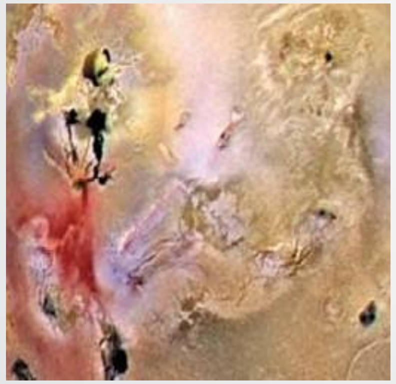


CSC 3141

IMAGE PROCESSING LABORATORY

05 – Intensity Transformations

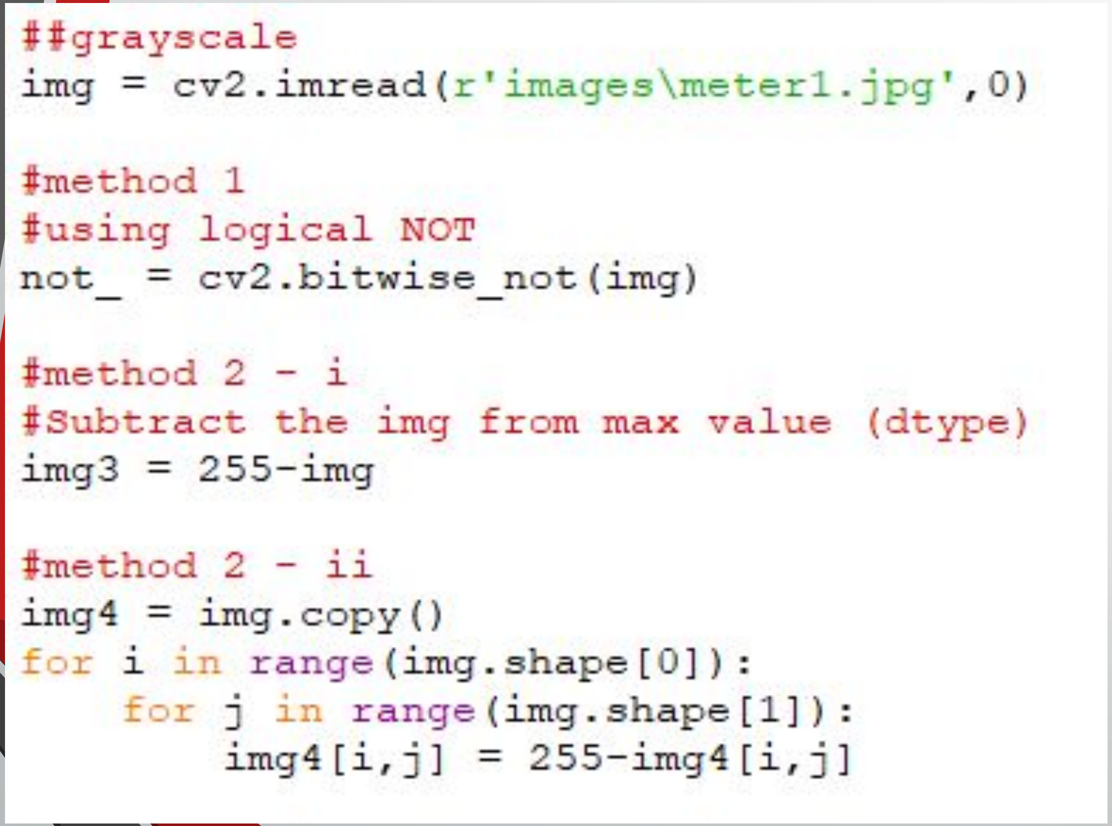
Basic Gray Level Transformations

Image Negative

$$T(r) = L - 1 - r$$

(L-1) □ max intensity value

r □ current pixel intensity value



```
##grayscale
img = cv2.imread(r'images\meter1.jpg',0)

#method 1
#using logical NOT
not_ = cv2.bitwise_not(img)

#method 2 - i
#Subtract the img from max value (dtype)
img3 = 255-img

#method 2 - ii
img4 = img.copy()
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        img4[i,j] = 255-img4[i,j]
```

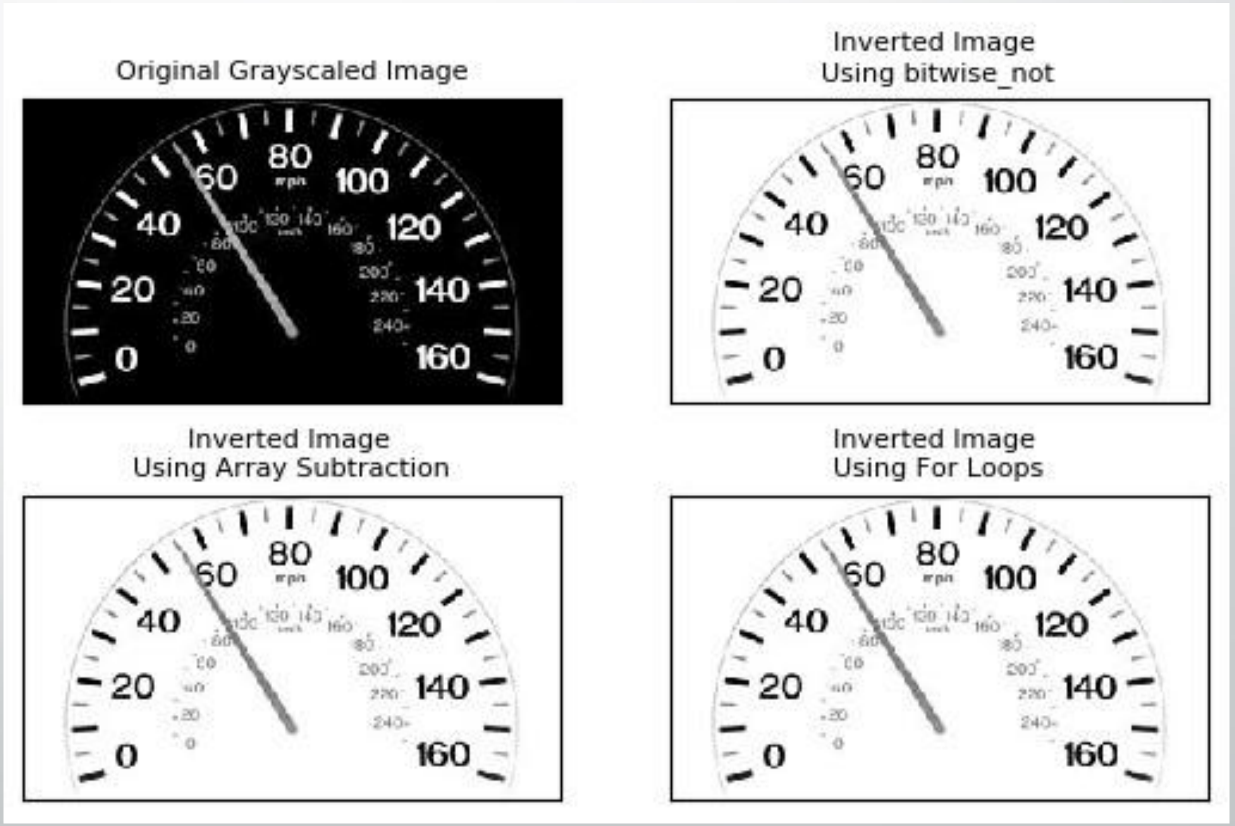
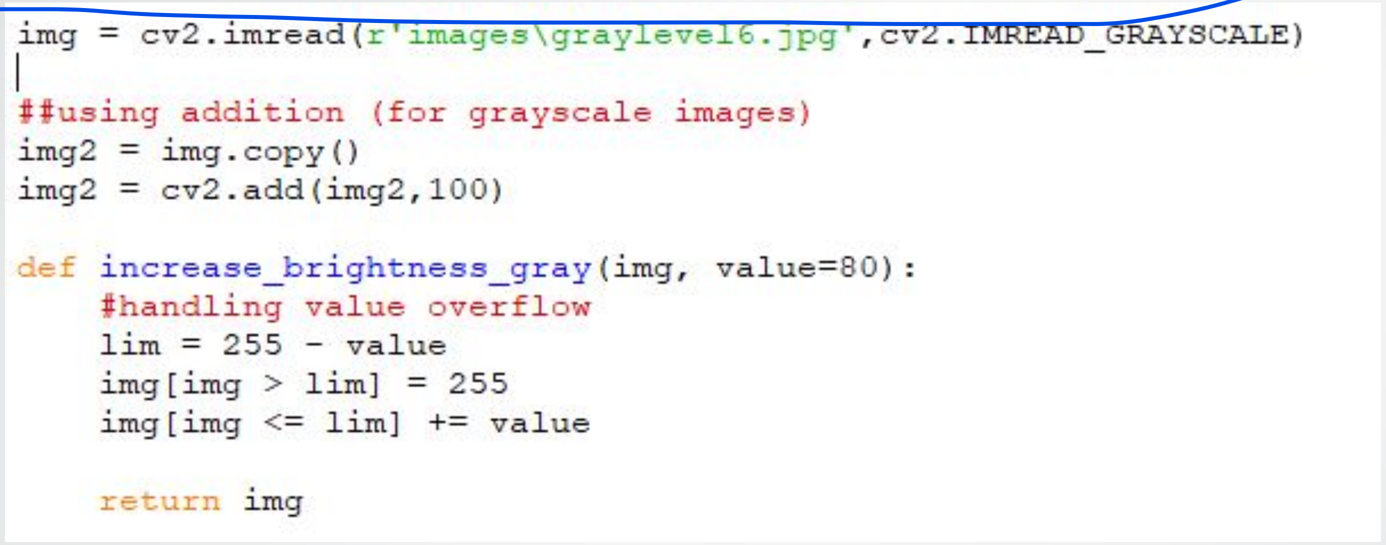

