Advanced Image Processing Using Open CV

1. Blending Two Images

 Blend two images of the same size using weighted addition.

Mathematical definition [edit]

Formally, the weighted mean of a non-empty finite tuple of data $(x_1, x_2, \ldots, x_n)$, with corresponding non-negative weights $(w_1, w_2, \ldots, w_n)$ is

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i},$$

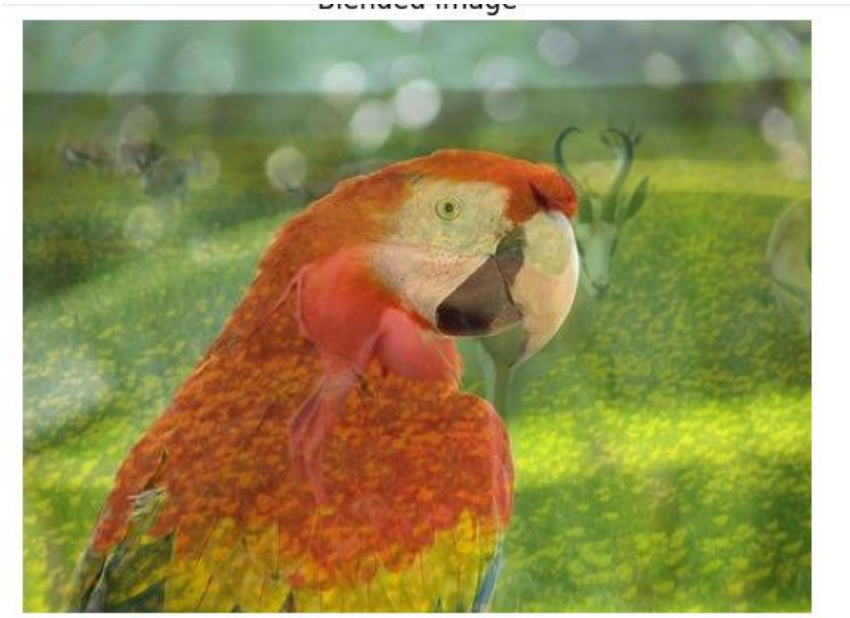
which expands to:

$$\bar{x} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}{w_1 + w_2 + \cdots + w_n}.$$

Therefore, data elements with a high weight contribute more to the weighted mean than do elements with a low weight. The weights may not be negative in order for the equation to work[a]. Some may be zero, but not all of them (since division by zero is not allowed).

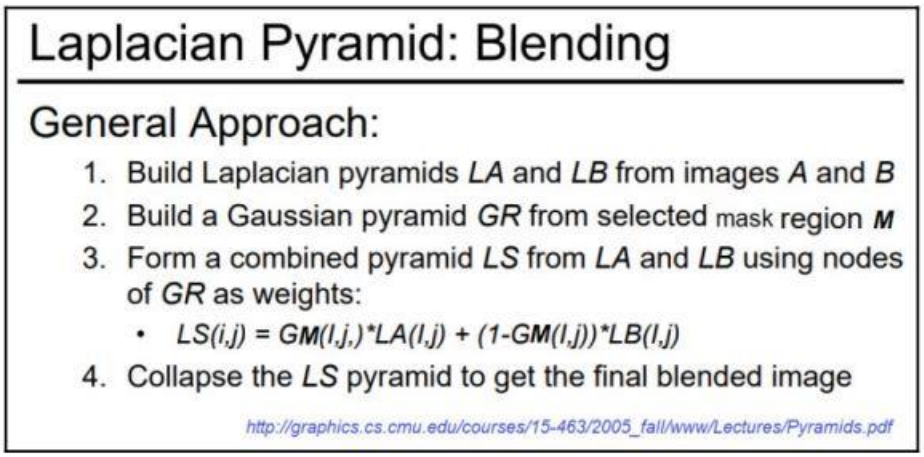
```
import cv2
import matplotlib.pyplot as plt

img1 = cv2.imread("images/parrot.jpg")
img2 = cv2.imread("images/antelops.jpeg")

# Resize second image to match first
```

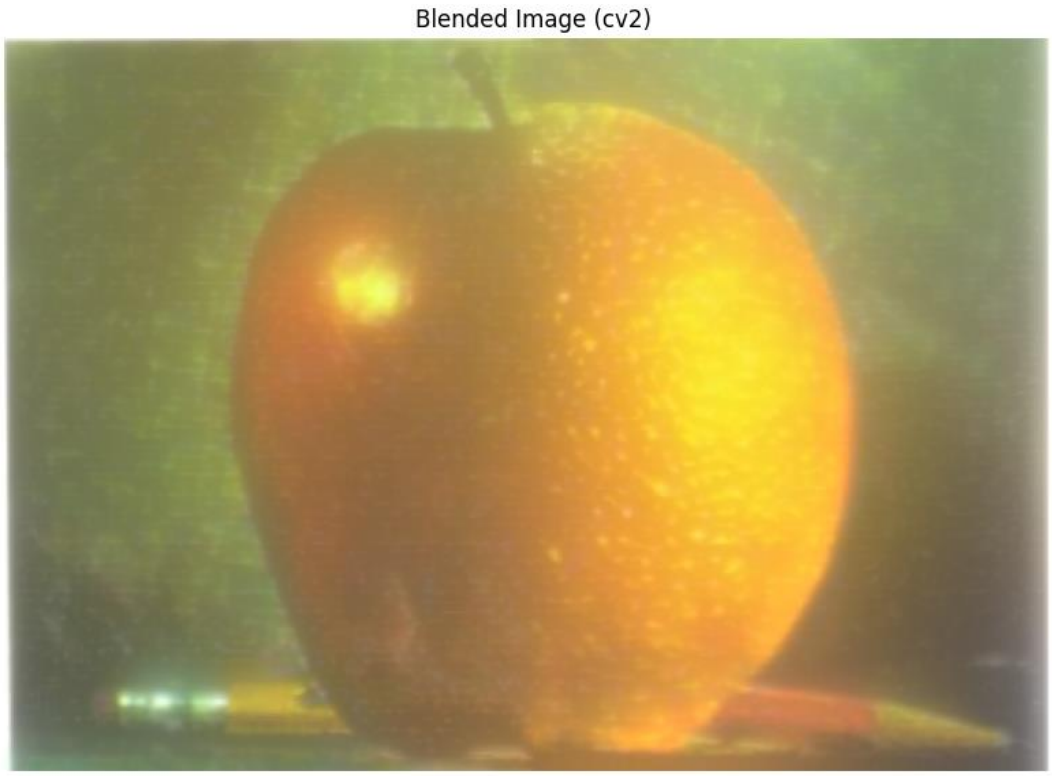
```
img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))

# Blend images
blended = cv2.addWeighted(img1, 0.7, img2, 0.3, 0)

# Convert BGR → RGB for matplotlib
blended_rgb = cv2.cvtColor(blended, cv2.COLOR_BGR2RGB)

plt.imshow(blended_rgb)
plt.axis('off')
plt.title("Blended Image")
plt.show()
```



