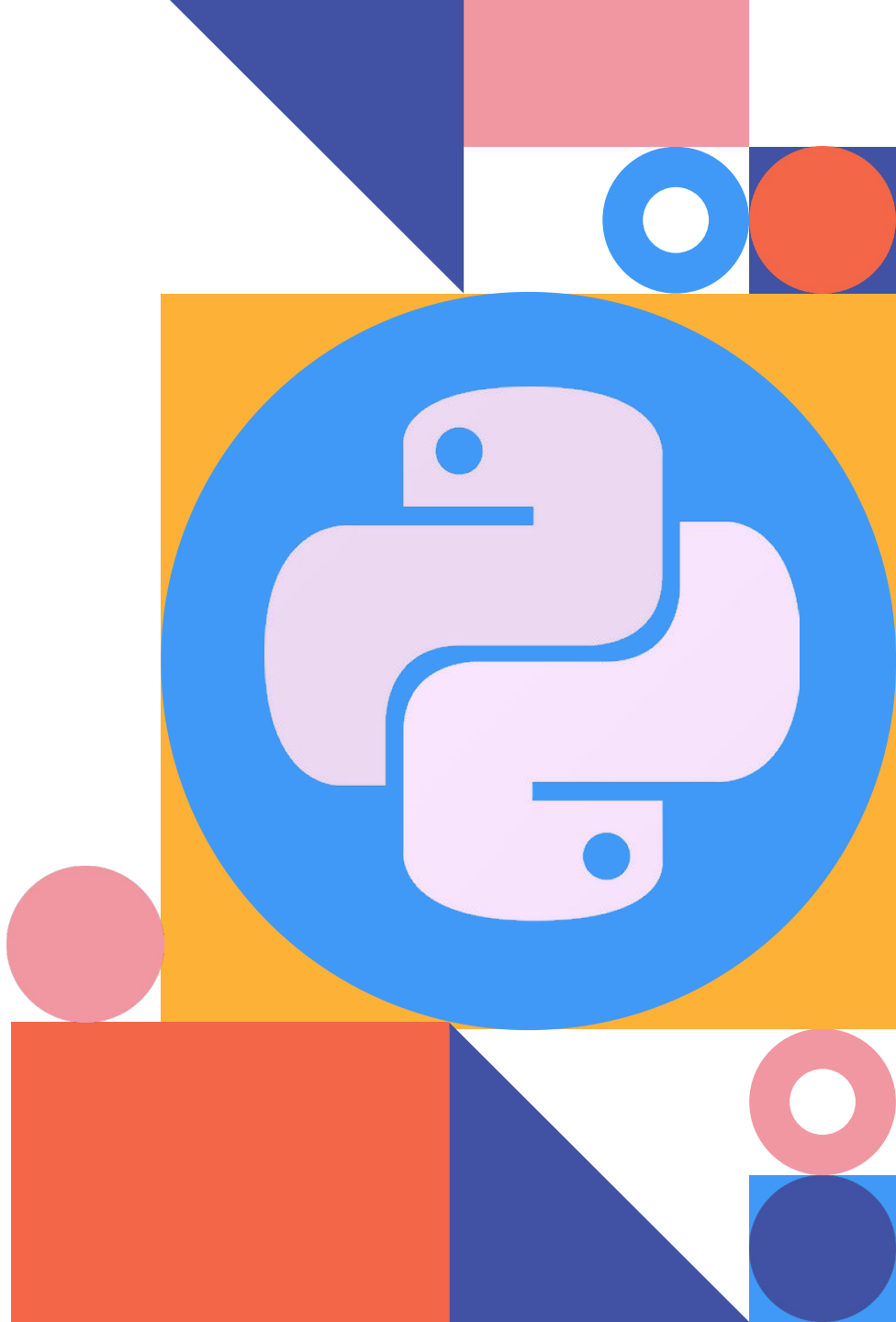
Image Processing II

Yan-Fu Kuo
Dept. of Biomechatronics Engineering
National Taiwan University



Contents

01 Histogram Processing

02 Image Segmentation

- Binary segmentation
- Color Space

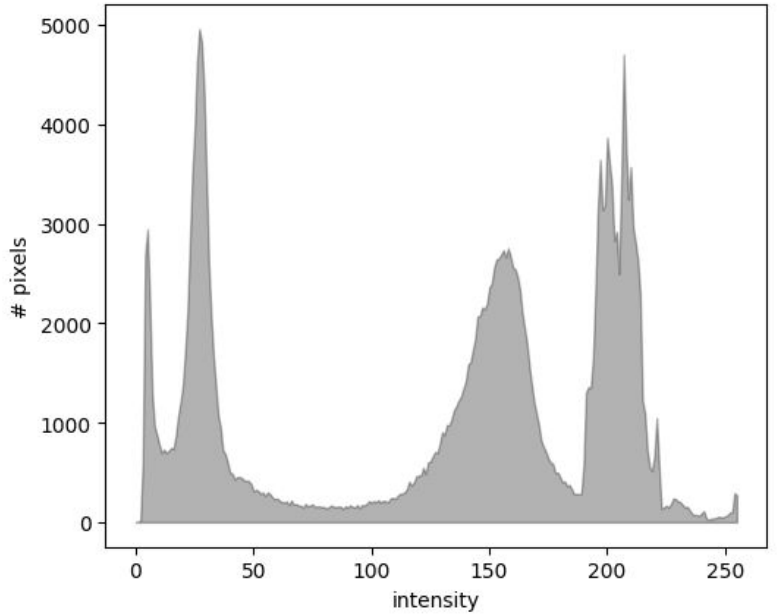
03 Spatial Filtering

- Neighborhood Operations
- Smoothing Filters

01 Histogram Processing

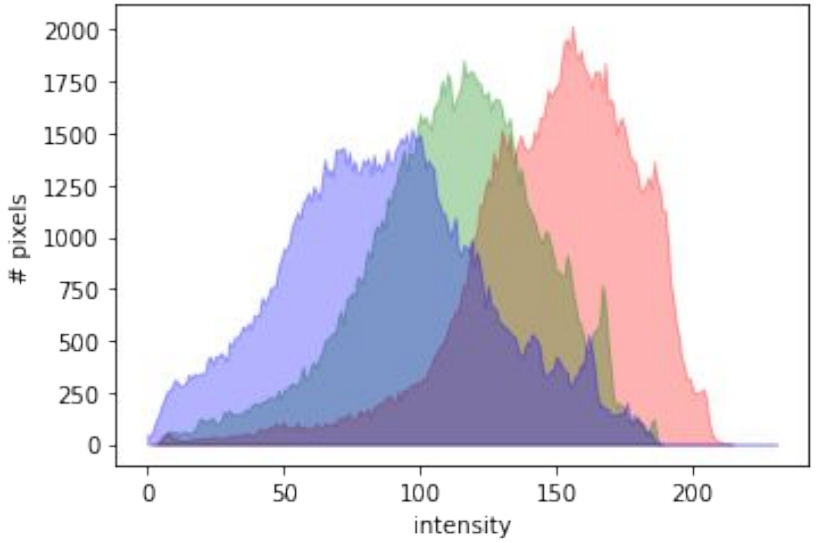
Histogram of a Graylevel Image

```
import skdemo
camera = data.camera()
skdemo.imshow_with_histogram(camera)
```