Original Grayscaled Image
Inverted Image Using bitwise_not
Inverted Image Using Array Subtraction
Inverted Image Using For Loops

Image Brightness – Grayscale Images

code



```
img = cv2.imread(r'images\graylevel6.jpg',cv2.IMREAD_GRAYSCALE)

##using addition (for grayscale images)
img2 = img.copy()
img2 = cv2.add(img2,100)

def increase_brightness_gray(img, value=80):
    #handling value overflow
    lim = 255 - value
    img[img > lim] = 255
    img[img <= lim] += value

    return img
```

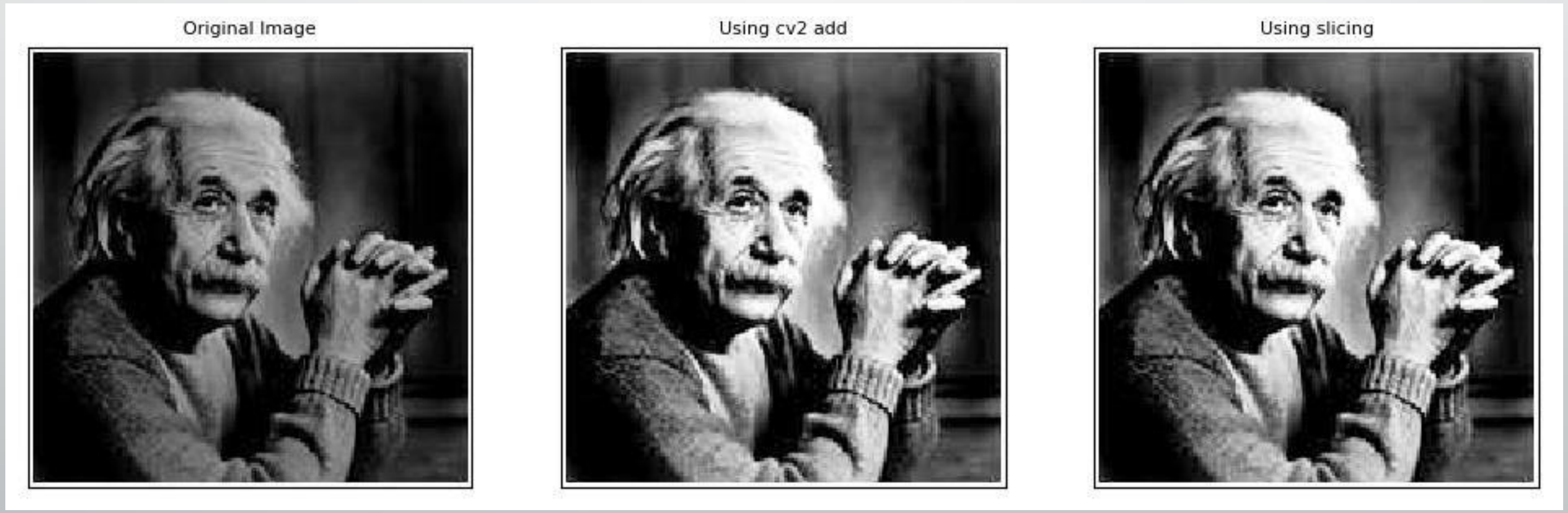
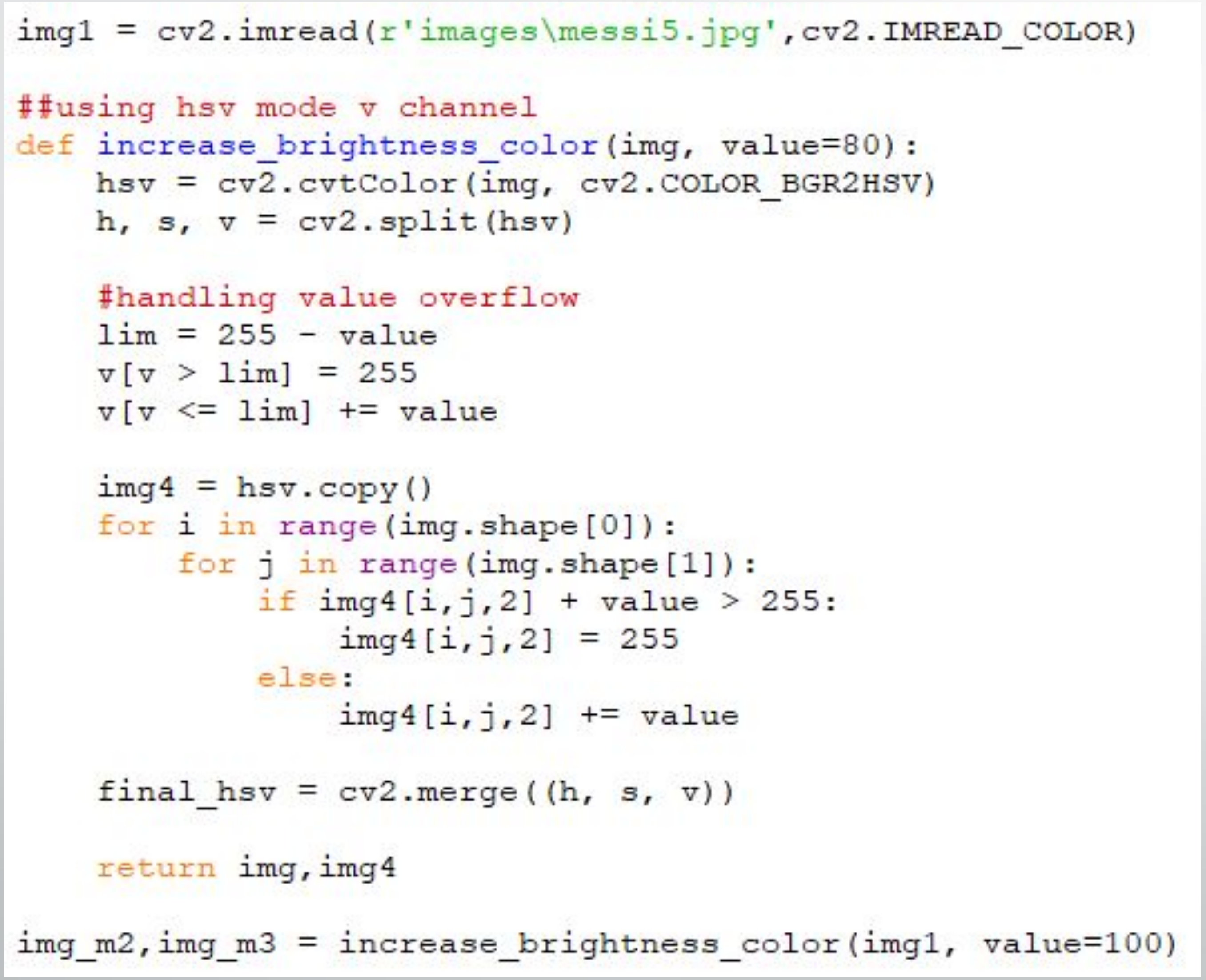
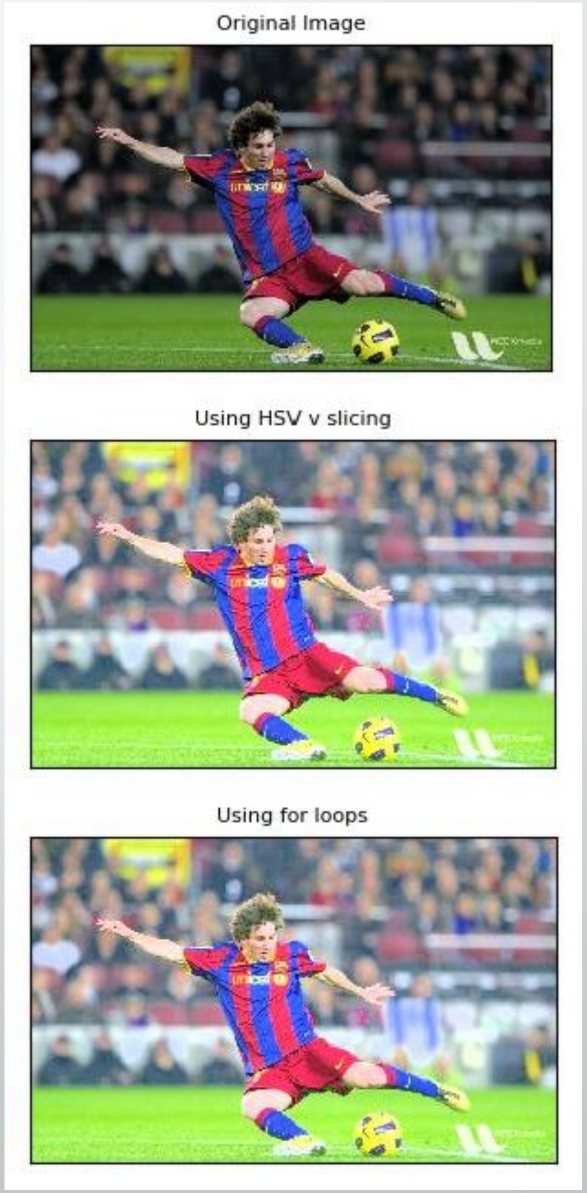


