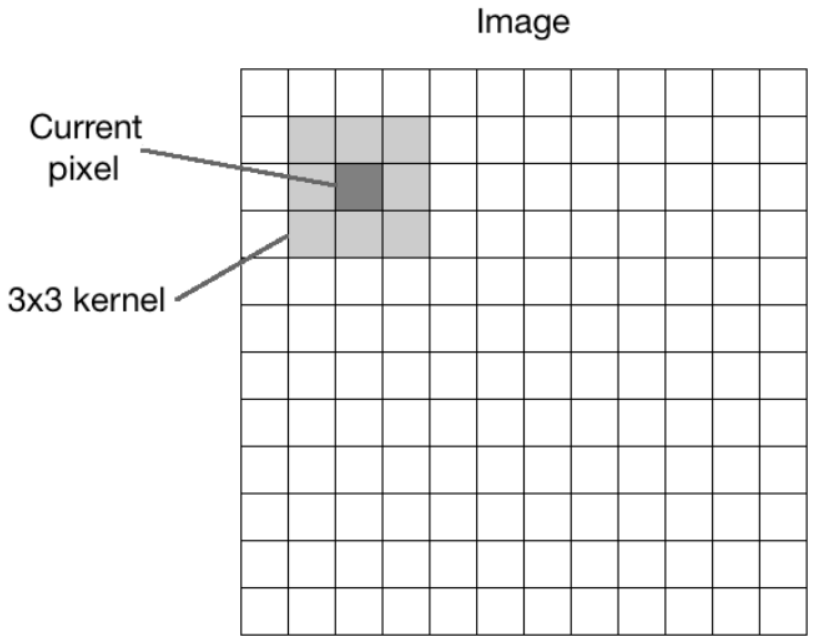


图像滤波和增强

2D卷积

- 卷积是图像处理中的一项基本操作。
- 2D卷积是利用卷积核在图像上滑动，将图像点上的像素值与对应的卷积核上的数值相乘，然后将所有相乘后的值相加作为卷积核中间像素对应的图像上像素值。
- 卷积核的大小通常比输入图像小得多。

图像模糊、图像锐化、图像平滑、边缘提取、滤波去噪等很多处理都是基于2D图像卷积。



图像2D卷积

图像平滑

- 模糊是指对邻域内的像素值进行平均。也称为低通滤波器，即允许低频并阻止高频的滤波器。
- 对于图像，频率是指像素值的变化率。尖锐的边缘是高频内容，因为该区域的像素值变化很快。相反，平原地区是低频内容。低通滤波器试图平滑图像边缘。

一种简单的构建低通滤波器的方法是对像素附近的值做平均。可以根据需要对图像进行平滑的程度来选择卷积核的大小。更大的卷积核将在更大的面积上取平均值，往往会增加平滑效果。

$$L = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

3*3的低通滤波卷积核

图像平滑

```
import cv2
import numpy as np
img = cv2.imread('input.jpg')
rows, cols = img.shape[:2]
kernel_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])
kernel_3x3 = np.ones((3,3), np.float32) / 9.0
kernel_5x5 = np.ones((5,5), np.float32) / 25.0
cv2.imshow('Original', img)
output = cv2.filter2D(img, -1, kernel_identity)
cv2.imshow('Identity filter', output)
output = cv2.filter2D(img, -1, kernel_3x3)
cv2.imshow('3x3 filter', output)
output = cv2.filter2D(img, -1, kernel_5x5)
cv2.imshow('5x5 filter', output)
cv2.waitKey(0)
```

随着卷积核大小的增加，图像变得越来越模糊。

也可以使用blur函数实现

output = cv2.blur(img, (3,3))

运动模糊

- 运动模糊是对特定方向上的像素值做平均（定向低通滤波器）

```
import cv2
import numpy as np
img = cv2.imread('input.jpg')
cv2.imshow('Original', img)
size = 15
# generating the kernel
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size-1)/2), :] =
np.ones(size)
kernel_motion_blur = kernel_motion_blur / size
# applying the kernel to the input image
output = cv2.filter2D(img, -1, kernel_motion_blur)
cv2.imshow('Motion Blur', output)
cv2.waitKey(0)
```