Histogram of a Color Image

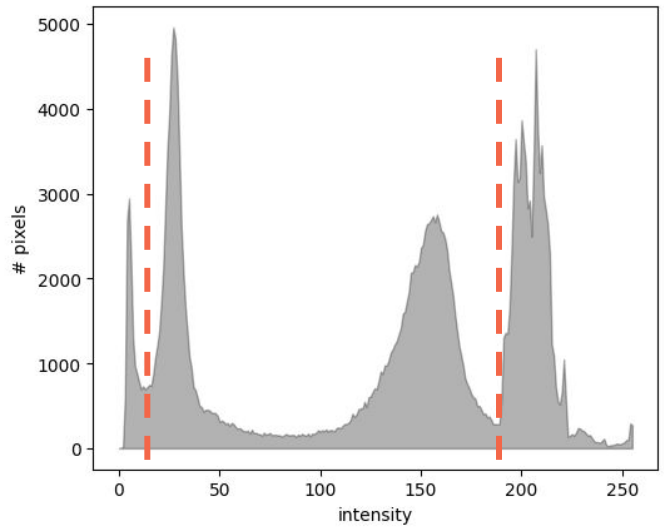
```
cat = data.chelsea()
skdemo.imshow_with_histogram(cat)
```

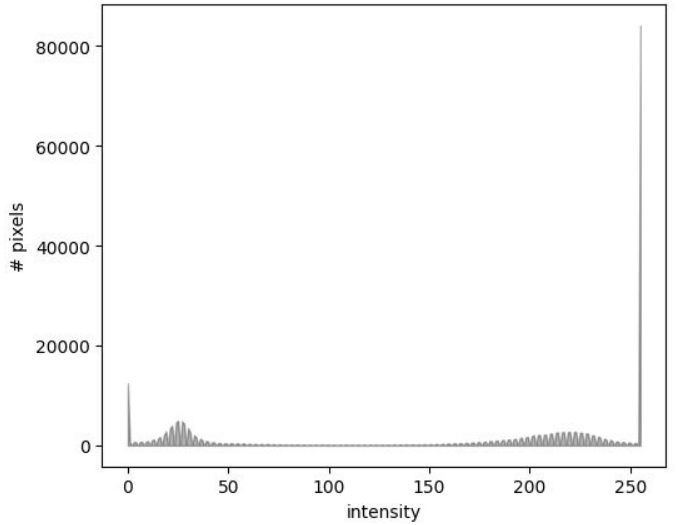


While RGB histograms are pretty, they are often not very intuitive or useful

Contrast Adjustment

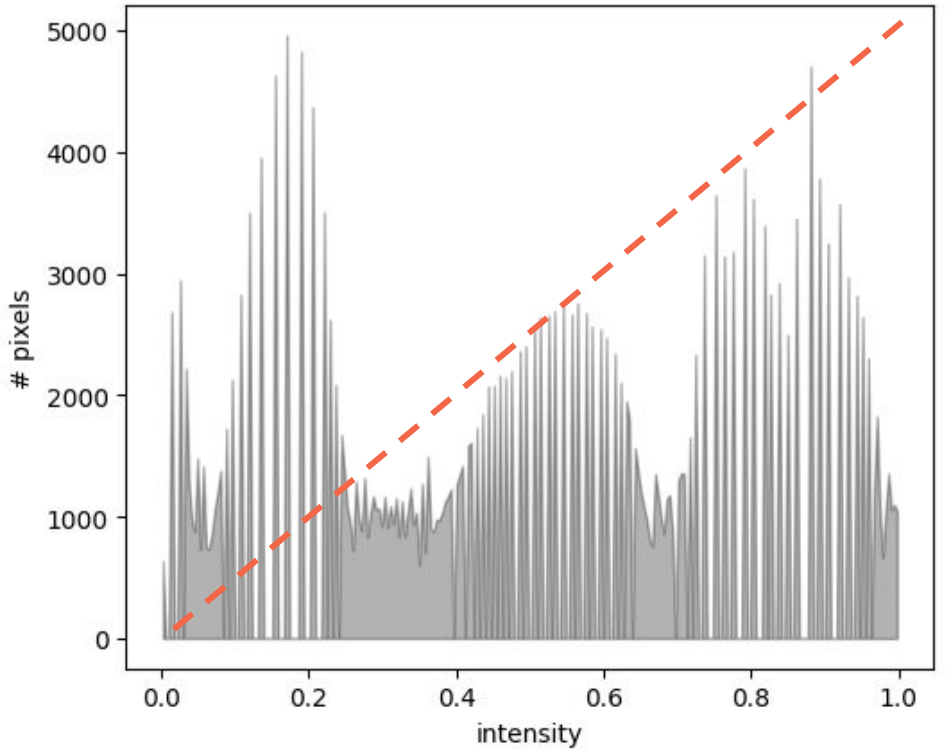
```
from skimage import exposure
high_contrast = exposure.rescale_intensity(camera, in_range=(10, 180))
skdemo.imshow_with_histogram(high_contrast)
```



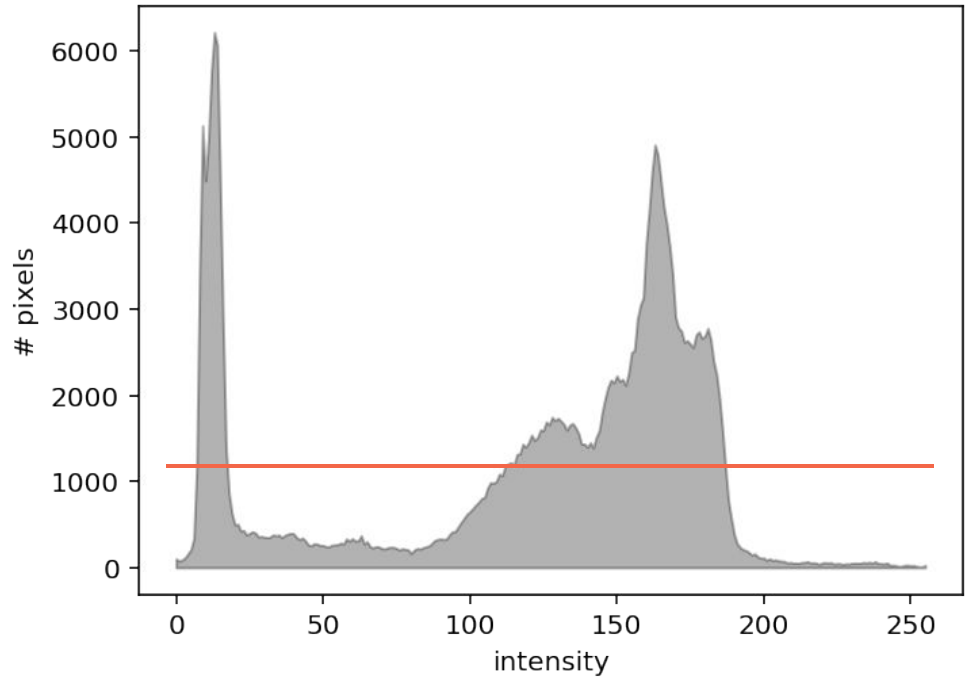


Histogram Equalization

```
equalized = exposure.equalize_hist(camera)
skdemo.imshow_with_histogram(equalized)
```