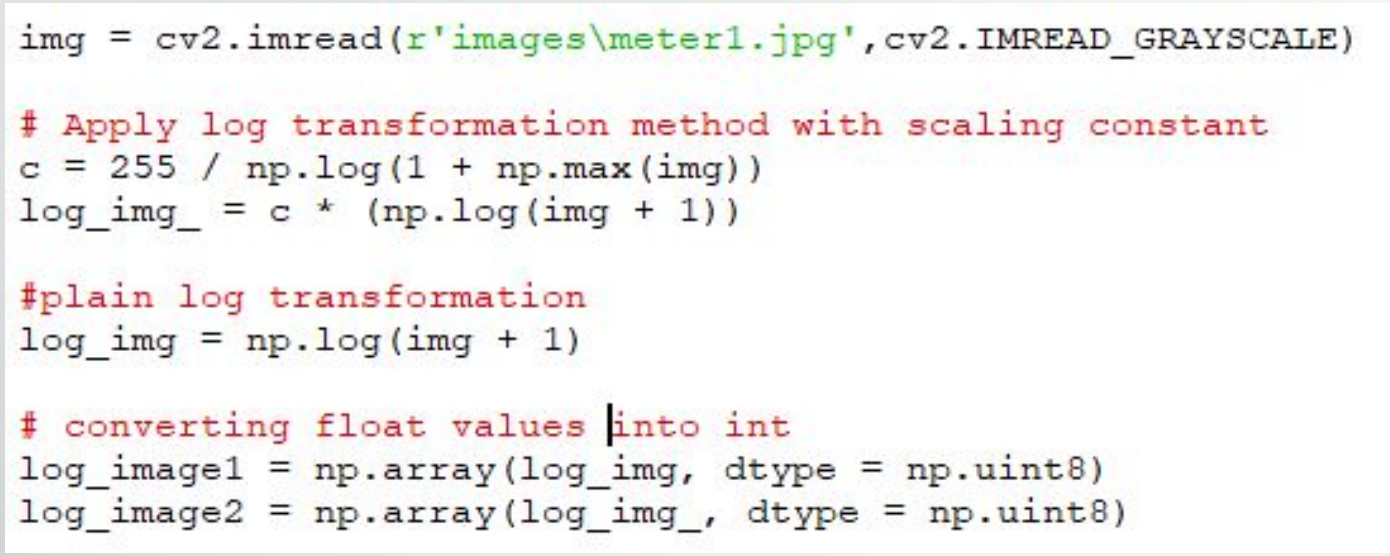
Image Brightness – Color Images



```
img1 = cv2.imread(r'images\messi5.jpg',cv2.IMREAD_COLOR)

##using hsv mode v channel
def increase_brightness_color(img, value=80):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    #handling value overflow
    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value

    img4 = hsv.copy()
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img4[i,j,2] + value > 255:
                img4[i,j,2] = 255
            else:
                img4[i,j,2] += value

    final_hsv = cv2.merge((h, s, v))

    return img,img4

img_m2,img_m3 = increase_brightness_color(img1, value=100)
```



Log Transformations



```
img = cv2.imread(r'images\meter1.jpg',cv2.IMREAD_GRAYSCALE)

# Apply log transformation method with scaling constant
c = 255 / np.log(1 + np.max(img))
log_img_ = c * (np.log(img + 1))

#plain log transformation
log_img = np.log(img + 1)

# converting float values into int
log_image1 = np.array(log_img, dtype = np.uint8)
log_image2 = np.array(log_img_, dtype = np.uint8)
```

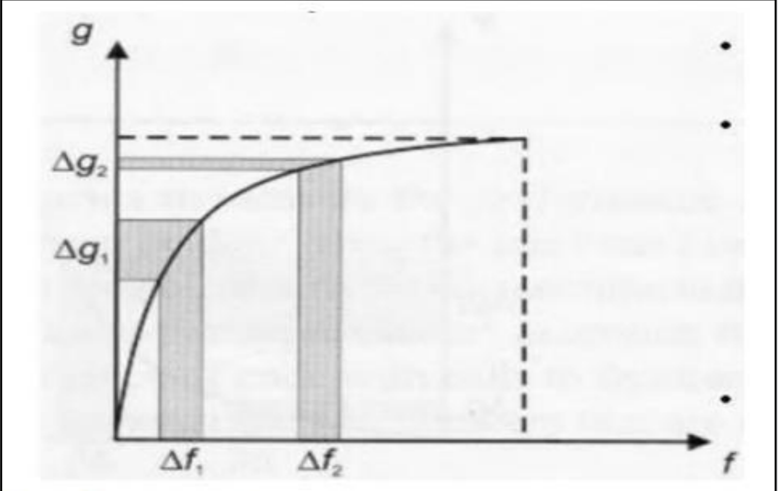
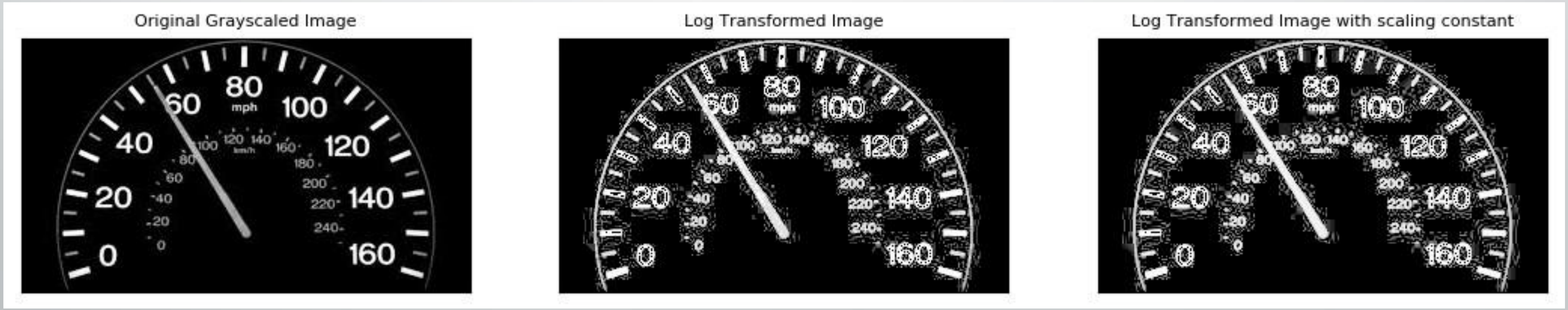
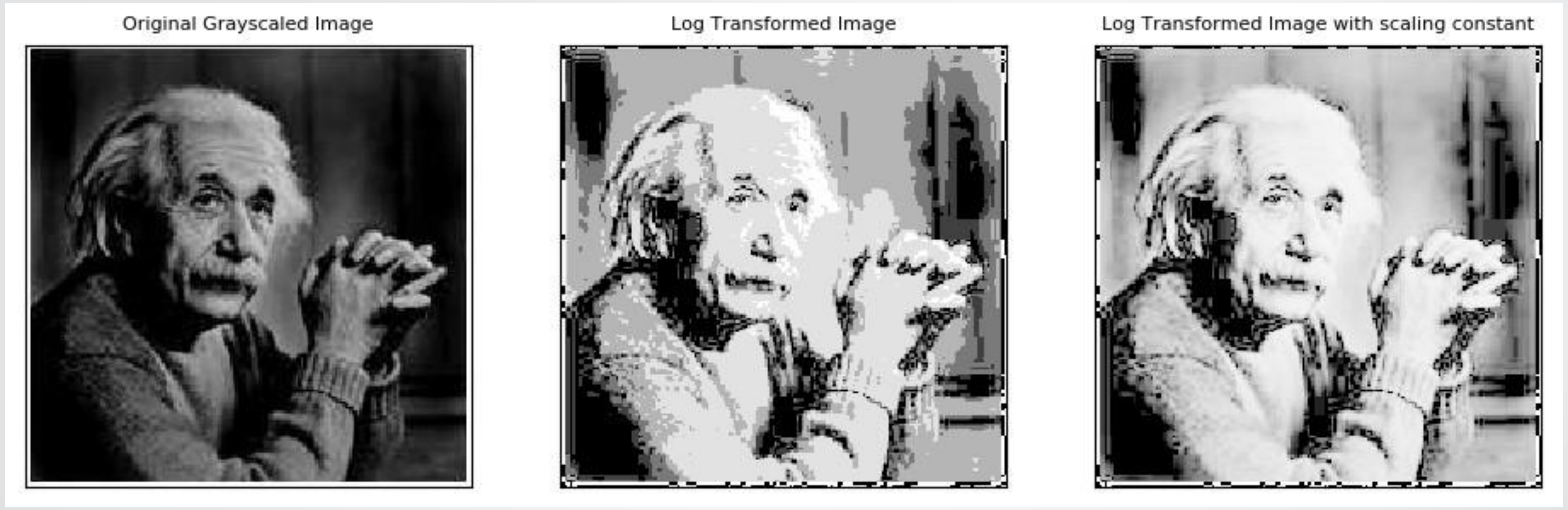


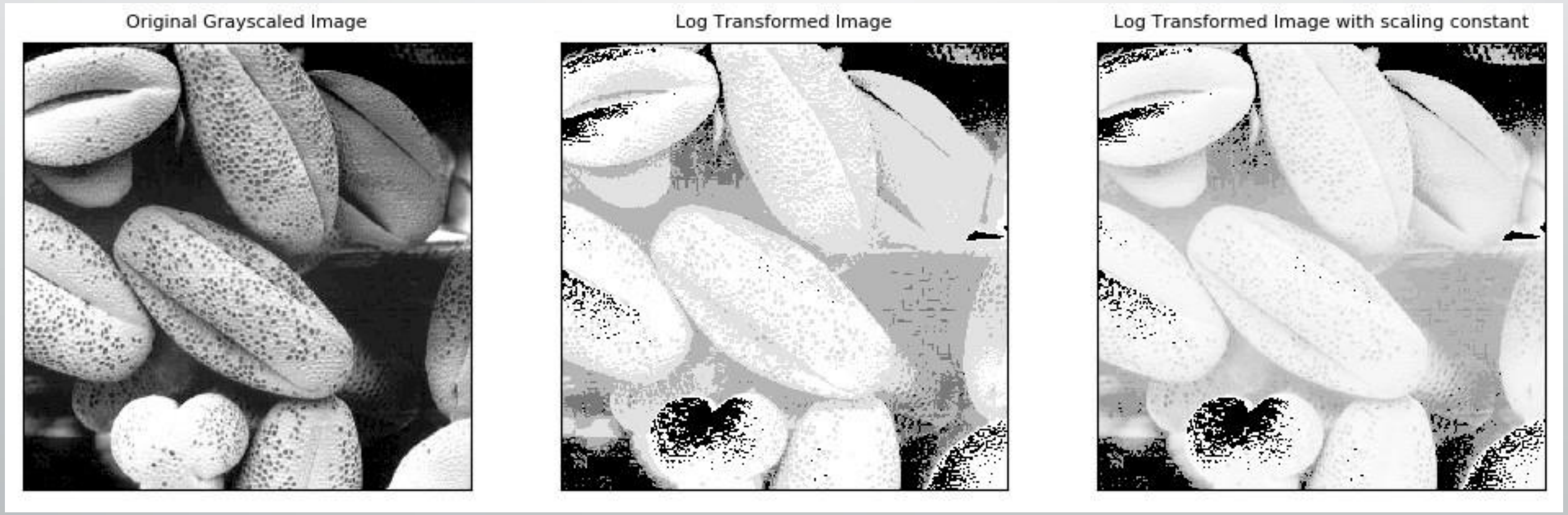
Fig. log transformation curve input vs output