Blending images with pyramids

Let's say we have a couple of RGB color input images, A (apple) and B (orange), and a third binary mask image, M; all three images are of the same size. The objective is to blend image A with B, guided by the mask, M (if a pixel in the mask image M has a value of 1, it implies that this pixel is be taken from the image A, otherwise from image B). The following algorithm can be used to blend two images using the Laplacian pyramids of images A and B (by computing the blended pyramid using the linear combination of the images at the same levels of the Laplacian pyramids from A and B, with the weights from the same level of the Gaussian pyramid of the mask image M), followed by reconstructing the output image from the Laplacian pyramid:



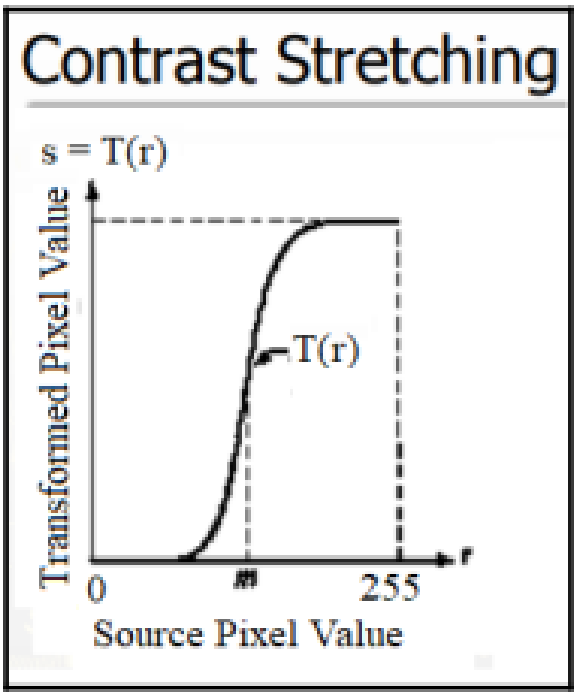


2. Changing Contrast and Brightness

 Use cv2.convertScaleAbs().

Contrast

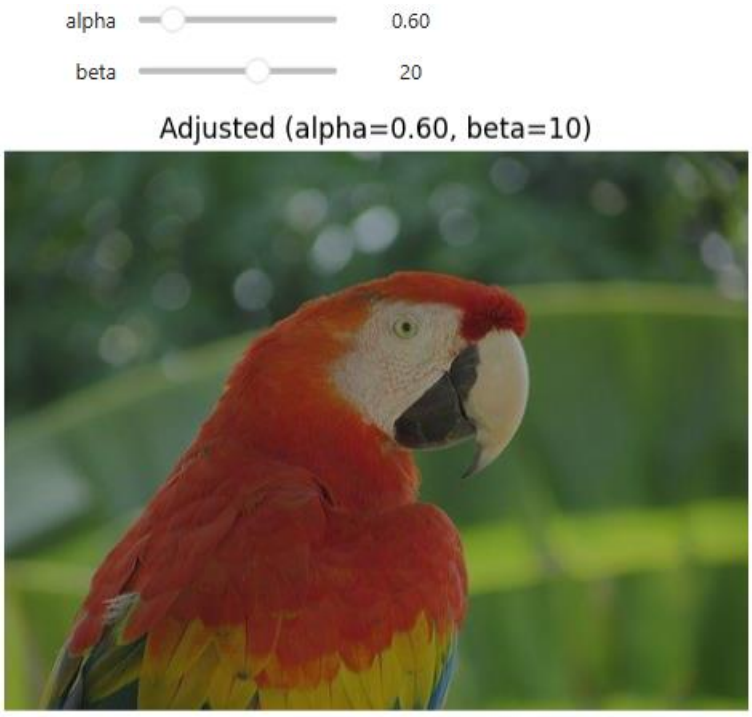
The contrast stretching operation takes a low-contrast image as input and stretches the narrower range of the intensity values to span a desired wider range of values in order to output a high-contrast output image, thereby enhancing the image's contrast. It is just a linear scaling function that is applied to image pixel values, and hence the image enhancement is less drastic (than its more sophisticated counterpart histogram equalization, to be described shortly). The following screenshot shows the point transformation function for contrast stretching:



As can be seen from the previous screenshot, the upper and lower pixel value limits (over which the image is to be normalized), need to be specified before the stretching can be performed (for example, for a gray-level image, the limits are often set to 0 and 255, in order for the output image to span the entire range of available pixel values). All we need to find is a suitable value of m from the CDF of the original image. The contrast stretching transform produces higher contrast than the original by darkening the levels below the value m (in other words, stretching the values toward the lower limit) in the original image and brightening the levels previous to value m (stretching the values toward the upper limit) in the original image. The following sections describe how to implement contraststretching using the PIL library

```
# # alpha: contrast, beta: brightness
# adjusted = cv2.convertScaleAbs(img1, alpha=1.5, beta=50)


# cv2.imshow("Contrast & Brightness", adjusted)
# cv2.waitKey(0)
import cv2
import matplotlib.pyplot as plt


# Load image
img1 = cv2.imread("images/parrot.jpg")


# Adjust contrast (alpha) and brightness (beta)
```

```
adjusted = cv2.convertScaleAbs(img1, alpha=1.5, beta=50)

# Convert BGR → RGB for matplotlib
adjusted_rgb = cv2.cvtColor(adjusted, cv2.COLOR_BGR2RGB)

# Display safely in notebook
plt.imshow(adjusted_rgb)
plt.axis('off')
plt.title("Contrast & Brightness Adjusted")
plt.show()
```

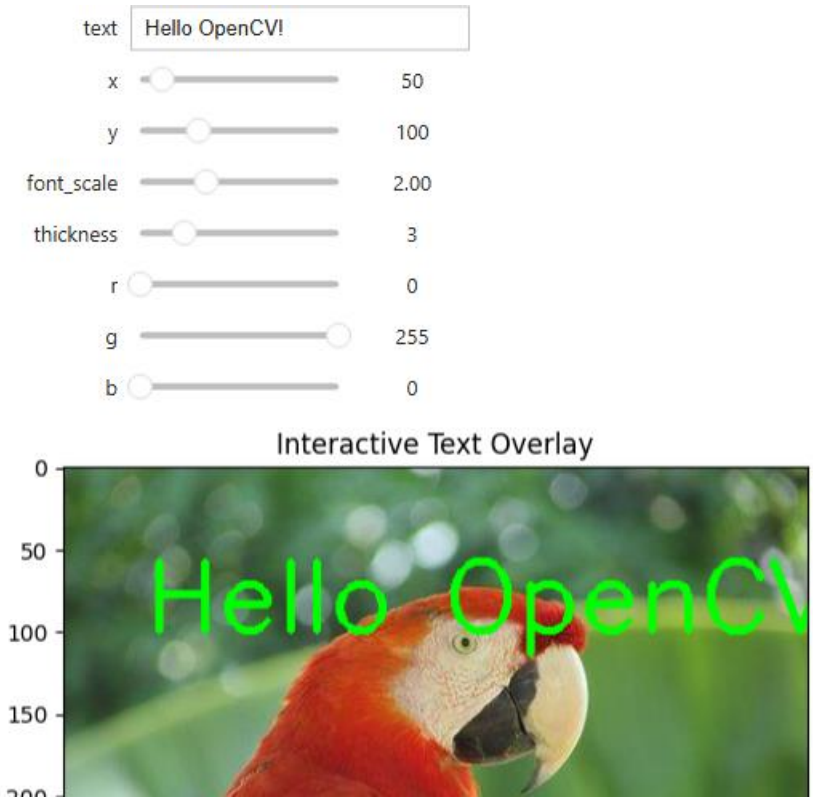



```
: <function __main__.adjust(alpha=1.0, beta=0)>
```

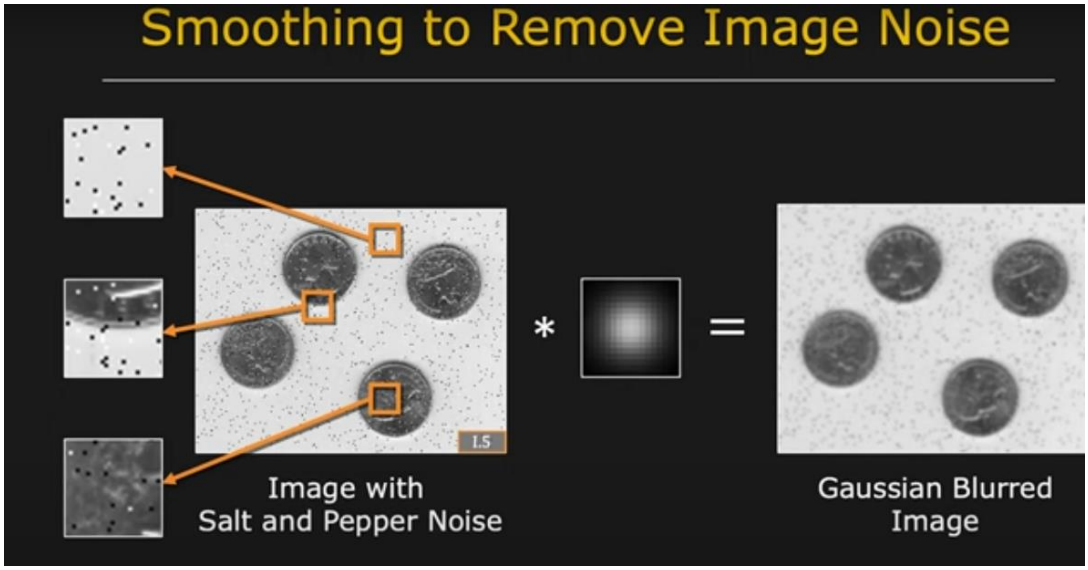
Adding Text to Images

□ Use cv2.putText()

```python
import cv2
import matplotlib.pyplot as plt

# Load image
img1 = cv2.imread("images/parrot.jpg")
output = img1.copy()

# Put text on image
cv2.putText(output, "Hello OpenCV!", (50, 100),
            cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 3)

# Convert BGR → RGB for matplotlib
output_rgb = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)

# Display safely in notebook
plt.imshow(output_rgb)
plt.axis('on')
plt.title("Text on Image")
plt.show()
```



Smoothing Images:



Problem with Smoothing

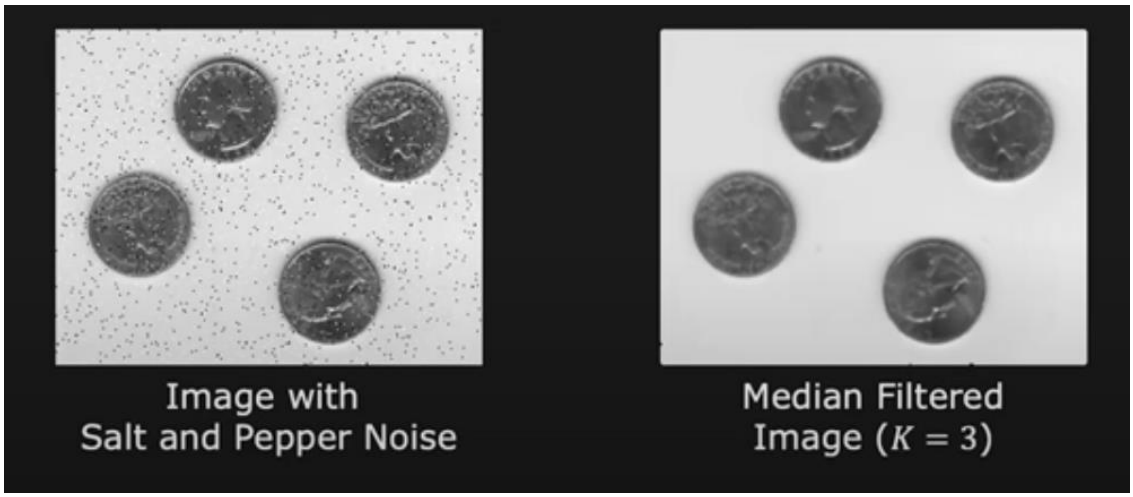
-Does Not remove outlies(Noise)

- Smooths edges(Blur)

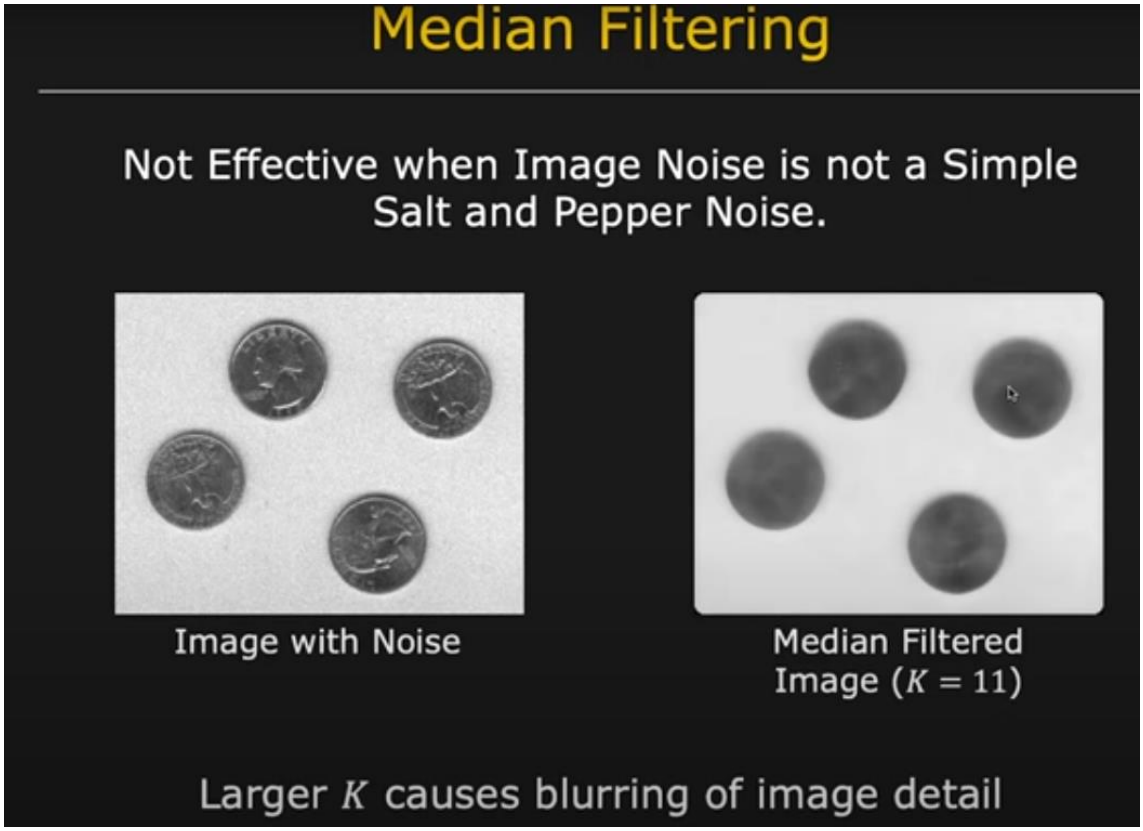
Median filter:

Median Filtering

1.Sort the K2 Values in window Centered at The Pixel
2.Assign The Middle Value(Median) to Pixel



Non-Linear Operation
(Cannot be Implemented using convolution)