例如，3*3水平运动模糊卷积核如下，它将在水平方向上模糊图像。可以选择任何运动模糊方向。

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

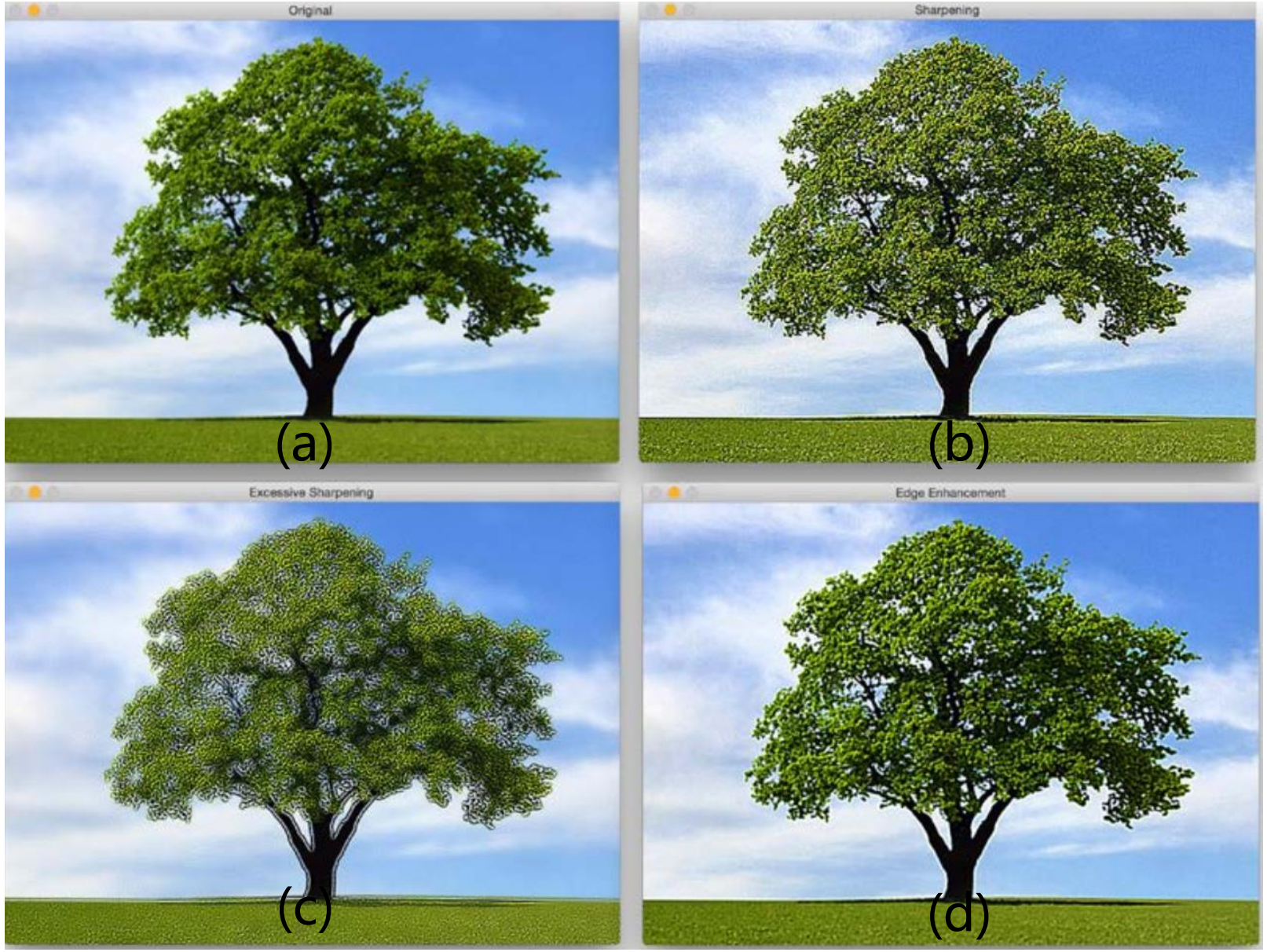
运动模糊效果

图像锐化

- 图像锐化是增强图像的边缘。锐化层度取决于使用不同的卷积核类型，不同的卷积核会导致不同的锐化结果。

```
img = cv2.imread('input.jpg')
cv2.imshow('Original', img)
# generating the kernels
kernel_sharpen_1 = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
kernel_sharpen_2 = np.array([[1,1,1], [1,-7,1], [1,1,1]])
kernel_sharpen_3 = np.array([[-1,-1,-1,-1,-1], [-1,2,2,2,-1],
[-1,2,8,2,-1], [-1,2,2,2,-1], [-1,-1,-1,-1,-1]]) / 8.0
# applying different kernels to the input image
output_1 = cv2.filter2D(img, -1, kernel_sharpen_1)