Original Grayscaled Image
Log Transformed Image
Log Transformed Image with scaling constant

Ref.
https://www.geeksforgeeks.org/log-transformation-of-an-image-using-python-and-opencv/

Log Transformations - Examples

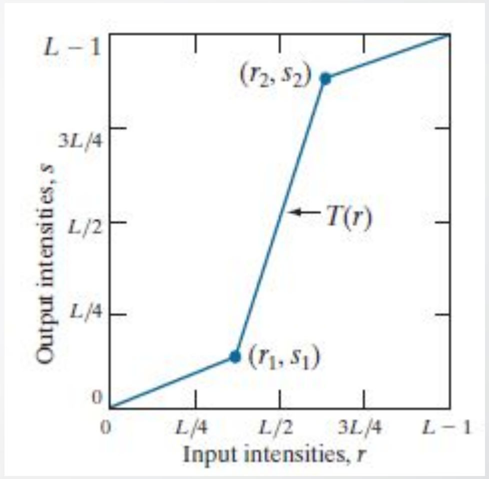

Original Grayscaled Image
Log Transformed Image
Log Transformed Image with scaling constant


Original Grayscaled Image
Log Transformed Image
Log Transformed Image with scaling constant

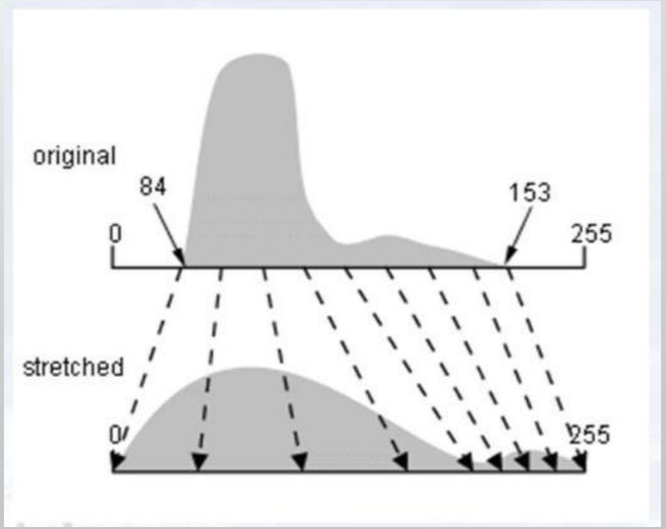
Piecewise Linear Transformation Functions

Contrast Stretching / Normalization

$$P_{out} = (P_{in} - c)\left(\frac{b-a}{d-c}\right) + a$$

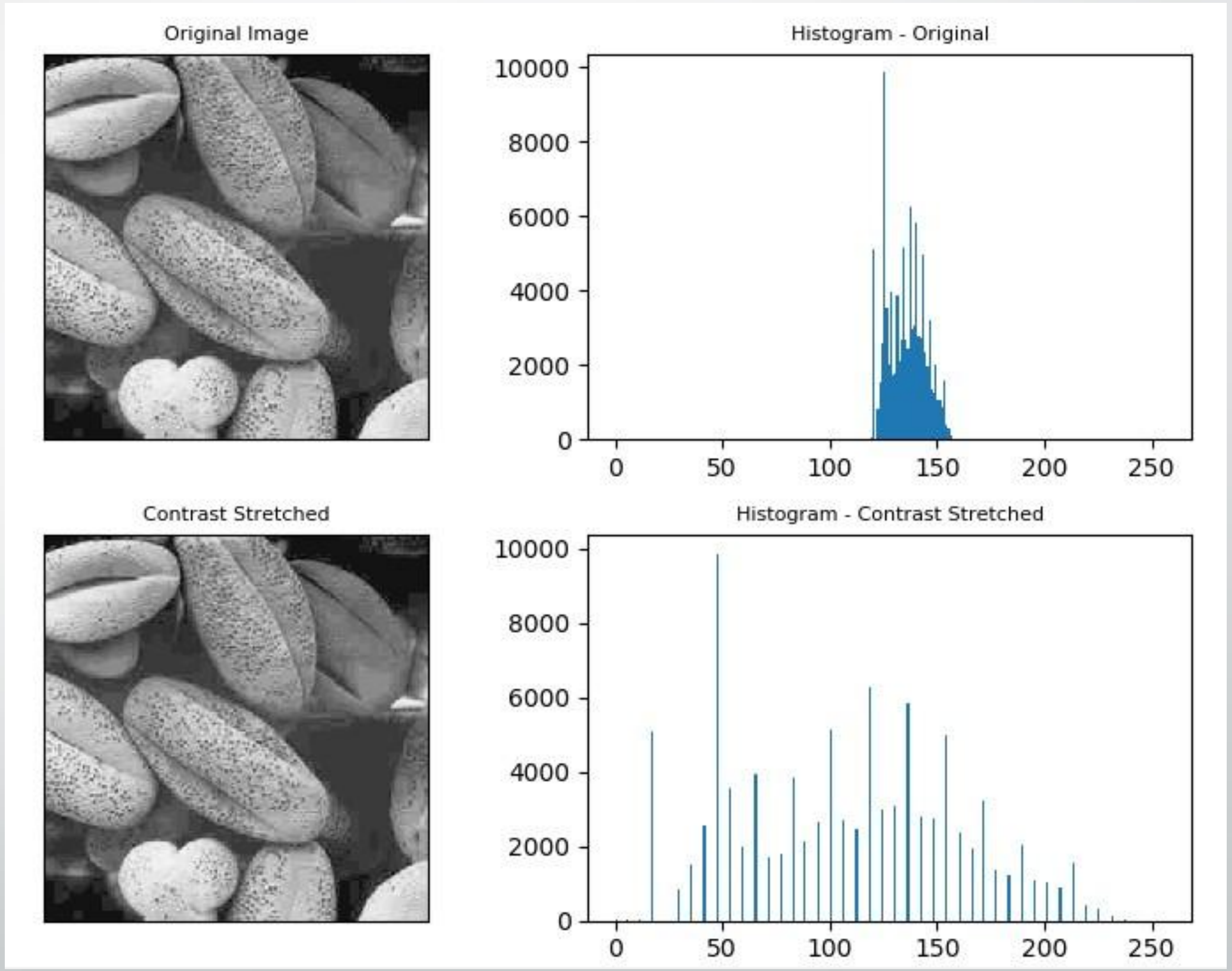


- P_{out} = new pixel value
- P_{in} = Input image Pixel Value
- a = lower limit for the data type
- b = upper limit for the data type
- c = min pixel value for the input image
- d = max pixel value for the input image

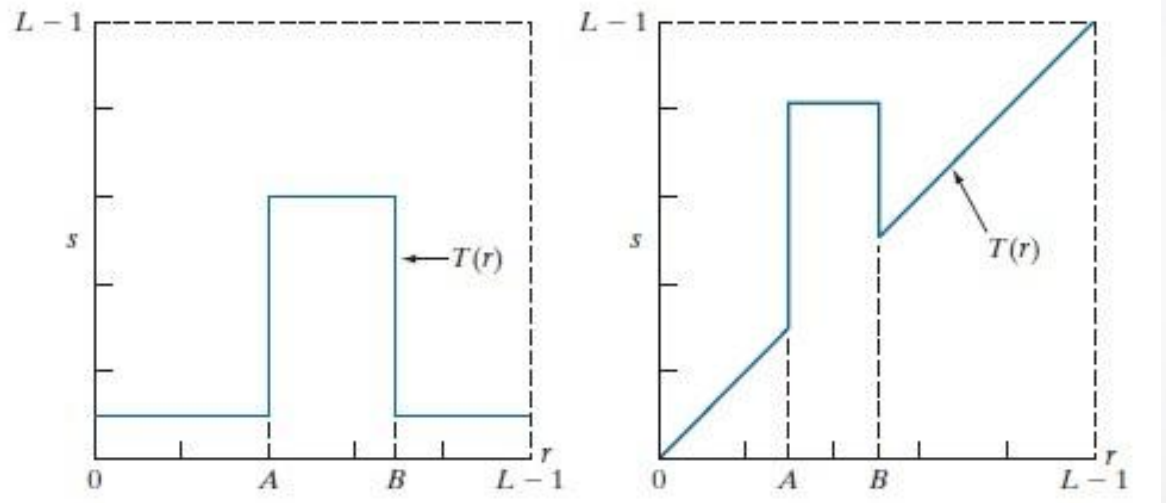


Contrast Stretching / Normalization

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r'images\contrast_str.png',
                 cv2.IMREAD_GRAYSCALE)

img__ = img.copy()