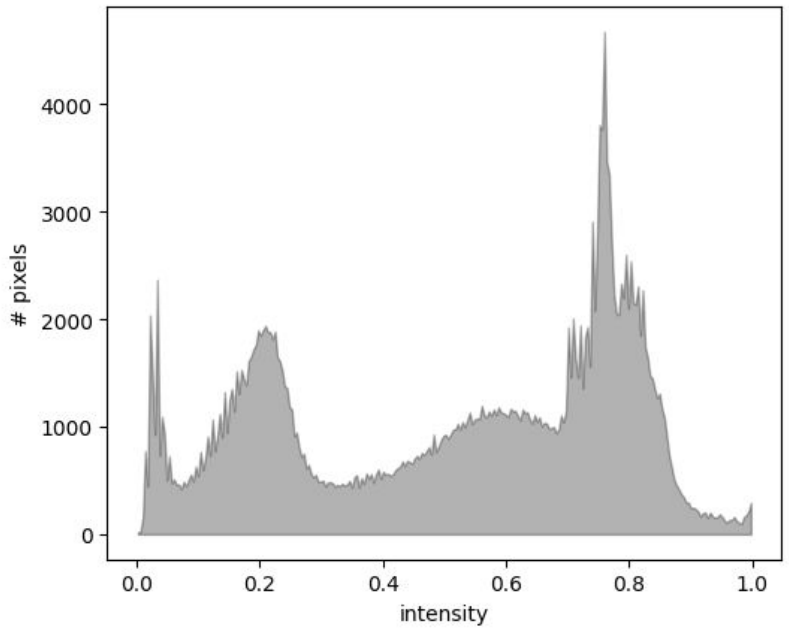
Implementation



$$\frac{512 \times 512}{256} = 1024$$

Adaptive Histogram Equalization

```
adaptive_equalized = exposure.equalize_adapthist(camera)
skdemo.imshow_with_histogram(adaptive_equalized)
```



02 Image Segmentation

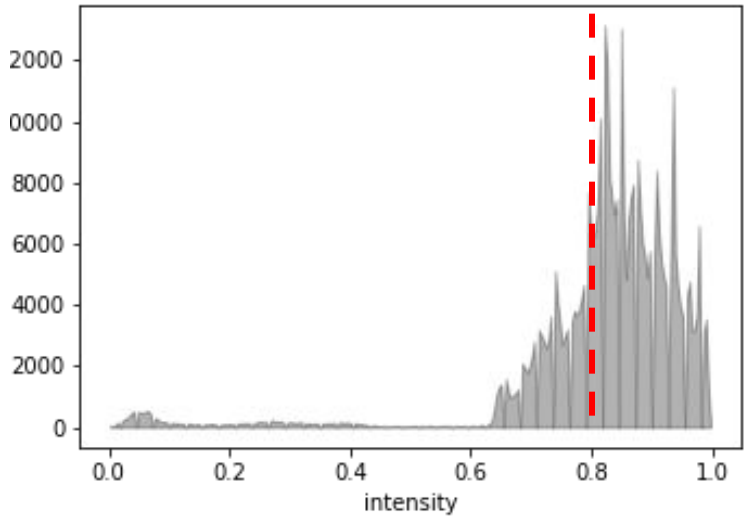
Problem Setup

Get the boat and the people in this image

Image Thresholding

A grayscale image can be turned into a binary image by using a threshold

```
boat = io.imread('boat.png', as_gray = True)
boat_mask = boat > 0.8
plt.imshow(boat_mask, cmap='gray')
```



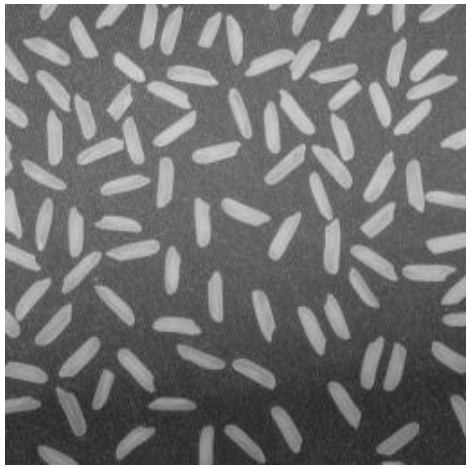
Otsu's method

`threshold_otsu` calculates an “optimal” threshold by maximizing the variance between two classes of pixels

```
boat = io.imread('boat.png', as_gray = True)
from skimage import filters
otsu_thresh = filters.threshold_otsu(boat)
print(otsu_thresh)
otsu_binary = boat > otsu_thresh
plt.imshow(otsu_binary, cmap='gray')
plt.show()
```

Exercise 1 (5 mins)

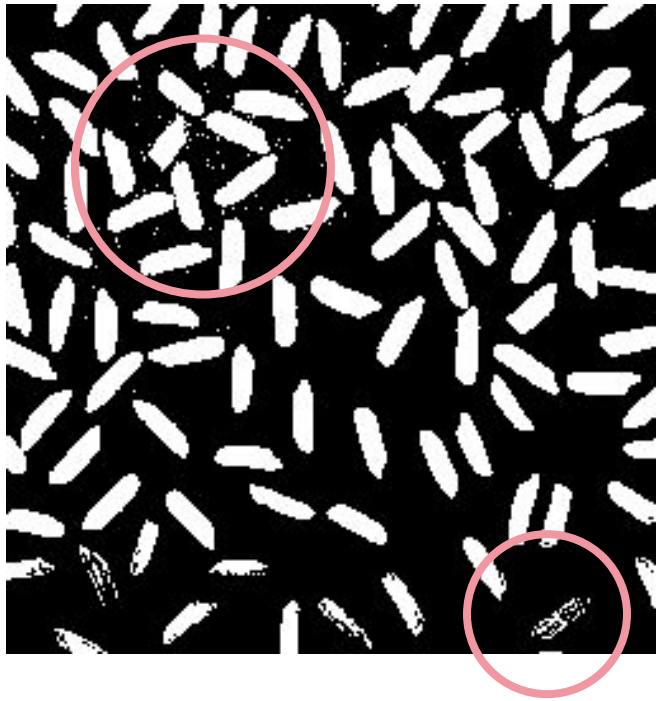
Try different threshold values to see if Otsu's method really finds the best threshold to your image.



Issues of Thresholding

The binary image is not “perfect”

- Existing sparkles in the background
- Some grains are missing



What is the cause of this issue?

How do we solve it?

Local thresholding

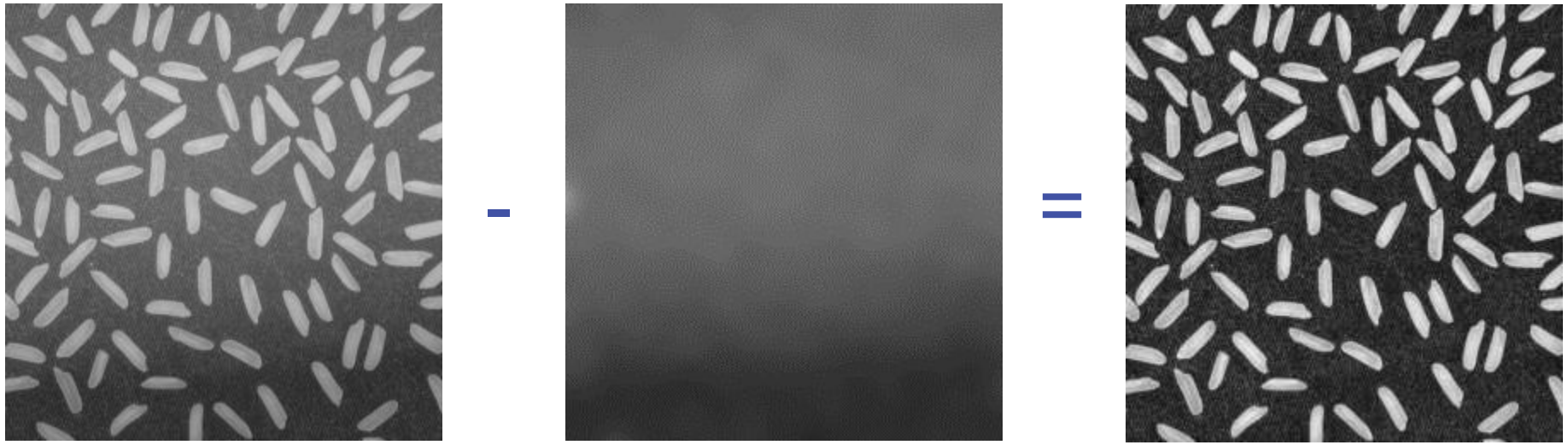
```
block_size = 55
local_thresh = filters.threshold_local(rice,
block_size, offset=-20)
```