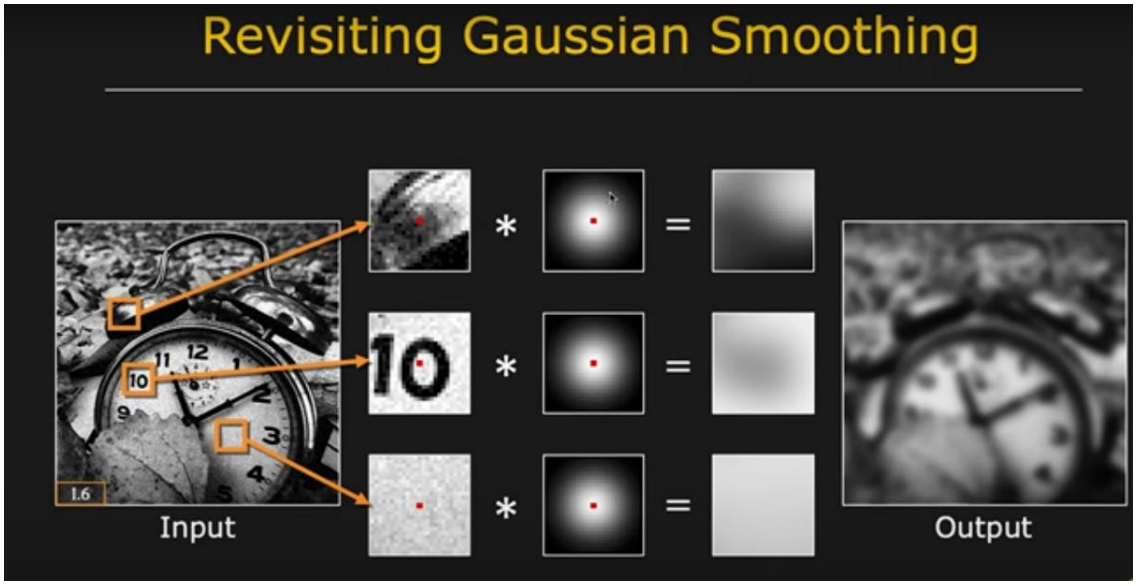


It is the best order statistic filter; it replaces the value of a pixel by the median of gray levels in the Neighborhood of the pixel.

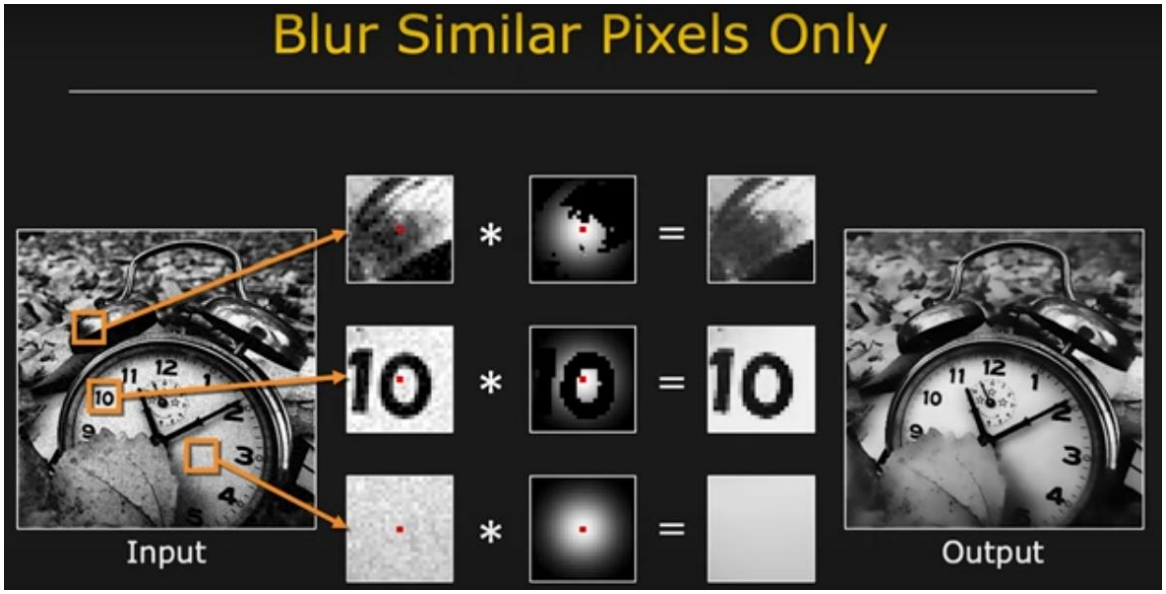
$$\hat{f}(x,y) = \underset{(s,t)\in Sxy}{\text{median}}\{g(s,t)\}$$

The original of the pixel is included in the computation of the median of the filter are quite possible because for certain types of random noise, the provide excellent noise reduction capabilities with considerably less blurring then smoothing filters of similar size. These are effective for bipolar and unipolar impulse noise.

GAUSSIAN LOWPASS FILTERS:



Same Gaussian Kernel is used everywhere.
Blurs across edges



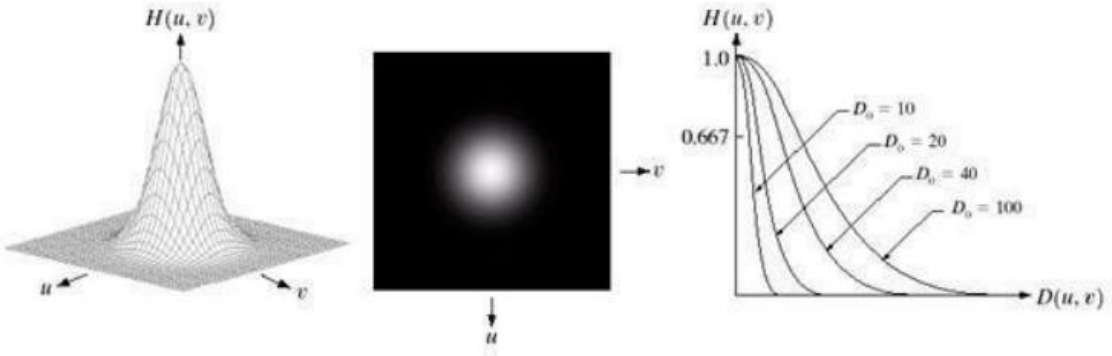
"Bias" Gaussian Kernel such that Pixcels not similar in Intensity to Center Pixel Receive a lower weight

The form of these filters in two dimensions is given by

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

- This transfer function is smooth, like Butterworth filter.
- Gaussian in frequency domain remains a Gaussian in spatial domain
- Advantage: No ringing artifacts.