plt.figure()
plt.title("Original Grayscale Image Histogram")
plt.hist(img.ravel(),bins=256,range=[0,256])
plt.show()

input_max = np.max(img)  #125
input_min = np.min(img)   #50

output_max = 255 # np.iinfo('uint8').max
output_min = 0 ## np.iinfo('uint8').min

out_img_ = (img-input_min) *(
              (output_max-output_min)/(
                 input_max-input_min))+output_min
out_img_ = np.array(out_img_, dtype = np.uint8)
```

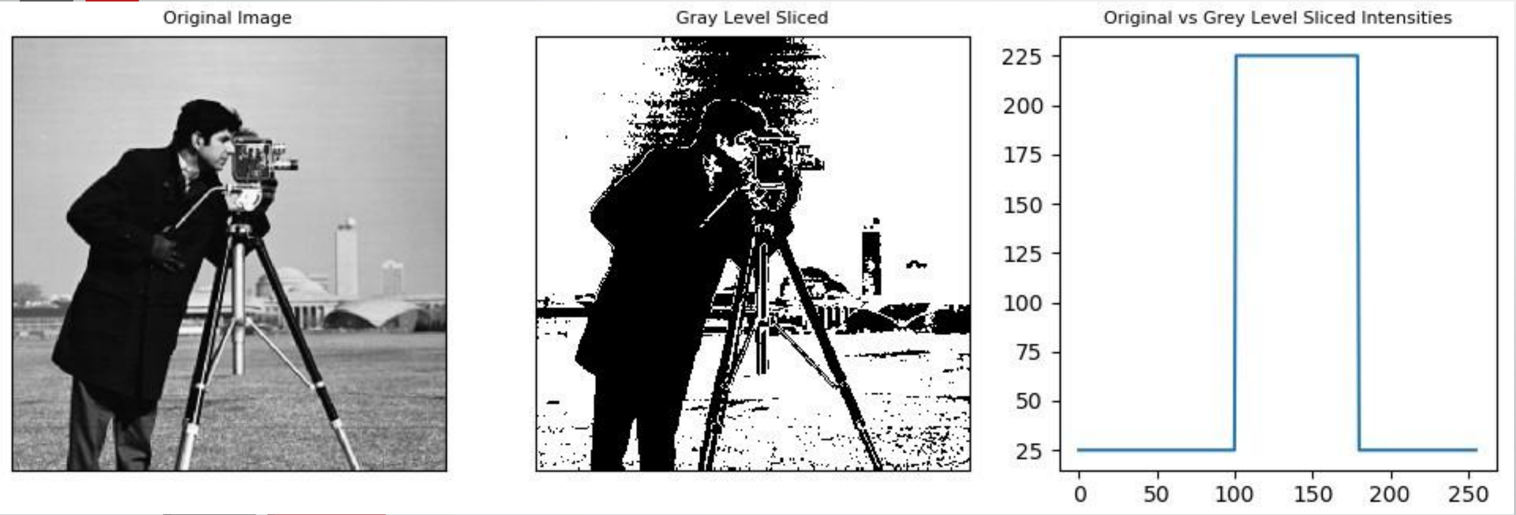


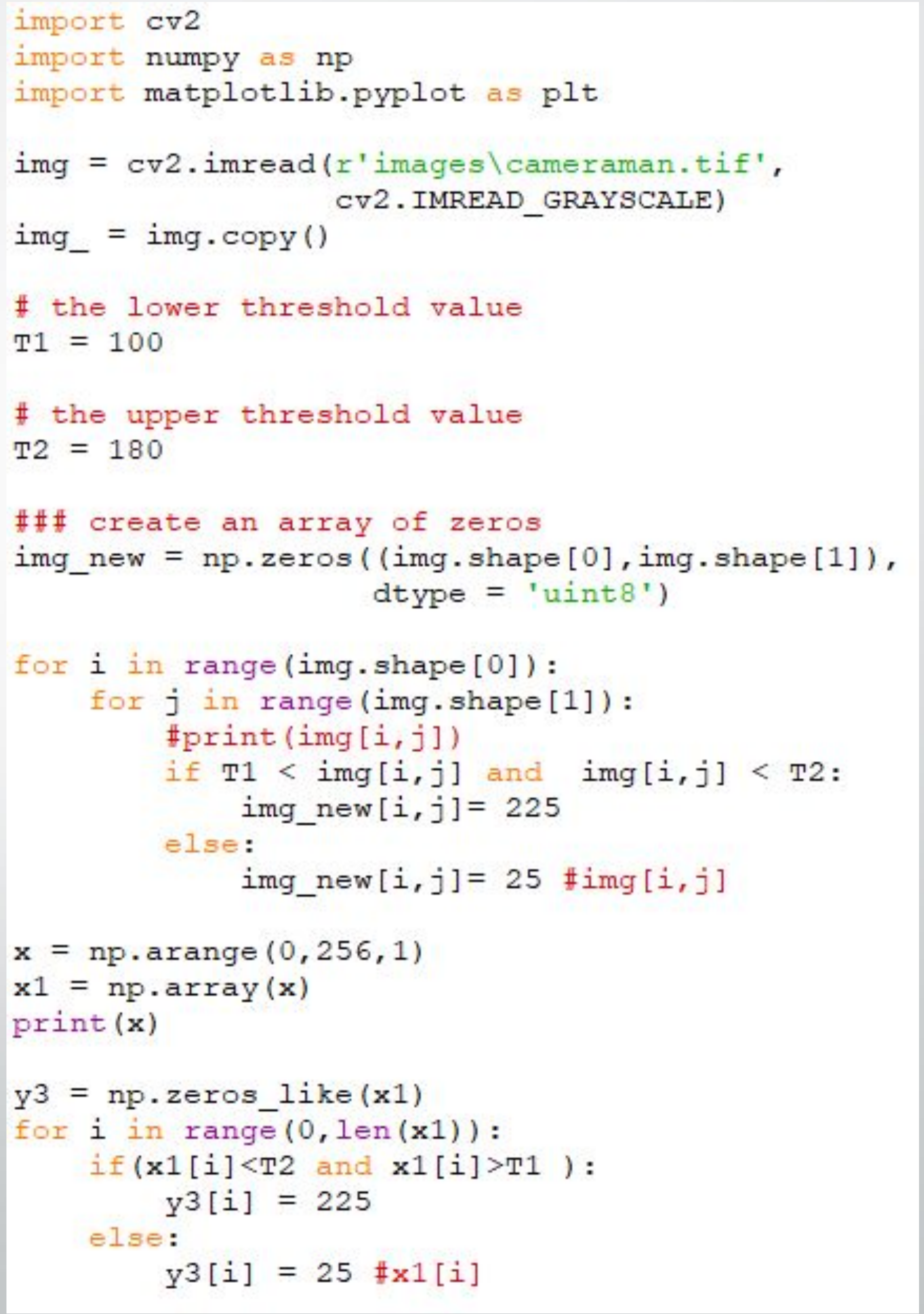
Gray-Level Slicing



a b

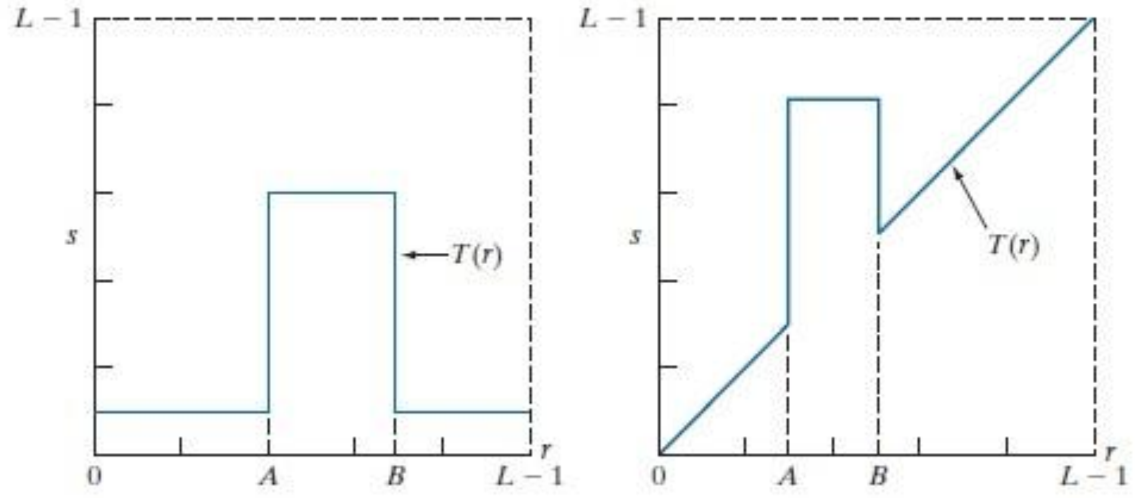
FIGURE 3.11 (a) This transformation function highlights range $[A, B]$ and reduces all other intensities to a lower level. (b) This function highlights range $[A, B]$ and leaves other intensities unchanged.





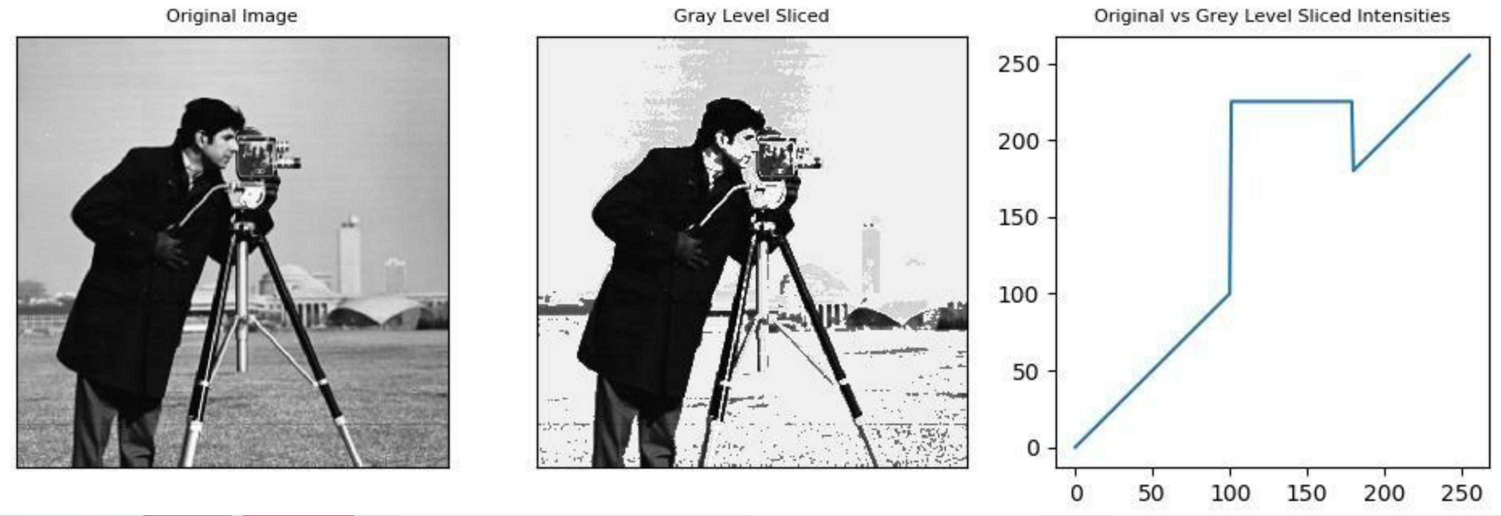
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r'images\cameraman.tif',
                 cv2.IMREAD_GRAYSCALE)
img_ = img.copy()

# the lower threshold value
T1 = 100

# the upper threshold value
T2 = 180

### create an array of zeros
img_new = np.zeros((img.shape[0],img.shape[1]),
                   dtype = 'uint8')

for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        #print(img[i,j])
        if T1 < img[i,j] and  img[i,j] < T2:
            img_new[i,j]= 225
        else:
            img_new[i,j]= 25 #img[i,j]

x = np.arange(0,256,1)
x1 = np.array(x)
print(x)

y3 = np.zeros_like(x1)
for i in range(0,len(x1)):
    if(x1[i]<T2 and x1[i]>T1 ):
        y3[i] = 225
    else:
        y3[i] = 25 #x1[i]
```