output_2 = cv2.filter2D(img, -1, kernel_sharpen_2)
output_3 = cv2.filter2D(img, -1, kernel_sharpen_3)
cv2.imshow('Sharpening', output_1)
cv2.imshow('Excessive Sharpening', output_2)
cv2.imshow('Edge Enhancement', output_3)
cv2.waitKey(0)
```

图像锐化



图（b）(c)中，虽然输出图像看起来增强了，但图像看起来不自然。

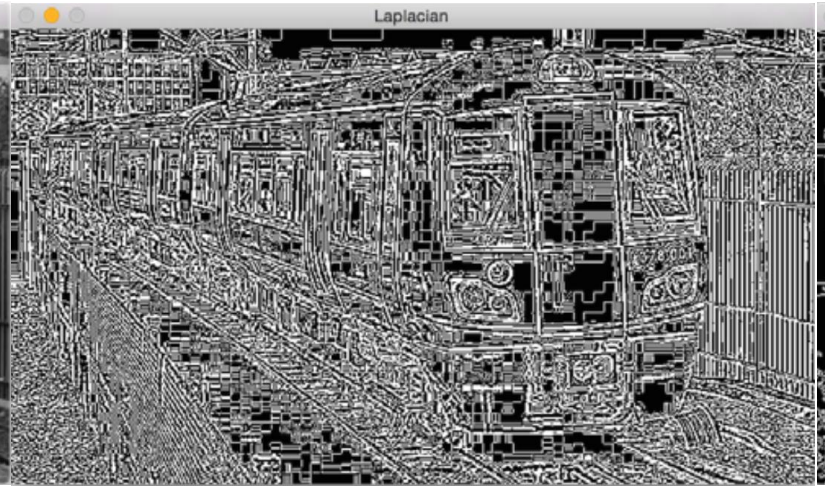
(d)使用近一个似高斯核来构建此滤波器，在增强边缘的同时平滑图像，从而使图像看起来更自然。

边缘检测

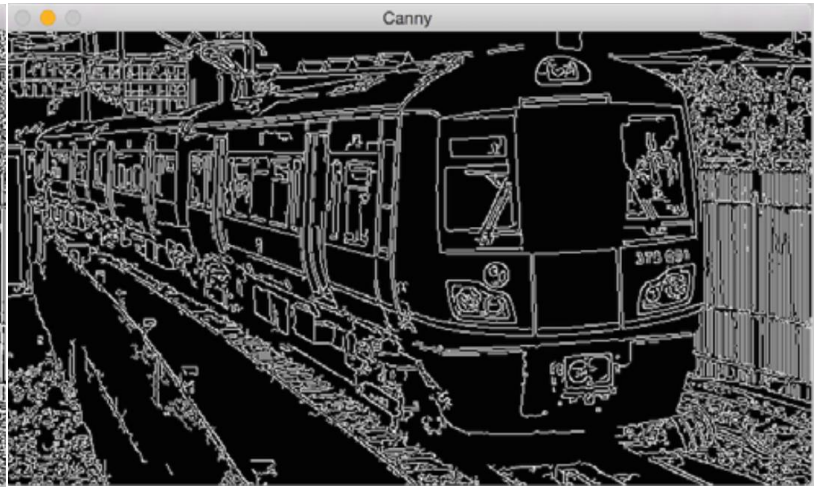
- Laplacian边缘检测，对噪声比较敏感。
- Canny边缘检测（多级边缘检测算法），不容易受噪声干扰。

laplacian = cv2.Laplacian(img, cv2.CV_64F)
canny = cv2.Canny(img, 50, 240) 后两个参数为处理过程中的可调节的阈值