What is the value of local_thresh?

```
local_binary = rice > local_thresh
plt.imshow(local_binary, cmap='gray')
```

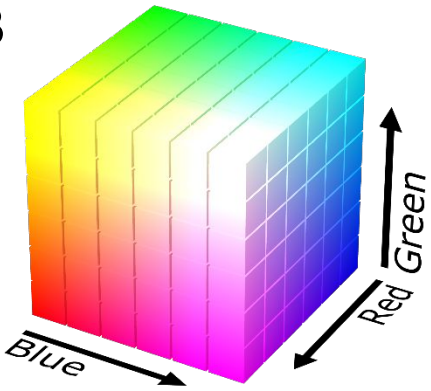
Background Estimation

```
img = img_as_float(rice)
bg = morphology.erosion(img, morphology.square(9))
bg = filters.gaussian(bg, sigma=7)
```



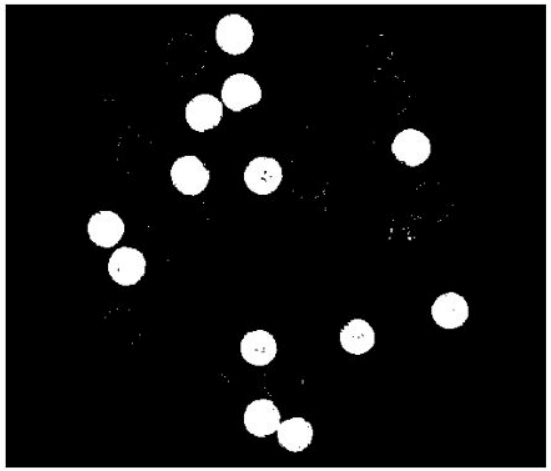
Color Segmentation

RGB



Segment green M&M's.

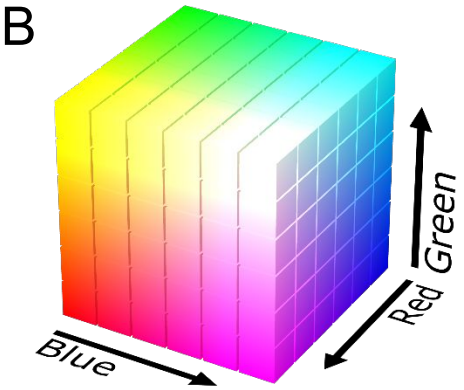
Green



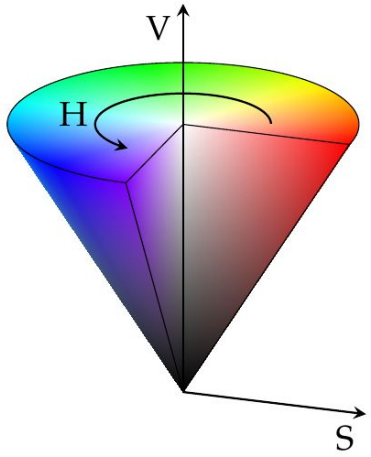
What are your strategies?

Color (Space) Conversion

RGB

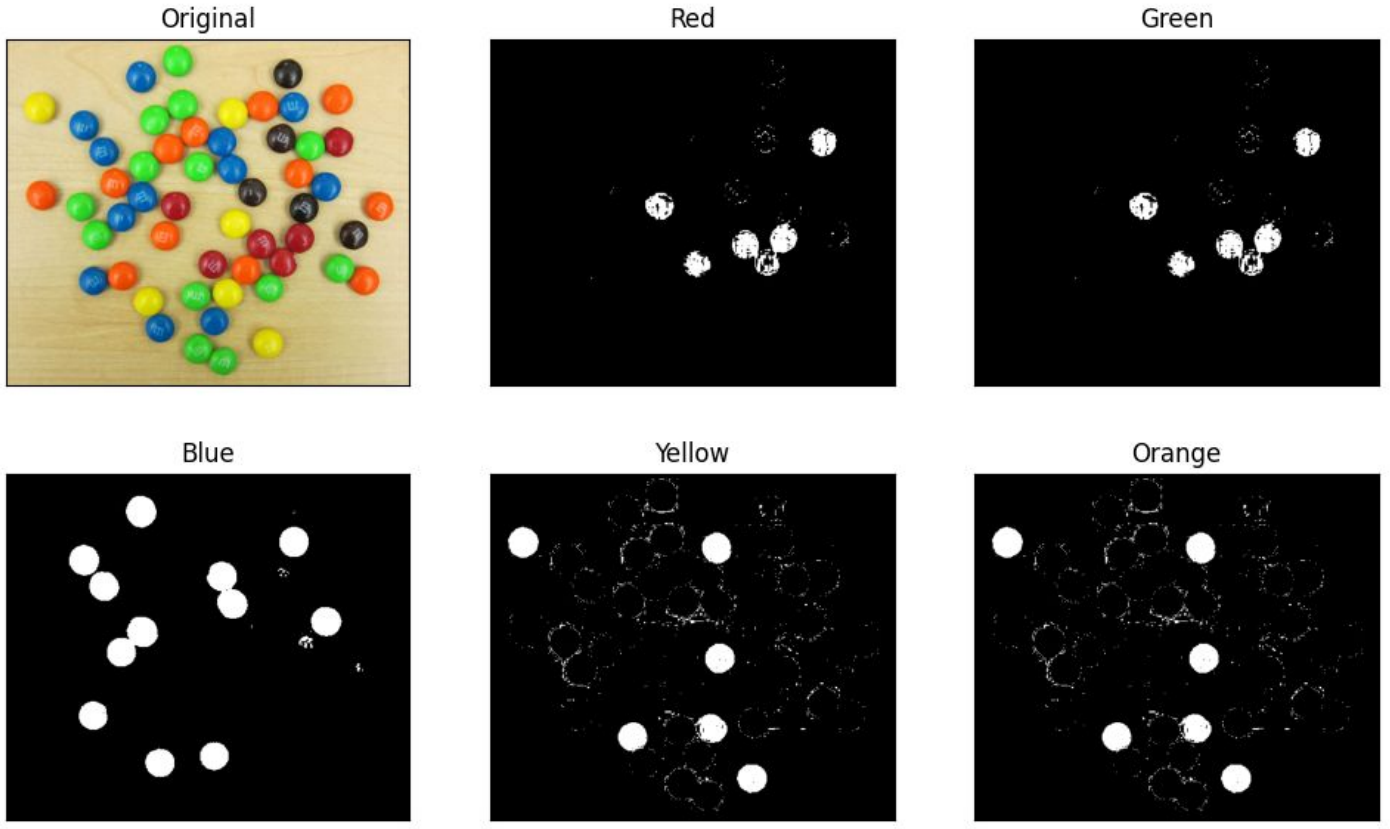


HSV




```
import numpy as np
from skimage.color import rgb2hsv
balloon = io.imread("mm.jpg")
width = balloon.shape[1]
height = balloon.shape[0]
img = np.zeros((height,width))
balloon_hsv = rgb2hsv(balloon)
```

Exercise 2 (10 mins)