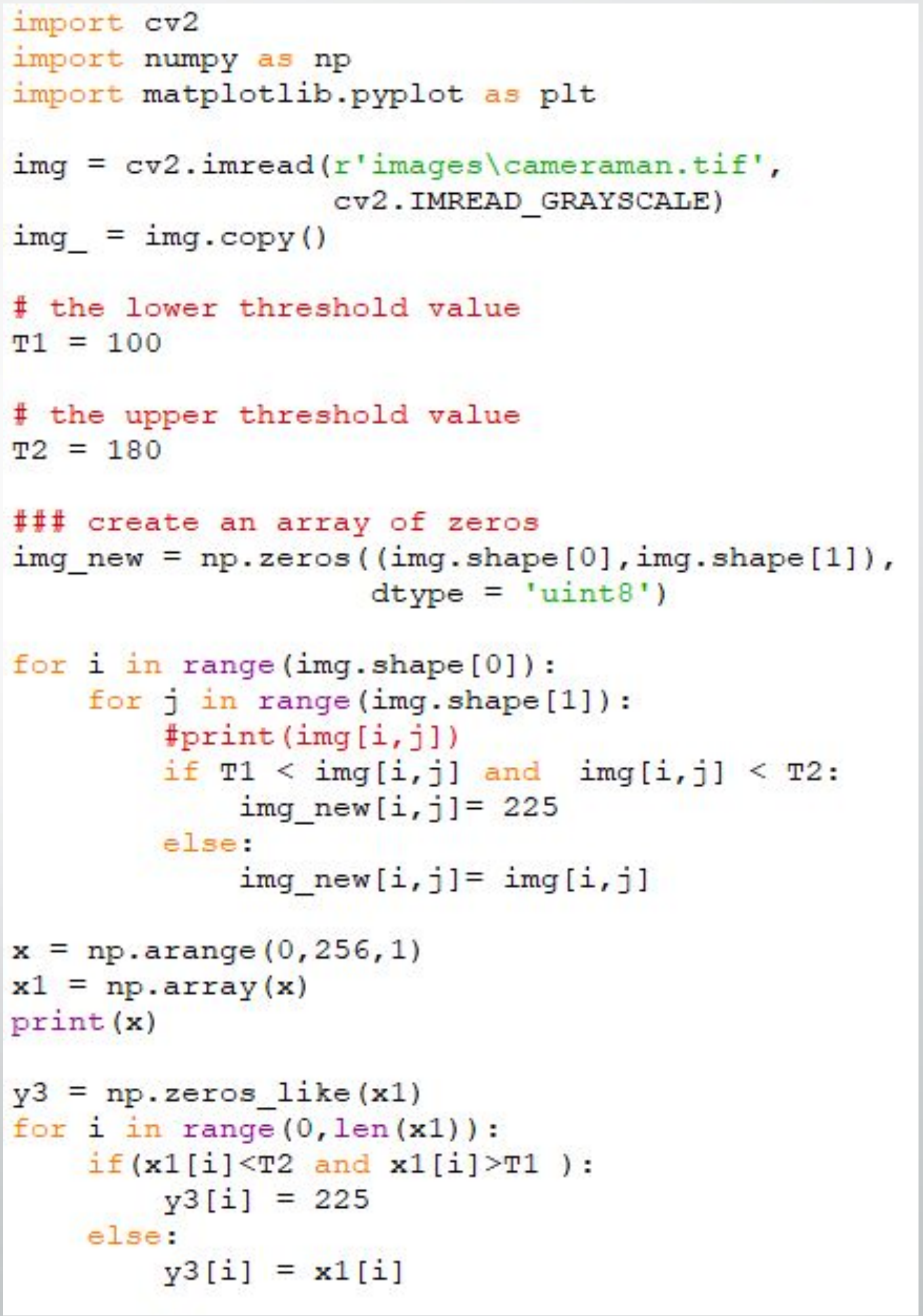
Gray-Level Slicing



a b

FIGURE 3.11 (a) This transformation function highlights range $[A, B]$ and reduces all other intensities to a lower level. (b) This function highlights range $[A, B]$ and leaves other intensities unchanged.





```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r'images\cameraman.tif',
                 cv2.IMREAD_GRAYSCALE)
img_ = img.copy()

# the lower threshold value
T1 = 100

# the upper threshold value
T2 = 180

### create an array of zeros
img_new = np.zeros((img.shape[0],img.shape[1]),
                   dtype = 'uint8')

for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        #print(img[i,j])
        if T1 < img[i,j] and  img[i,j] < T2:
            img_new[i,j]= 225
        else:
            img_new[i,j]= img[i,j]

x = np.arange(0,256,1)
x1 = np.array(x)
print(x)

y3 = np.zeros_like(x1)
for i in range(0,len(x1)):
    if(x1[i]<T2 and x1[i]>T1 ):
        y3[i] = 225
    else:
        y3[i] = x1[i]
```

Bit Plane Slicing

https://theailearner.com/2019/01/25/bit-plane-slicing/

Thresholding

Thresholding / Binarization

The simplest form of segmenting images – background / foreground

Types of Thresholding

- Global Thresholding / Simple Thresholding

  Uses a global threshold value

- Adaptive Thresholding

  The algorithm will find a local threshold value for each area
  (better when considering images with inconsistent illumination)

- Otzu's Thresholding

  Optimized adaptive thresholding

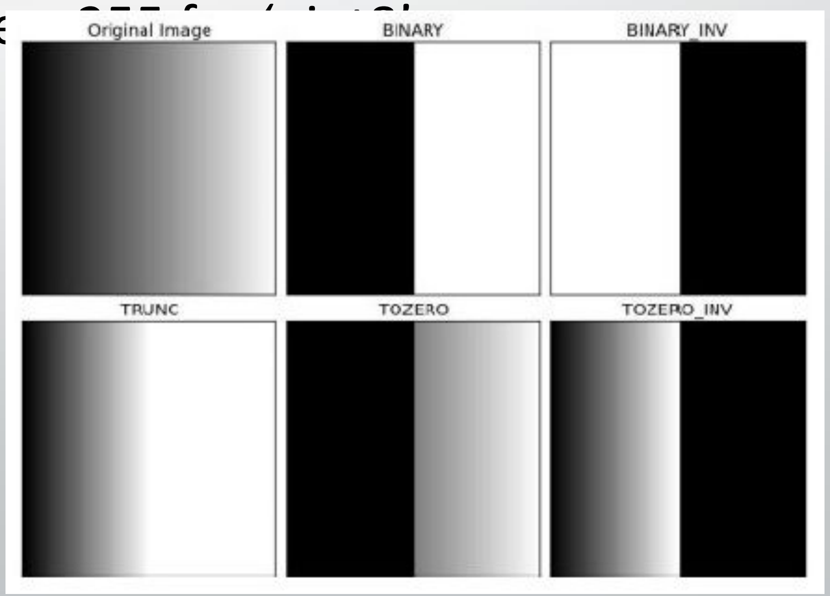
Global Thresholding

Syntax:

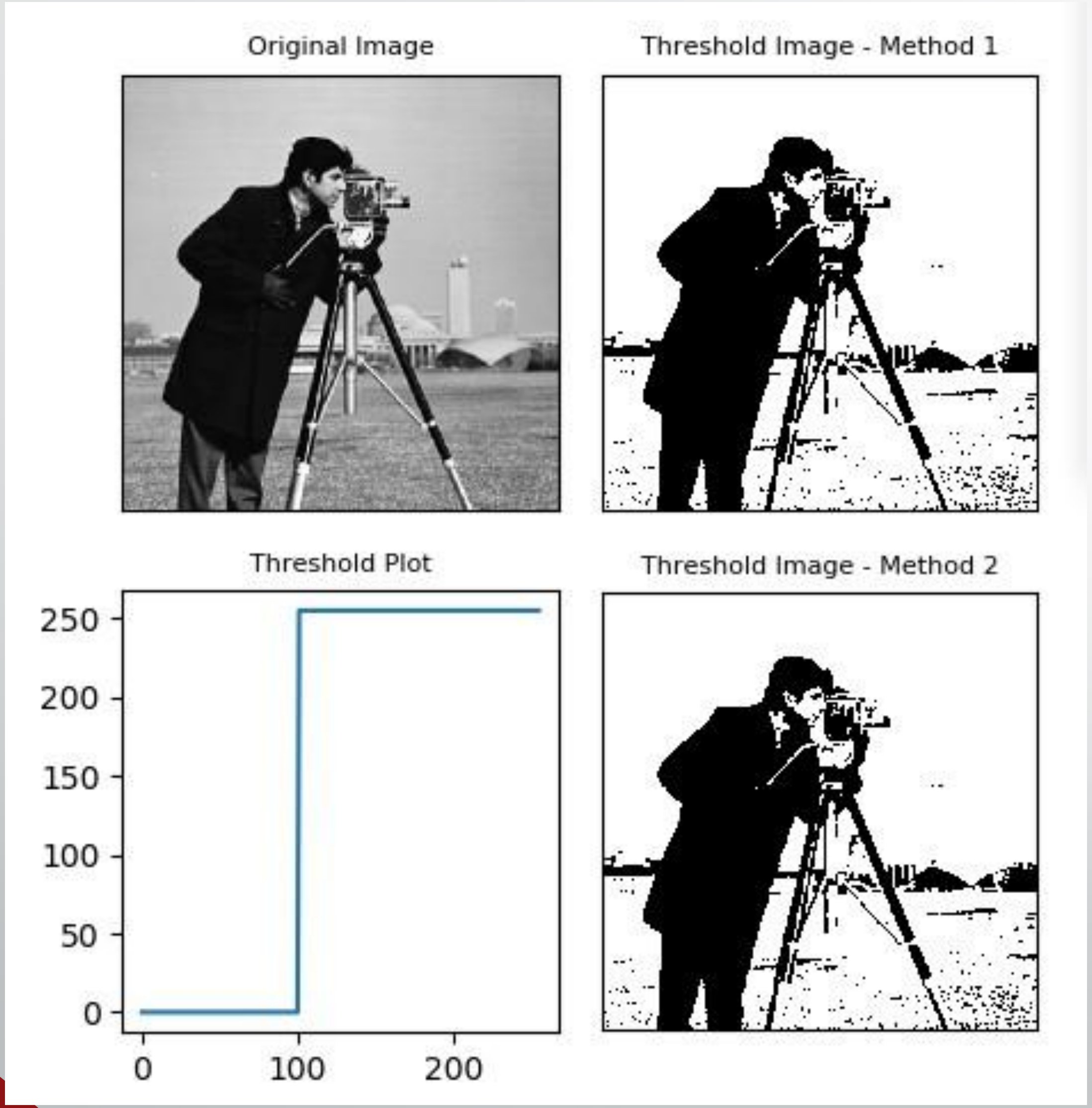
```
ret,thresh = cv2.threshold(img, thr_val, max_val, thresh_mode)
```

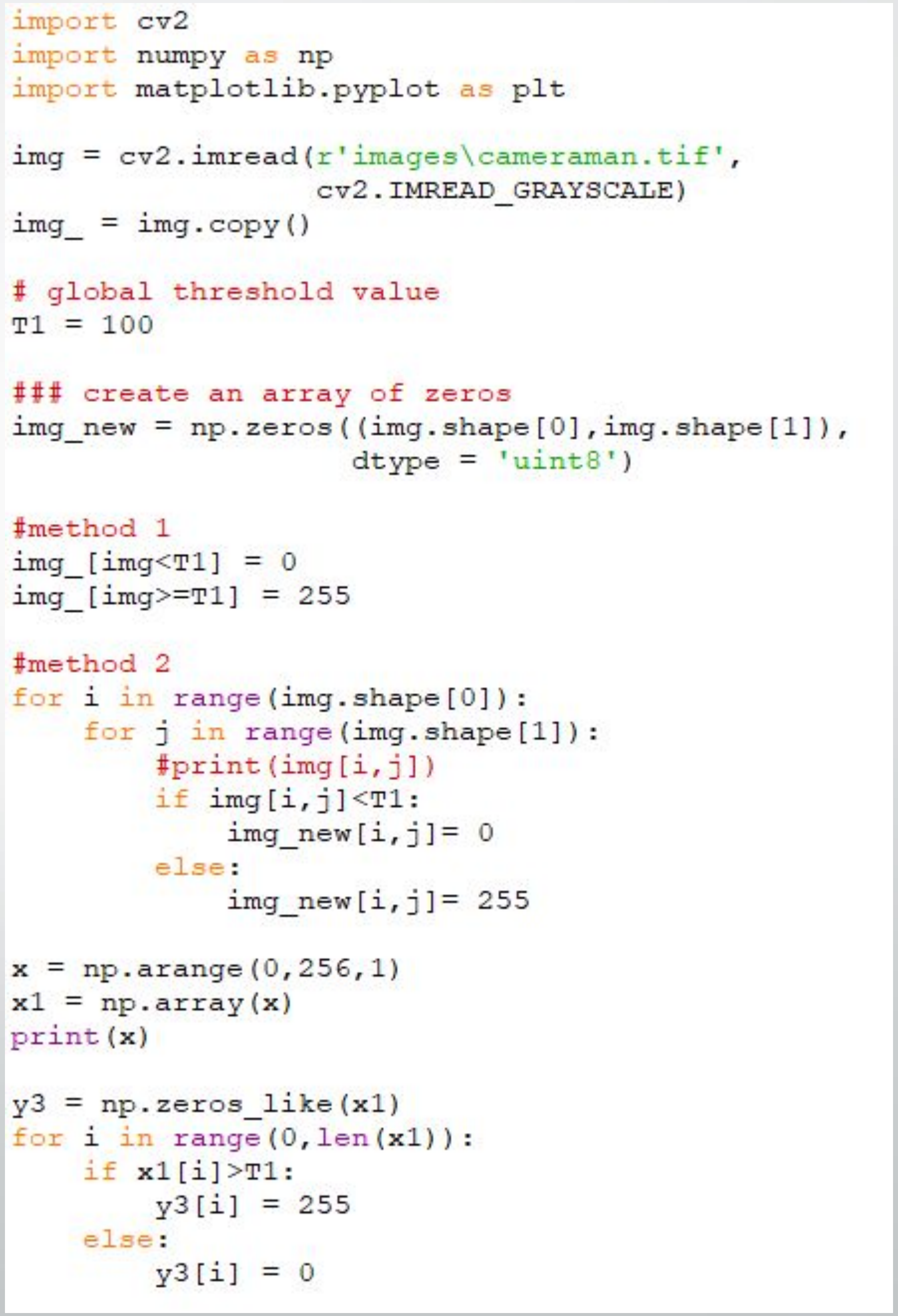
img	: image array
thr_val	: global threshold value
thresh_mode	: is a flag which specifies the global threshold mode
max_val	: maximum intensity value images

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV



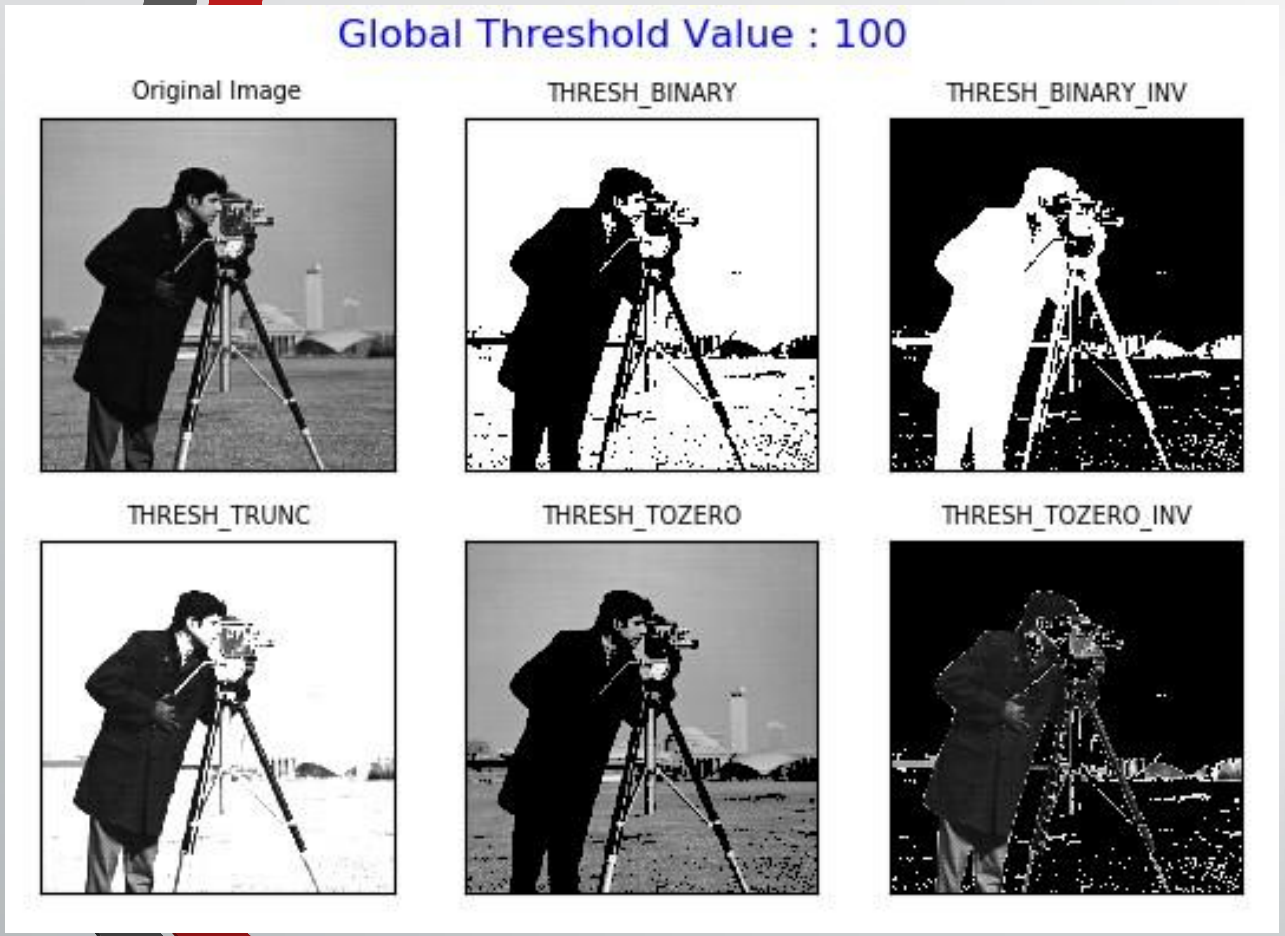
Global Thresholding





```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r'images\cameraman.tif',
                 cv2.IMREAD_GRAYSCALE)
img_ = img.copy()

# global threshold value
T1 = 100

### create an array of zeros
img_new = np.zeros((img.shape[0],img.shape[1]),
                   dtype = 'uint8')

#method 1
img_[img<T1] = 0
img_[img>=T1] = 255

#method 2
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        #print(img[i,j])
        if img[i,j]<T1:
            img_new[i,j]= 0
        else:
            img_new[i,j]= 255

x = np.arange(0,256,1)
x1 = np.array(x)
print(x)

y3 = np.zeros_like(x1)
for i in range(0,len(x1)):
    if x1[i]>T1:
        y3[i] = 255
    else:
        y3[i] = 0
```