原图



Laplacian边缘检测

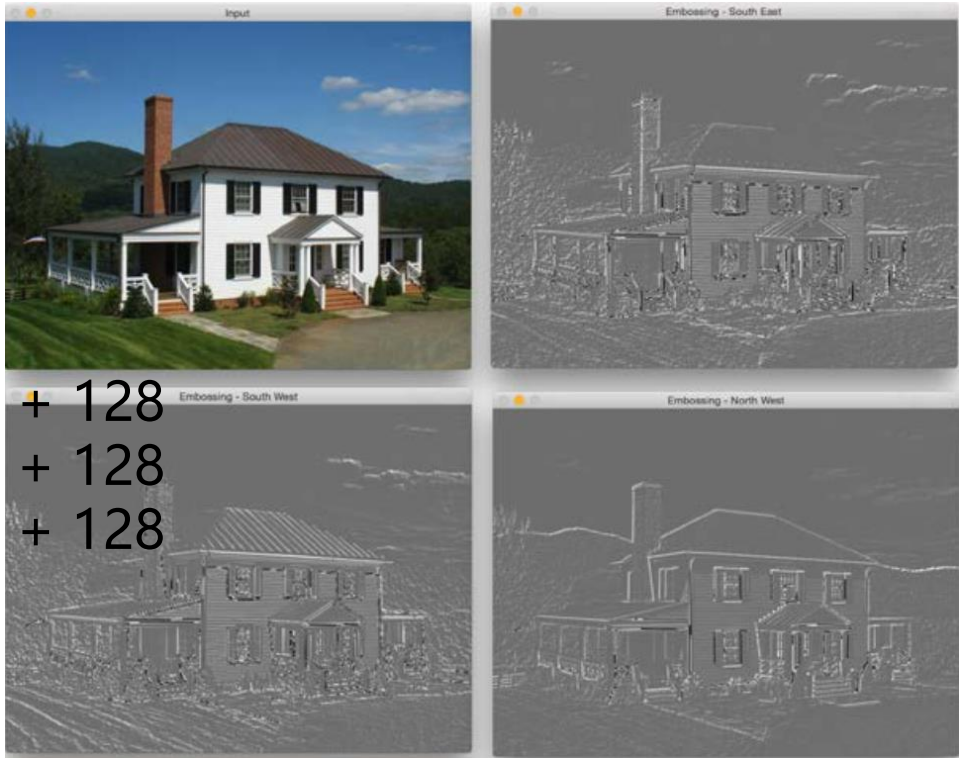


Canny边缘检测

图像浮雕（Image Embossing）

- 浮雕就是把所要呈现的图像突起于石头表面，根据凹凸的程度不同从而形成三维的立体感。图像浮雕是通过相邻元素相减的方法得到轮廓与边缘的差，从而获得凹凸的立体感觉。

```
# generating the kernels
kernel_emboss_1 = np.array([[0,-1,-1],[1,0,-1],[1,1,0]])
kernel_emboss_2 = np.array([[-1,-1,0],[-1,0,1],[0,1,1]])
kernel_emboss_3 = np.array([[1,0,0],[0,0,0],[0,0,-1]])
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# applying the kernels to image and adding the offset
output_1 = cv2.filter2D(gray_img, -1, kernel_emboss_1) + 128
output_2 = cv2.filter2D(gray_img, -1, kernel_emboss_2) + 128
output_3 = cv2.filter2D(gray_img, -1, kernel_emboss_3) + 128
cv2.imshow('Input', img_emboss_input)
cv2.imshow('Embossing - South West', output_1)
cv2.imshow('Embossing - South East', output_2)
cv2.imshow('Embossing - North West', output_3)
cv2.waitKey(0)
```