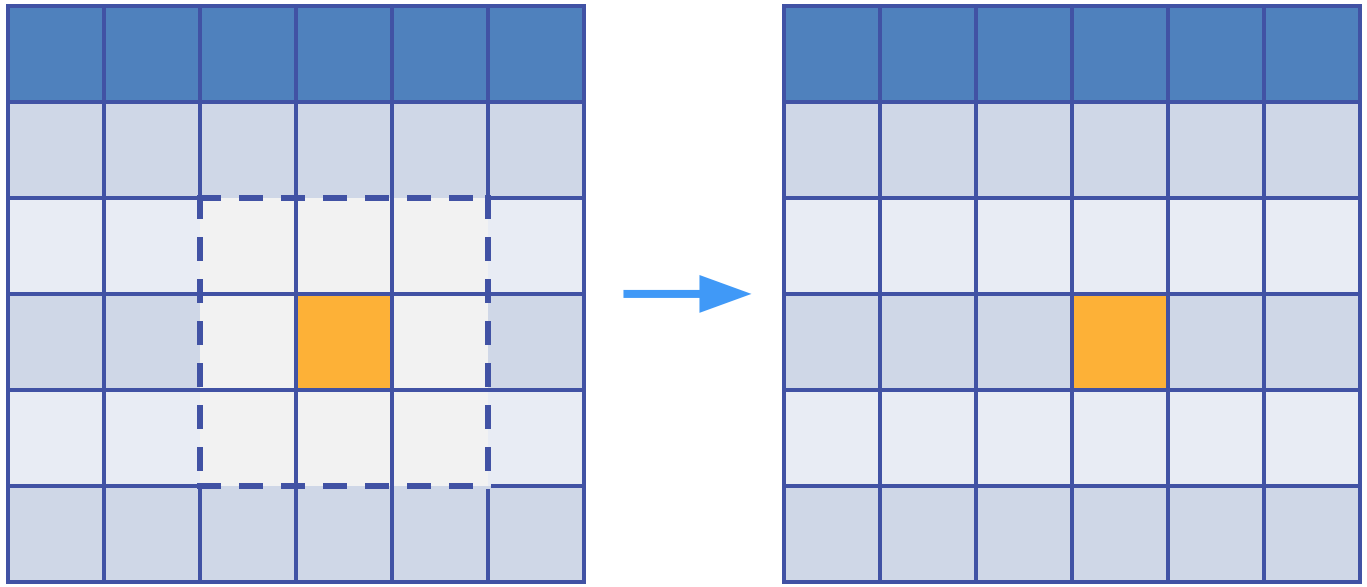
Green:

```
img[np.logical_and(balloon_hsv[:, :, 0]>0.3, balloon_hsv[:, :, 0]<0.37)] = 255
```

03 Spatial Filtering

Neighborhood Operations

Determine the gray-level of a pixel using the graylevel of its neighborhood.

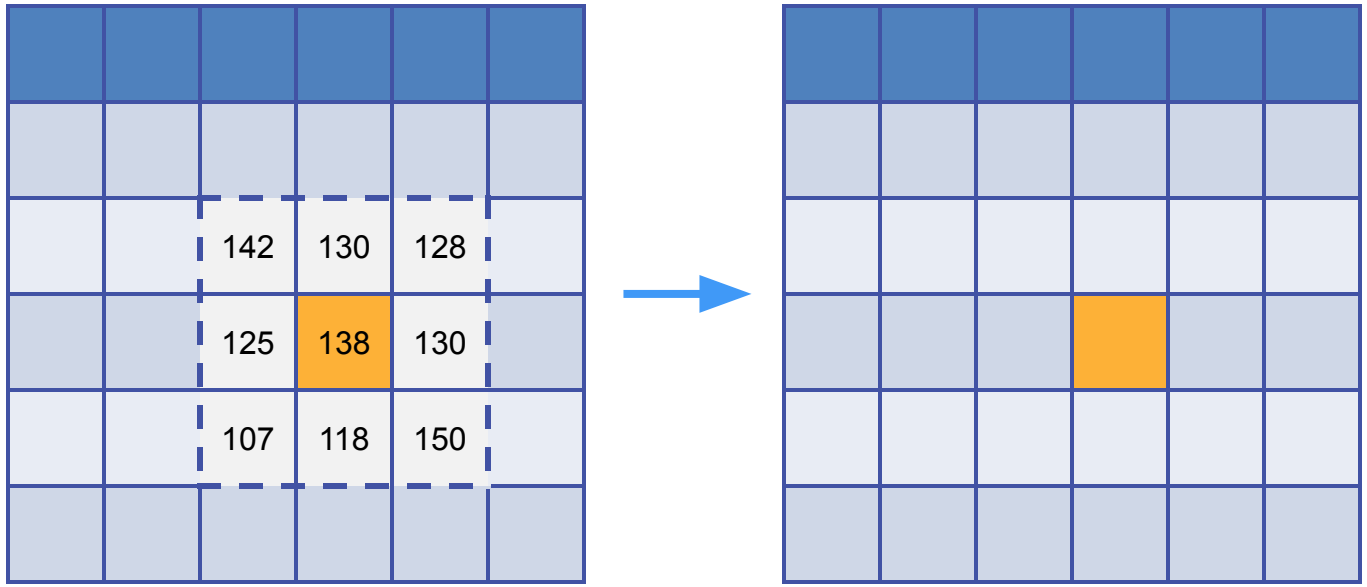


Original Image

Target Image

Simple Neighborhood Operations

Set the pixel value to the (a) maximum, (b) minimum, (c) median, or (d) mean in the neighborhood.

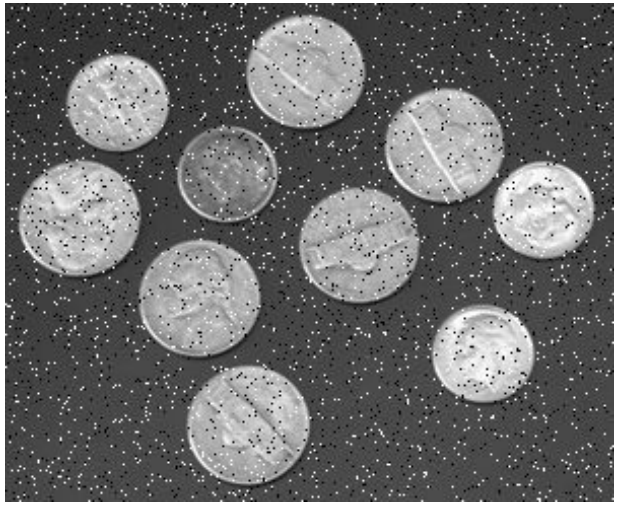


Original Image

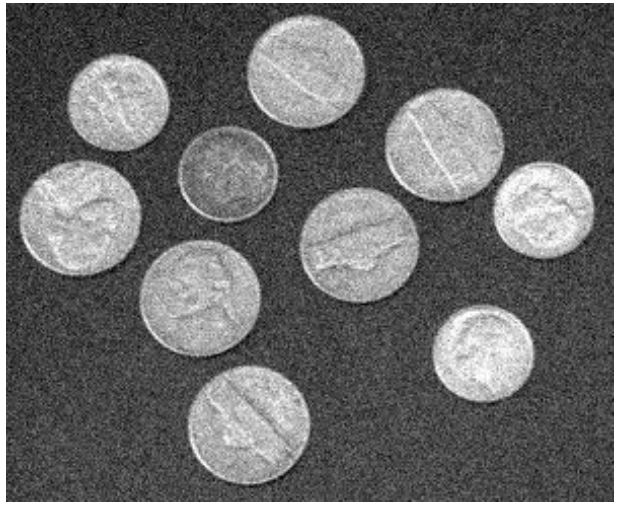
Target Image

Problem Setup

Improve the “quality” of a gray-level image.