Where D_0 is the cutoff frequency. When D(u,v) = D_0, the GLPF is down to 0.607 of its maximum value. This means that a spatial Gaussian filter, obtained by computing the IDFT of above equation., will have no ringing. Fig..Shows a perspective plot, image display and radial cross sections of a GLPF function.



a b c

FIGURE (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.

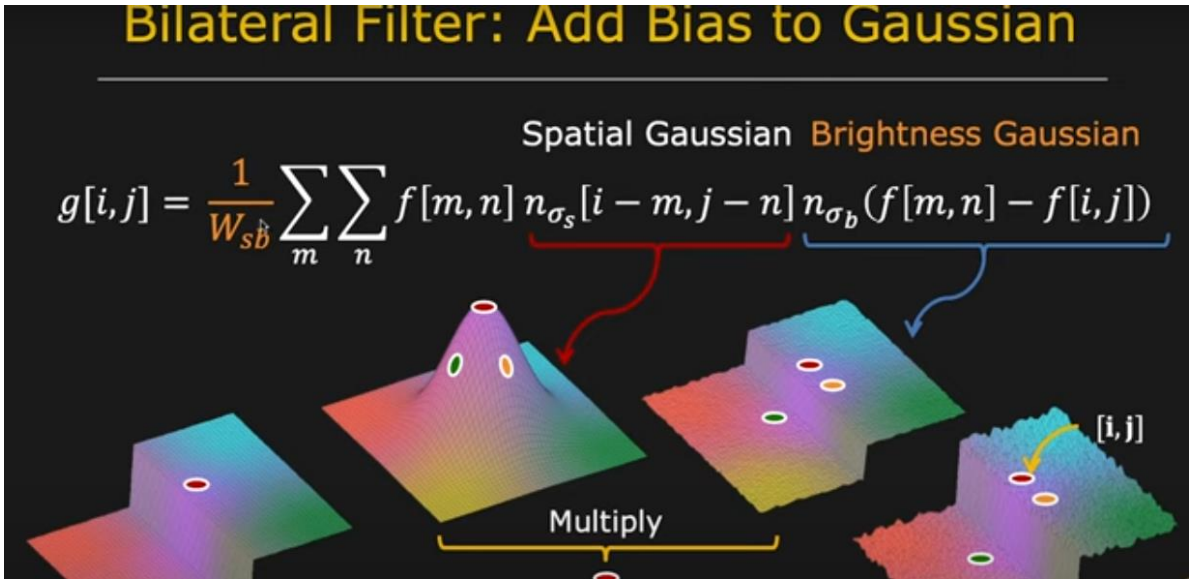
Fig. (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c). Filter radial cross sections for various values of D_0

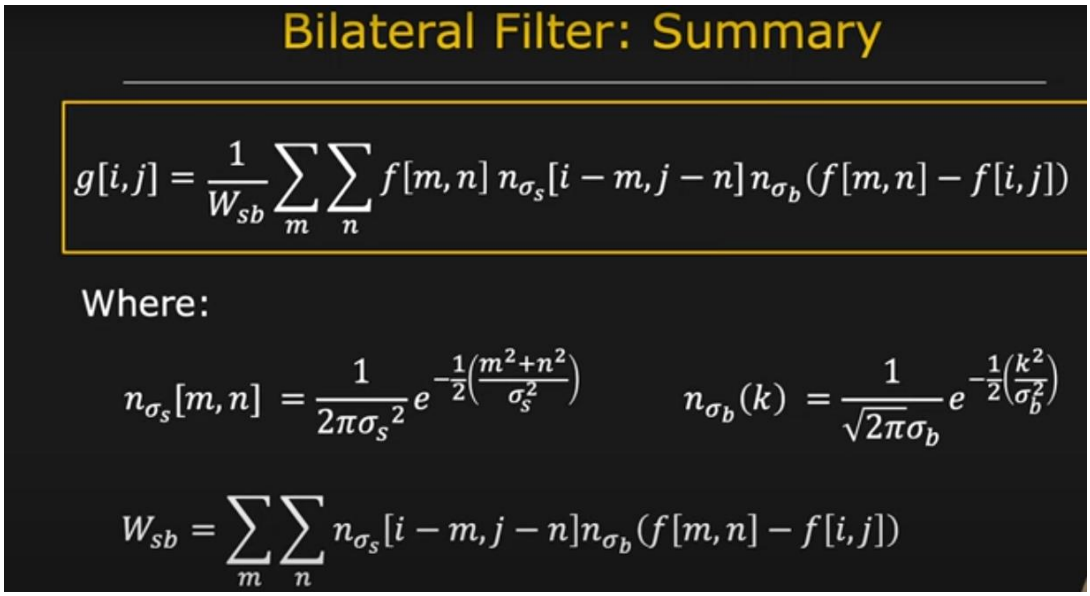
a b c

Fig. (a) Original image (784x 732 pixels). (b) Result of filtering using a GLPF with D0 = 100. (c) Result of filtering using a GLPF with D0 = 80. Note the reduction in fine skin lines in the

magnified sections in (b) and(c).
Fig. shows an application of lowpass filtering for producing a smoother, softer-looking result from a sharp original. For human faces, the typical objective is to reduce the sharpness of fine skin lines and small blemished.