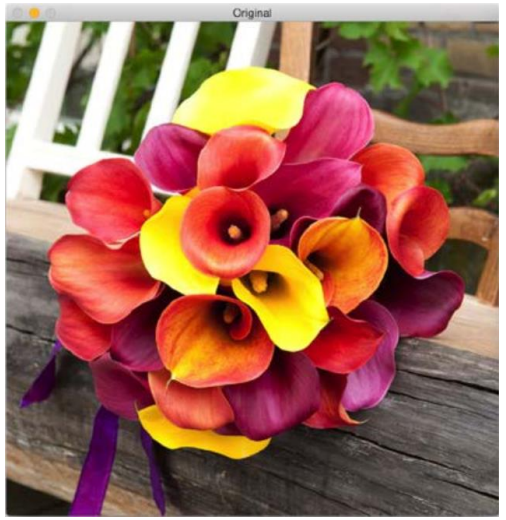


渐晕（vignette filter）

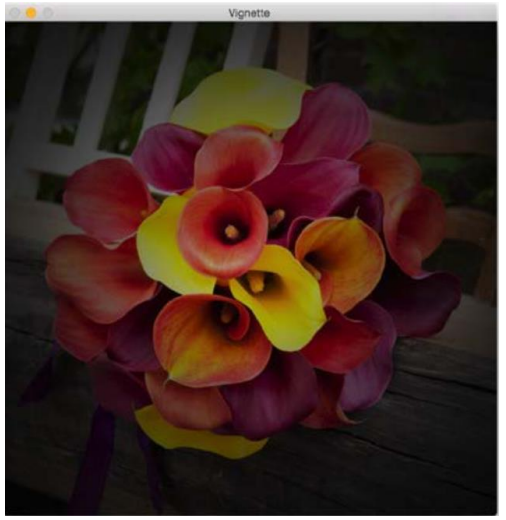
- Vignette渐晕将亮度集中在图像的特定部分，而其他部分看起来会褪色。使用高斯核过滤掉图像中的每个通道。

```
rows, cols = img.shape[:2]
# generating vignette mask using Gaussian kernels
kernel_x = cv2.getGaussianKernel(cols, 200)
kernel_y = cv2.getGaussianKernel(rows, 200)
kernel = kernel_y * kernel_x.T
mask = 255 * kernel / np.linalg.norm(kernel)
output = np.copy(img)
# applying the mask to each channel
for i in range(3):
    output[:,:,i] = output[:,:,i] * mask
cv2.imshow('Original', img)
cv2.imshow('Vignette', output)
cv2.waitKey(0)
```