Salt-and-pepper noise

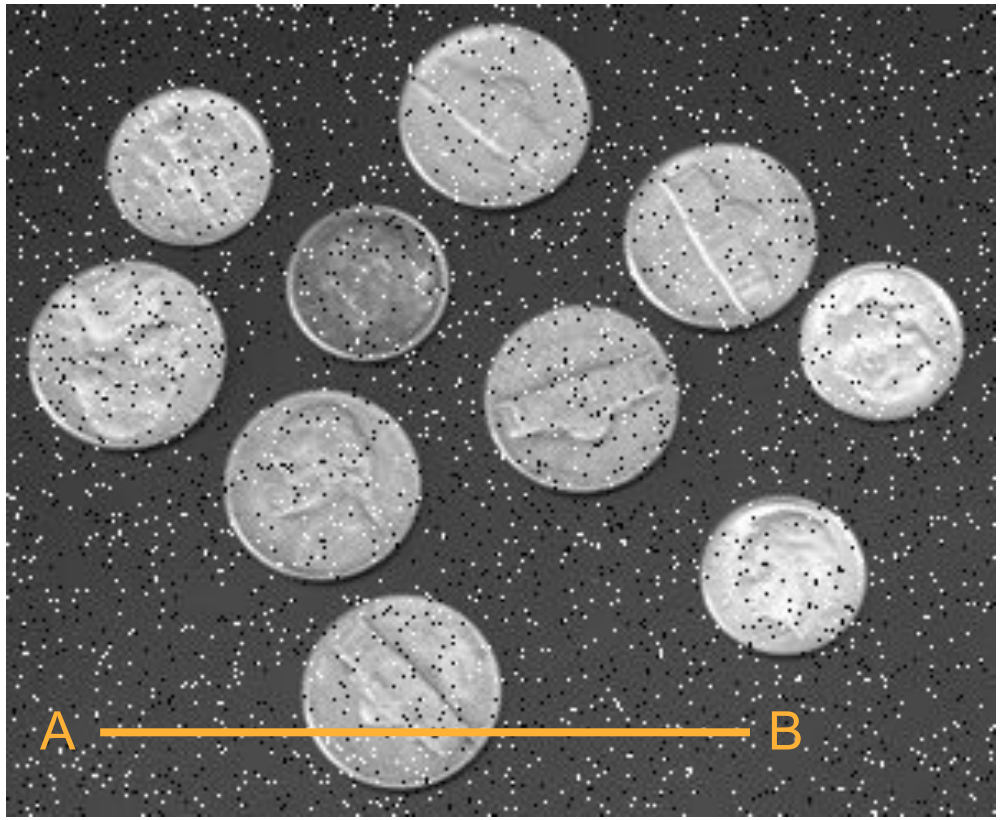


Gaussian noise

Improved image

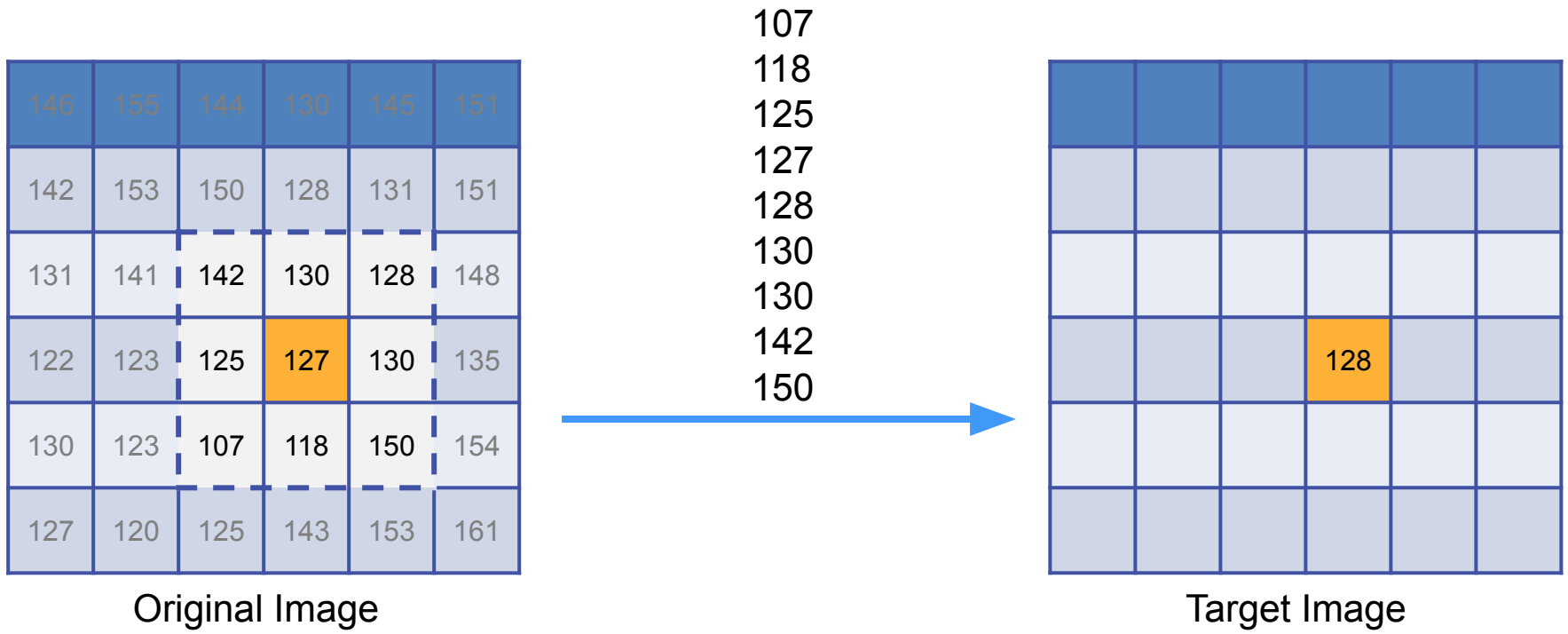
What are your strategies?

Salt-and-Pepper Noise



Salt-and-pepper noise is known as impulse noise. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels.