Bilateral Filter





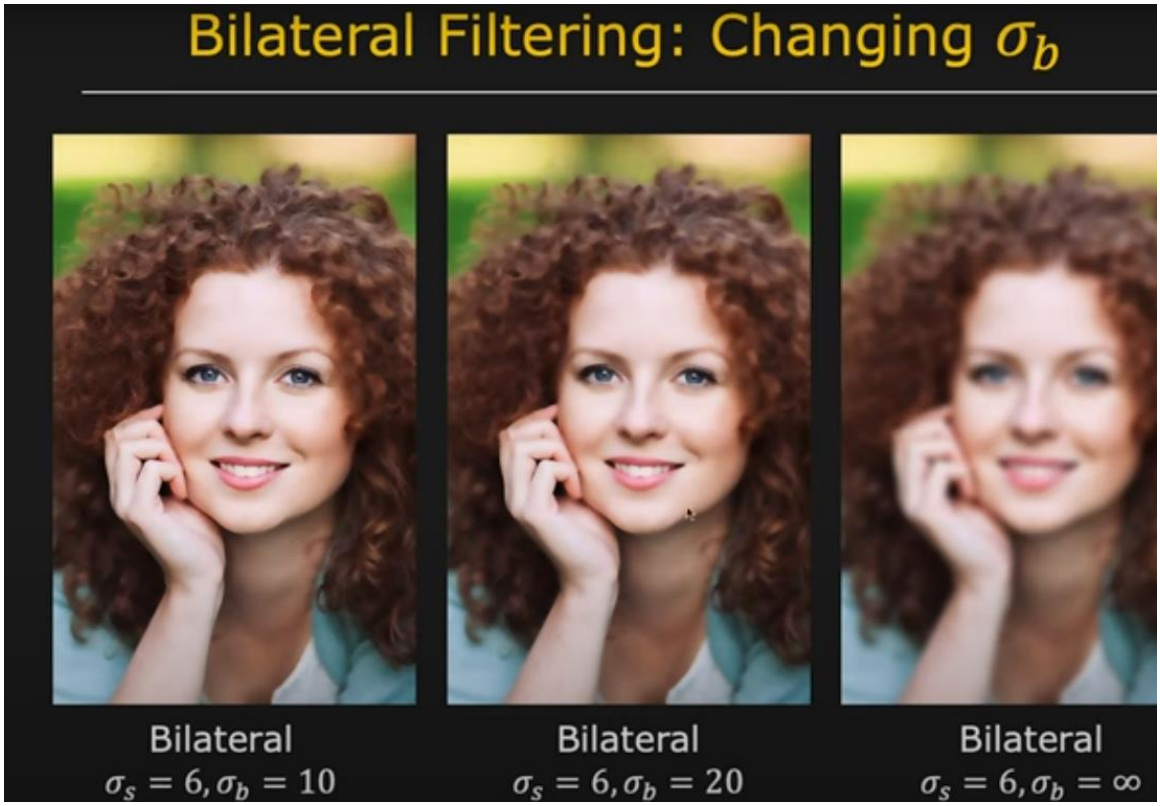
Non-Linear Operation
(Cannot be Implemented using Convolution)

Gaussian vs. Bilateral Filtering: Example

Original | Gaussian $\sigma_s = 2$ | Bilateral $\sigma_s = 2, \sigma_b = 10$

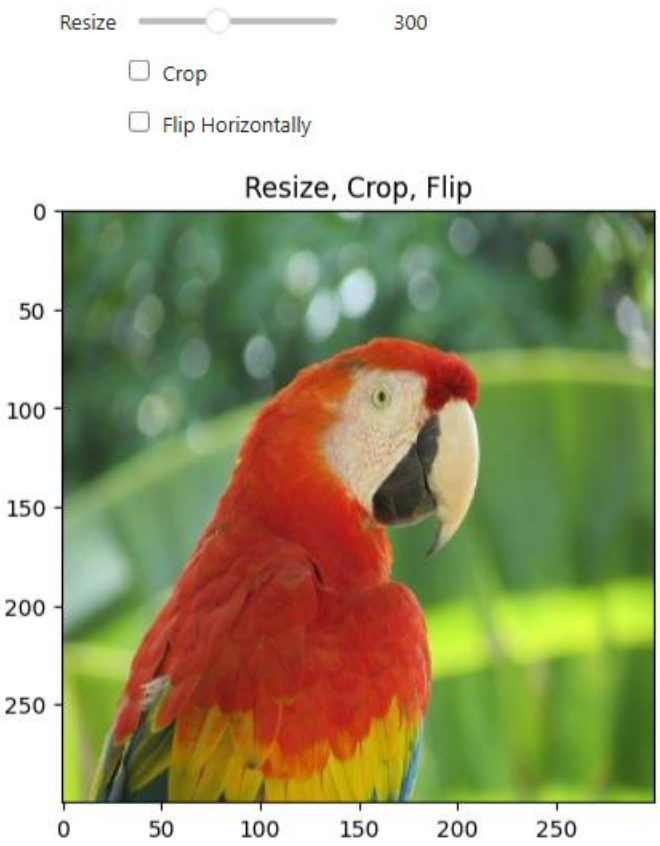
Gaussian vs. Bilateral Filtering: Exampl

Original | Gaussian $\sigma_s = 4$ | Bilateral $\sigma_s = 4, \sigma_b = 10$



Changing the Shape of Images

```
resized = cv2.resize(img1, (300, 300))
cropped = img1[50:200, 100:300]   # crop region
flipped = cv2.flip(img1, 1)       # horizontal flip
```