cv2.getGaussianKernel(kernel_size, sigma)
生成一维高斯核。
乘以转置得到二维高斯核。
标准化二维高斯核并放大。



原图



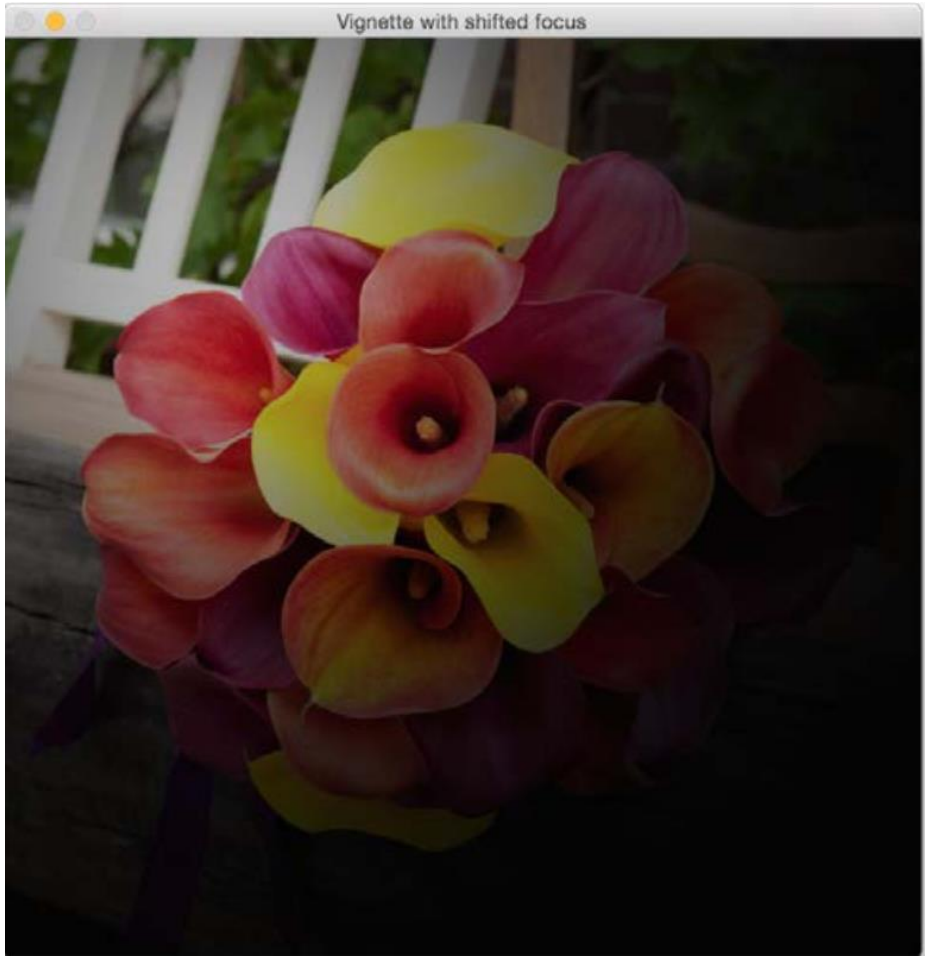
滤镜图像

渐晕（vignette filter）

- 实现相同的渐晕效果，但关注图像中的不同区域而不是中心。

```
# generating shifted vignette mask
kernel_x = cv2.getGaussianKernel(
int(1.5*cols),200)
kernel_y = cv2.getGaussianKernel(
int(1.5*rows),200)
kernel = kernel_y * kernel_x.T
mask = 255 * kernel / np.linalg.norm(kernel)
mask = mask[int(0.5*rows):, int(0.5*cols):]
```

创建一个更大的高斯核，并确保峰值
与感兴趣的区域一致。



图像对比度增强

- 在弱光条件下拍摄图像时，图像都会变暗，像素值往往集中在0附近，图像中的许多细节人眼看不清楚，这时候需要调整对比度。
- 图像对比度增强是调整像素值使其分布在0到255之间。

对于灰度图像的图像对比度增强

```
import cv2
import numpy as np
img = cv2.imread('input.jpg', 0)
# equalize the histogram of the input image
histeq = cv2.equalizeHist(img)
cv2.imshow('Input', img)
cv2.imshow('Histogram equalized', histeq)
cv2.waitKey(0)
```

原图

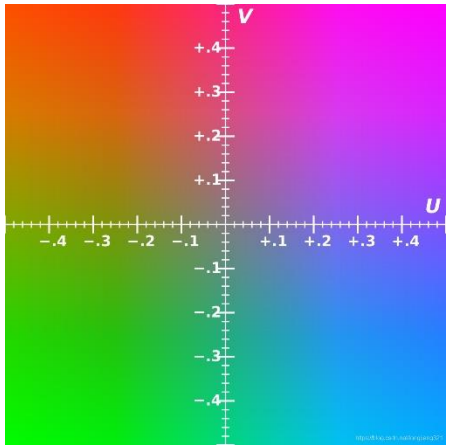
增强后

图像对比度增强

- 彩色图像对比度增强。不能将RGB图像三个通道分离出来，单独均衡直方图，然后将它们组合起来形成输出图像。
- 为了处理彩色图像的直方图均衡，需要将其转换为将强度与颜色信息分离的颜色空间，比如YUV空间。一旦将其转换为YUV，只需要均衡Y通道，并将其与其他两个通道组合即可获得输出图像。

```
img = cv2.imread('input.jpg')
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
# equalize the histogram of the Y channel
img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])
# convert the YUV image back to RGB format
img_output = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)
cv2.imshow('Color input image', img)
cv2.imshow('Histogram equalized', img_output)
cv2.waitKey(0)
```