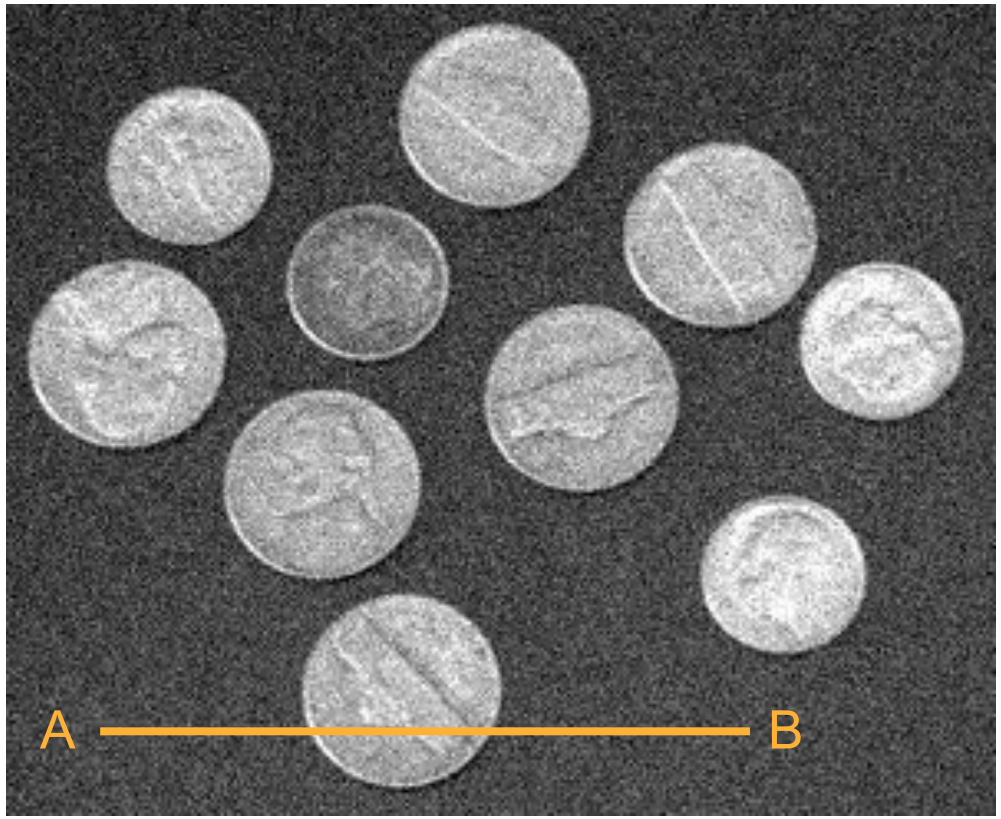
Median Neighborhood Operation



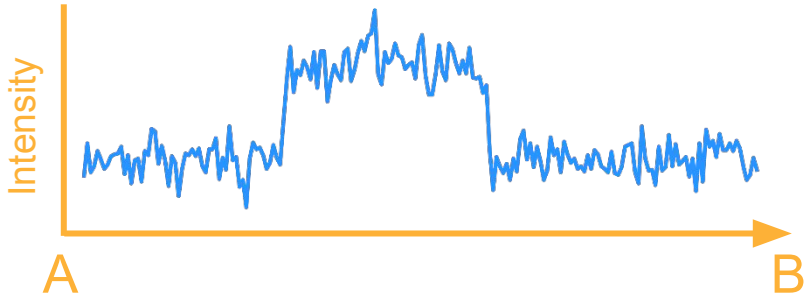
Filters for Neighborhood Operations

```
from skimage.filters.rank import median, minimum, maximum, mean
from skimage.morphology import square

neighborhood = square(width=3)
coins_denoised = median(coins_noisy, neighborhood)

plt.imshow(coins_denoised, cmap='gray')
plt.show()
```

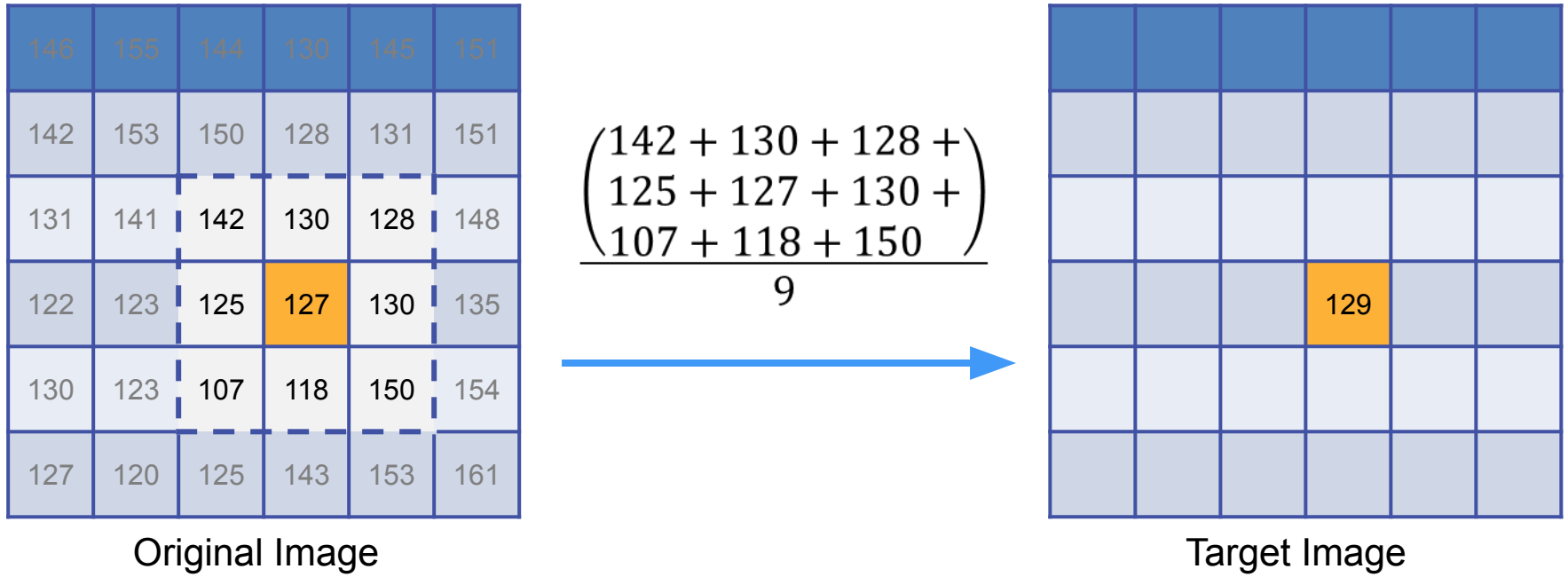
Gaussian Noise



Gaussian noise is statistical noise having a probability density function equal to that of the Gaussian distribution.

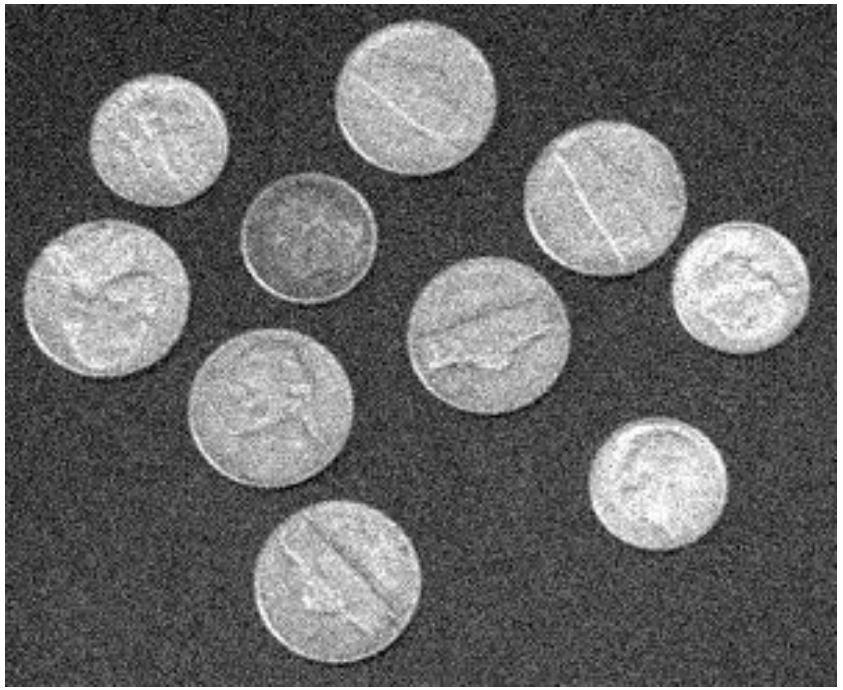


Mean Neighborhood Operation



Exercise 3 (5 mins)

Write a program to denoise the image ‘coins_gaussian.png’



Which of the following operations is better for denoising images with Gaussian noise?