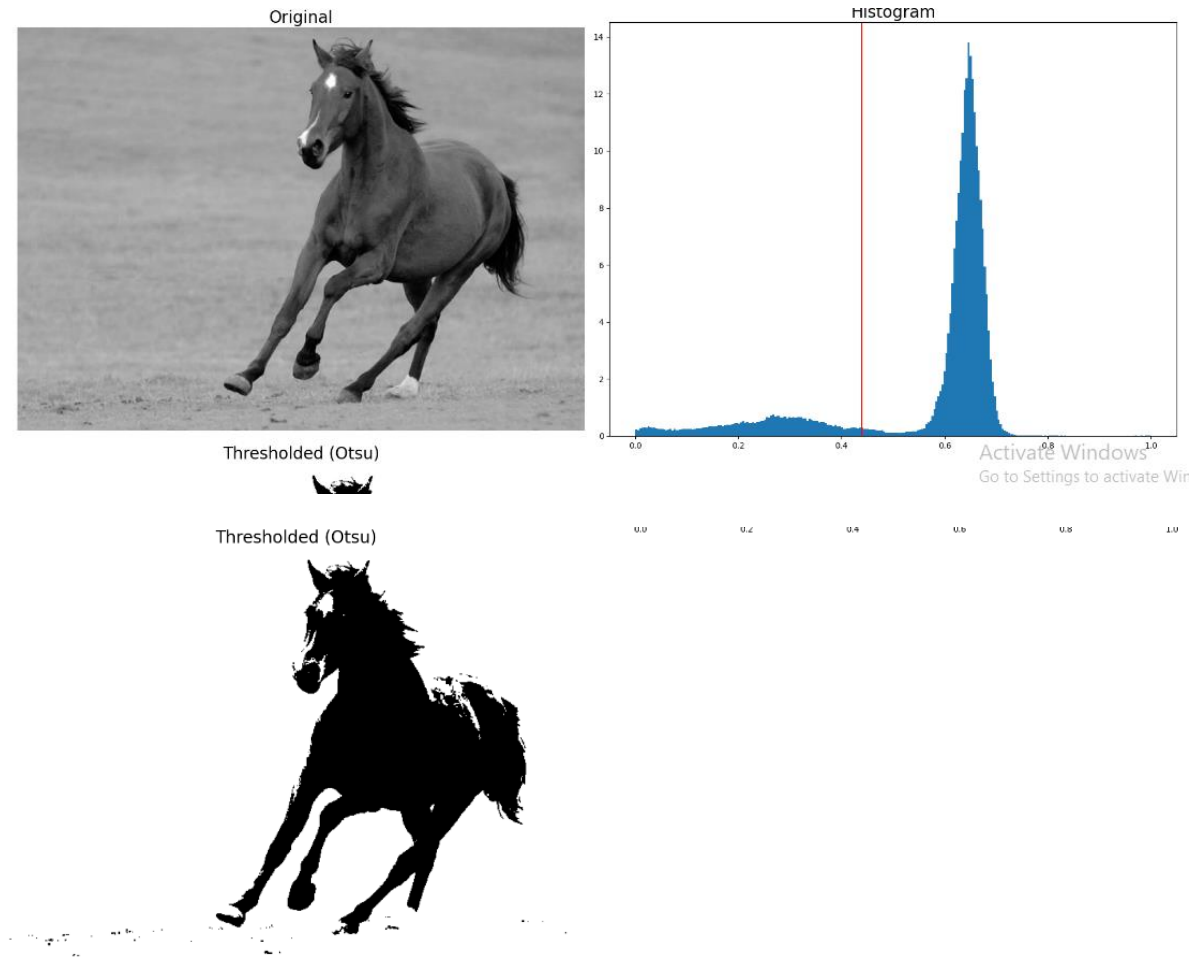


Effecting Image Thresholding

Binary thresholding and adaptive thresholding.

```
import numpy as np
from skimage.transform import (hough_line, hough_line_peaks, hough_circle,
hough_circle_peaks)


from skimage.draw import circle_perimeter
from skimage.feature import canny
from skimage.data import astronaut
from skimage.io import imread, imsave
from skimage.color import rgb2gray, gray2rgb, label2rgb
from skimage import img_as_float
from skimage.morphology import skeletonize
from skimage import data, img_as_float
import matplotlib.pyplot as plt
from matplotlib import cm
from skimage.filters import sobel, threshold_otsu
```

```
from skimage.feature import canny
from skimage.segmentation import felzenszwalb, slic, quickshift, watershed
from skimage.segmentation import mark_boundaries, find_boundaries
# Load and preprocess image
image = rgb2gray(imread('images/horse.jpg'))
thresh = threshold_otsu(image)
binary = image > thresh

# Plotting
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(20, 15))
ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Original', size=20)
ax[0].axis('off')

ax[1].hist(image.ravel(), bins=256, density=True)  # <-- use density instead of normed
ax[1].set_title('Histogram', size=20)
ax[1].axvline(thresh, color='r')

ax[2].imshow(binary, cmap=plt.cm.gray)
ax[2].set_title('Thresholded (Otsu)', size=20)
ax[2].axis('off')

ax[3].axis('off')

plt.tight_layout()
plt.show()
```



Performing Histogram Equalization.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
from skimage.color import rgb2gray
from skimage.filters.rank import enhance_contrast
from skimage.morphology import disk
from skimage import exposure, img_as_ubyte

# Helper function
def plot_gray_image(ax, image, title):
    ax.imshow(image, cmap='gray')
```

```
    ax.set_title(title)
    ax.axis('off')
    ax.set_adjustable('box')  # valid value

# Load image and convert to grayscale
image = rgb2gray(imread('images/squirrel.jpg'))

# Add noise
sigma = 0.05
noisy_image = np.clip(image + sigma * np.random.standard_normal(image.shape), 0, 1)

# Convert to uint8 for rank filters
noisy_uint8 = img_as_ubyte(noisy_image)

# Local morphological contrast enhancement
enhanced_image = enhance_contrast(noisy_uint8, disk(5))

# Adaptive histogram equalization (works with float)
equalized_image = exposure.equalize_adapthist(noisy_image)

# Plot results
fig, axes = plt.subplots(1, 3, figsize=(18, 7), sharex=True, sharey=True)
axes1, axes2, axes3 = axes.ravel()

plot_gray_image(axes1, noisy_image, 'Original')
plot_gray_image(axes2, enhanced_image / 255.0, 'Local morphological contrast
enhancement')  # normalize for display
plot_gray_image(axes3, equalized_image, 'Adaptive Histogram equalization')

plt.show()
```


Original
Local morphological contrast enhancement
Adaptive Histogram equalization