在YUV颜色空间中，
Y就表示了灰度信息。
UV组成的颜色平面如下



双边滤波

- 高斯滤波是一种低通滤波，在滤除图像中噪声信号的同时，也会对图像中的边缘信息进行平滑，表现出来的结果是图像变得模糊。
- 双边滤波是一种非线性的滤波方法，在去噪的同时可以保留边缘信息。
- 双边滤波在高斯滤波的基础上加入了像素值权重项，既要考虑距离因素，也要考虑像素值差异的影响，像素值越相近，权重越大。

$$W_q = \sum_{p \in S} G_s(p)\, G_r(p) = \sum_{p \in S} exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}\right) exp\left(-\frac{\|I_p - I_q\|^2}{2\sigma_r^2}\right)$$

dst = cv2.bilateralFilter（src,d,sigmaColor,sigmaSpace,borderType）

d：过滤时周围每个像素领域的直径 。
sigmaColor：颜色空间滤波器的sigma值，该值越大，就表明像素邻域内有更宽广的颜色会被混合到一起。
sigmaSpace：坐标空间中滤波器的sigma值，坐标空间的标注方差。该数值越大，意味着越远的像素会相互影响，从而使更大的区域足够相似的颜色获取相同的颜色。

样例：将图像卡通化

```
def cartoonize_image(img, ds_factor=4, sketch_mode=False):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_gray = cv2.medianBlur(img_gray, 7)
    edges = cv2.Laplacian(img_gray, cv2.CV_8U, ksize=5)
    ret, mask = cv2.threshold(edges, 100, 255, cv2.THRESH_BINARY_INV)
    # 'mask' is the sketch of the image
    if sketch_mode:
        return cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    # resize the image to a smaller size for faster computation
    img_small = cv2.resize(img, None, fx=1.0/ds_factor, fy=1.0/ds_factor, interpolation=cv2.INTER_AREA)
    num_repetitions = 10
    sigma_color = 5
    sigma_space = 7
    size = 5
    # apply bilateral filter the image multiple times
    for i in range(num_repetitions):
        img_small = cv2.bilateralFilter(img_small, size, sigma_color, sigma_space)

    img_output = cv2.resize(img_small, None, fx=ds_factor, fy=ds_factor, interpolation=cv2.INTER_LINEAR)
    dst = np.zeros(img_gray.shape)
    # add the thick boundary lines to the image using 'AND' operator
    dst = cv2.bitwise_and(img_output, img_output, mask=mask)
    return dst
```

样例：将图像卡通化

中值滤波 cv2.medianBlur(src, ksize），取当前像素点及其周围临近像素点的像素值的均值。
双边滤波 cv2.bilateralFilter（src,d,sigmaColor,sigmaSpace,borderType）

原图

草图

卡通彩色图像

专业技能实训：练习题

- 实现图像浮雕
  - 用2个滚动条，1个控制滤波器大小，1个控制浮雕方向。
  - 分别拖动滚动条，实现图像浮雕。
  - 在滚动条下方，对比显示原图和处理后的图像。
- 用多种方法生成草图
  - 设置一个type的滚动条，表示多种不同的生成草图方法（e.g., Canny边缘检测，Laplacian边缘检测等）。
  - 拖动滚动条，生成对应的草图。
  - 在滚动条下方，对比显示原图和草图。

专业技能实训：练习题

- 调节图像的亮度、对比度、饱和度、色温等。
  - 分别采用滚动条控制图像的亮度、对比度、饱和度和色温。
  - 在滚动条下方，对比显示原图和处理后的图像。
- 选择图像兴趣中心实现渐晕（vignette filter）。
  - 输入图像，采用滚动条控制渐晕高斯核大小。
  - 拖动鼠标选择图像ROI区域，以ROI的中心为感兴趣的区域，使用高斯核实现渐晕（vignette filter）效果。
  - 在滚动条下方，对比显示原图和处理后的图像。