(A) `median`
(B) `minimum`
(C) `maximum`
(D) `mean`

Mean Filter as Convolution ($*$)

$$\left(142 \times \frac{1}{9}\right) + \left(130 \times \frac{1}{9}\right) + \left(128 \times \frac{1}{9}\right) + \cdots + \left(118 \times \frac{1}{9}\right) + \left(150 \times \frac{1}{9}\right) = 129$$

142	130	128
125	127	130
107	118	150

Original Image (Cropped)

$*$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

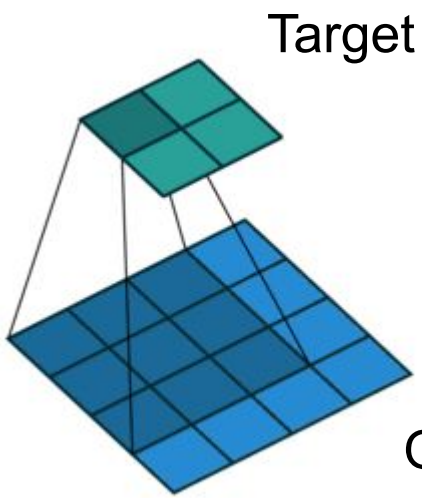
Mean **Filter**

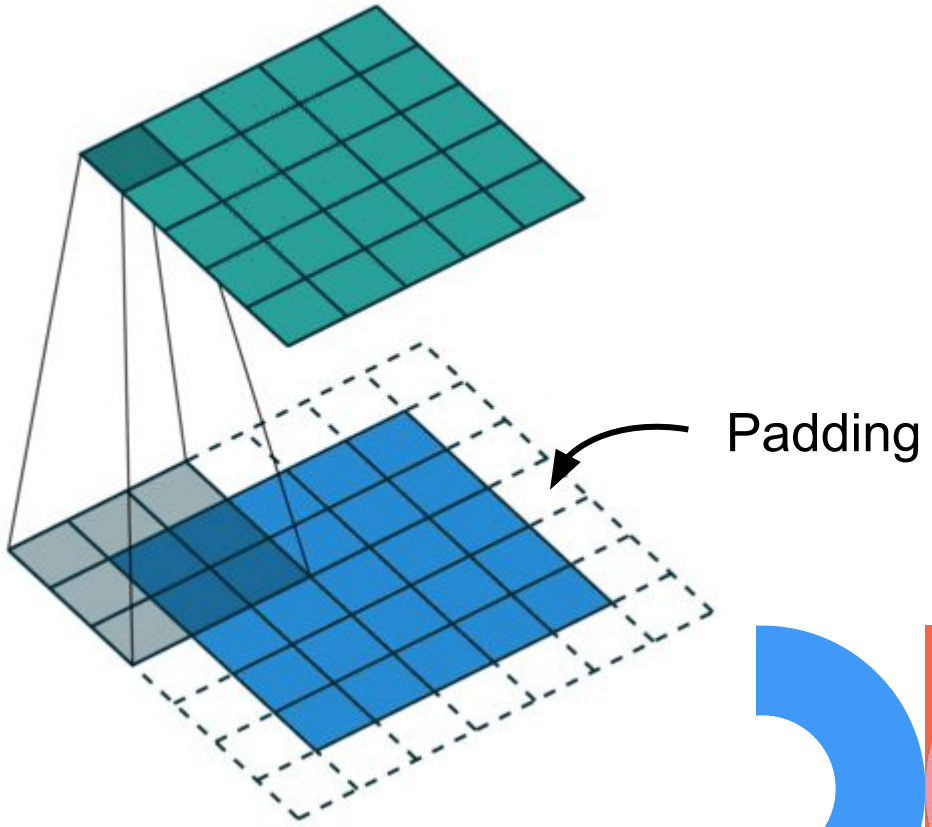
$=$

129

Target Image (Cropped)

Convolution ($*$)





Mean Filter as Convolution

```
from scipy.signal import convolve2d

filter2d = np.ones((3,3))/9

coins_smoothed = convolve2d(coins, filter2d)
```

Do you see any difference from using `filters.rank.mean`?

Try increasing the filter size

Exercise 4 (5 mins)

What do images look like after convolving them with the two filters?

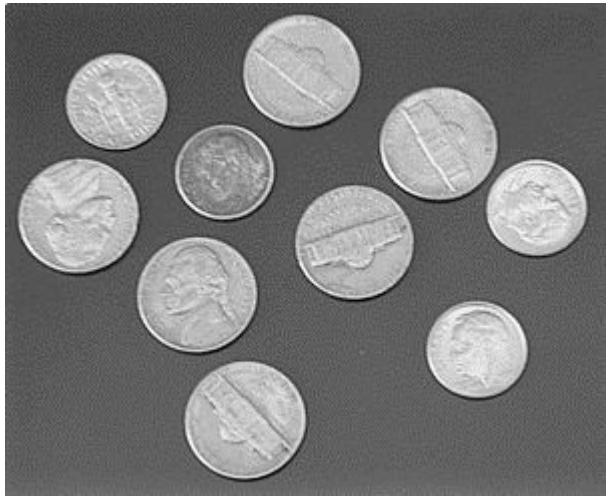
0	0	0
0	1	0
0	0	0

1	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Smoothing and Sharpening

Smoothed image

Original image



Sharpened image

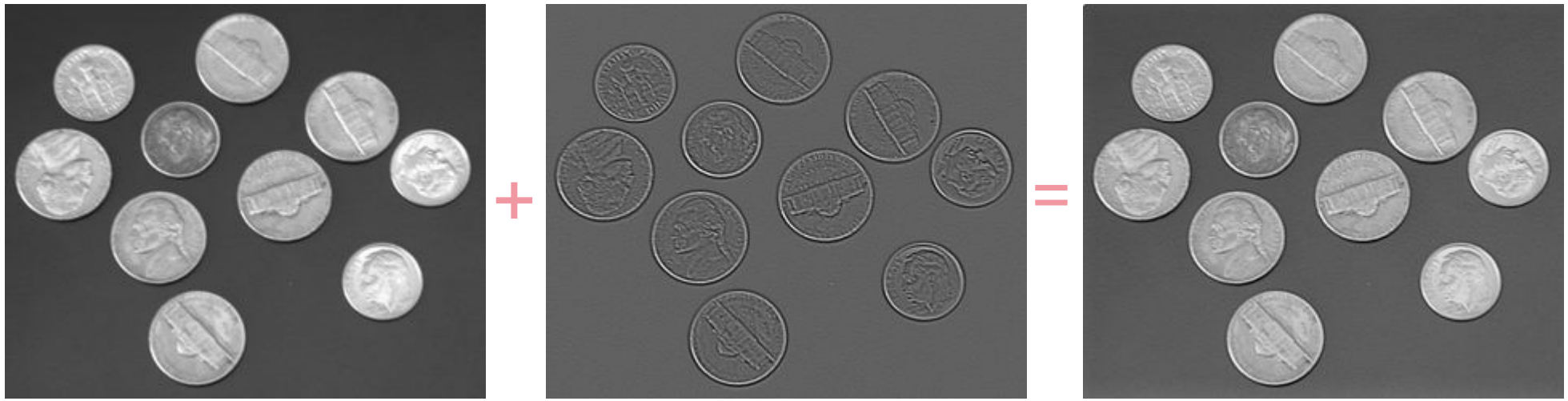
“Details” of an Image

Original image

Smoothed image

Details

Image Sharpening



Original image + Details = Sharpened image

Exercise 5 (10 mins)

Generate a filter that sharpens an image

Original image

$*$

?	?	?
?	?	?
?	?	?

$=$

Sharpened image

Hint:
Original image – Smoothed image = Details
Original image + Details = Sharpened image