Global Thresholding



```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r'images\cameraman.tif',cv2.IMREAD_GRAYSCALE)

# global threshold value
T1 = 100

ret,thresh1 = cv2.threshold(img,T1,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,T1,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,T1,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,T1,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,T1,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','THRESH_BINARY','THRESH_BINARY_INV',
          'THRESH_TRUNC','THRESH_TOZERO','THRESH_TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

plt.suptitle("Global Threshold Value : 100",color='blue')
for i in range(6):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i],'gray')
    plt.title(titles[i],fontsize=7)
    plt.xticks([]),plt.yticks([])

plt.show()
```

Adaptive Thresholding

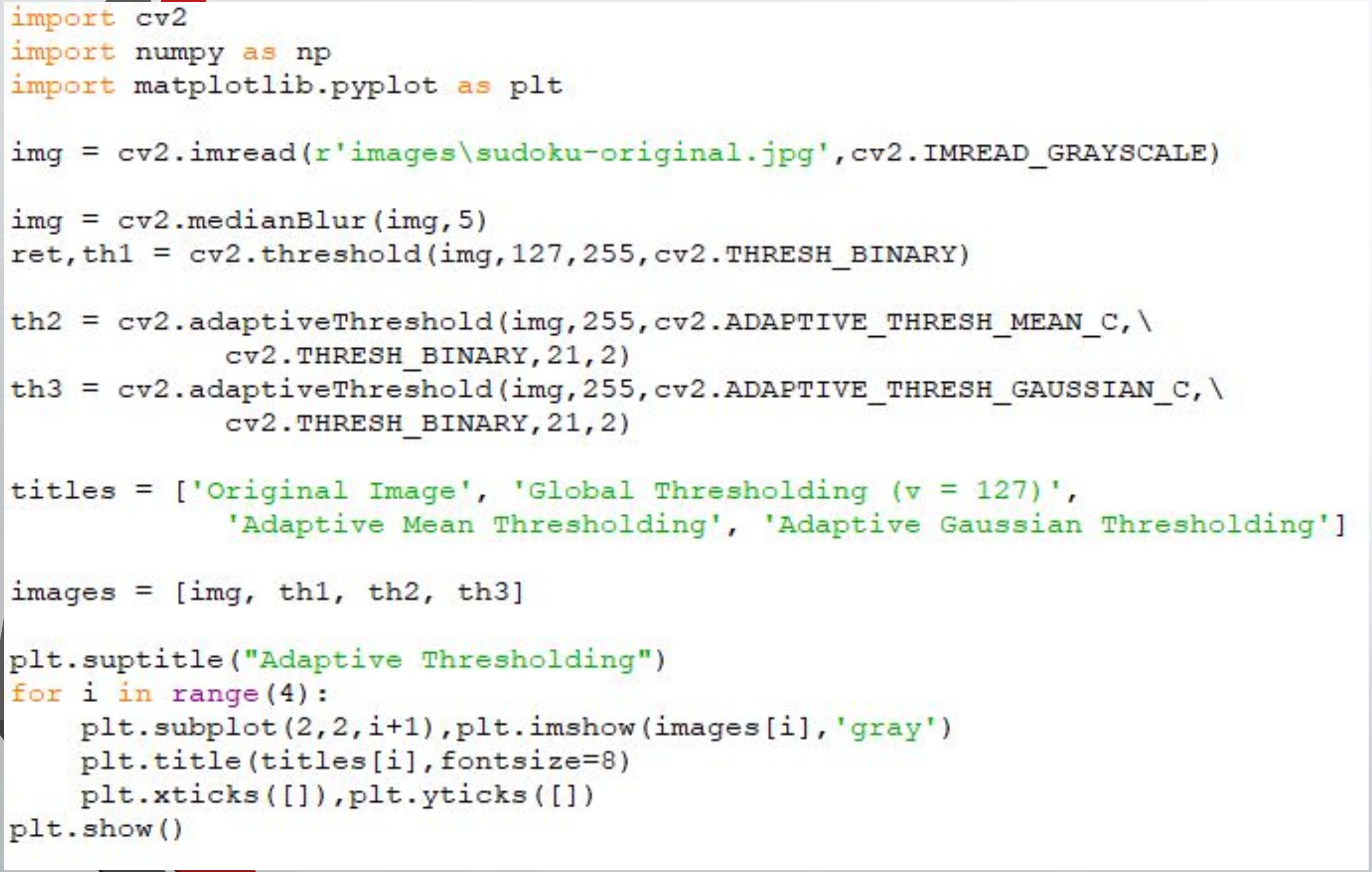
Syntax:

```
thr = cv2.adaptiveThreshold(img, max, adpt_m, thr_t, blk, C)
```

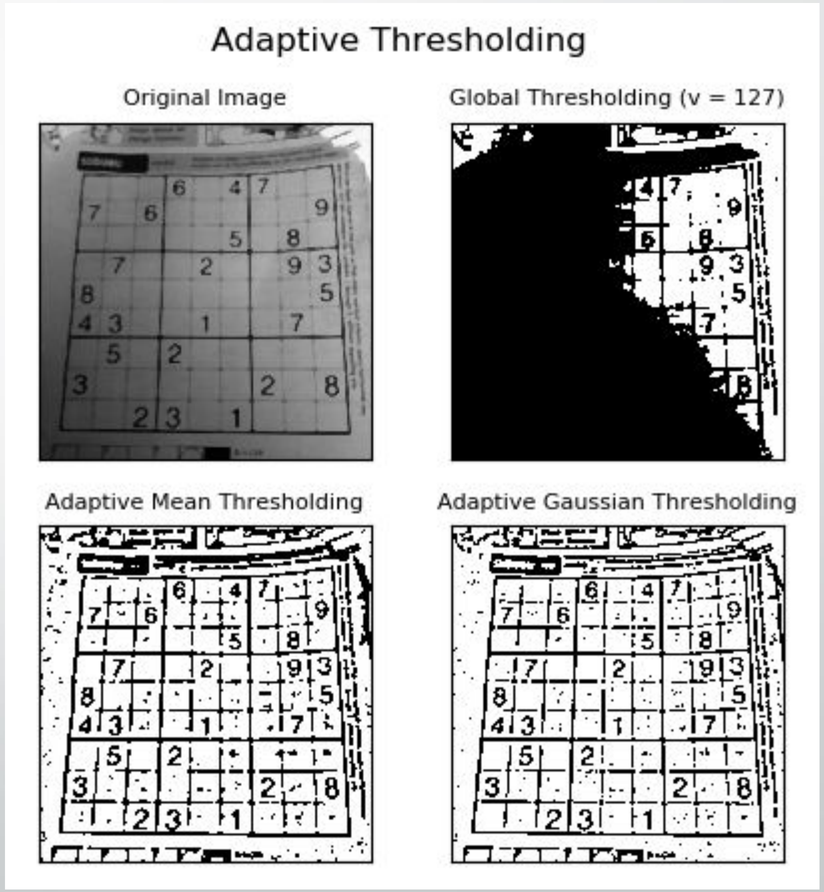
img	: image array
max	: maximum intensity value – 255 for 'uint8' images
thr_t	: cv2 threshold type (**cv2.THRESH_BINARY | cv2.THRESH_BINARY_INV**)
blk	: block size (size of the neighborhood area)
C	: constant subtracted from mean or weighted sum of the neighborhood pixels
thr	: threshold image
adpt_m	: adaptive method

- **cv2.ADAPTIVE_THRESH_MEAN_C** : The threshold value is the mean of the neighborhood area minus the constant C.
- **cv2.ADAPTIVE_THRESH_GAUSSIAN_C**: The threshold value is a Gaussian-weighted sum of the neighborhood values minus the constant C

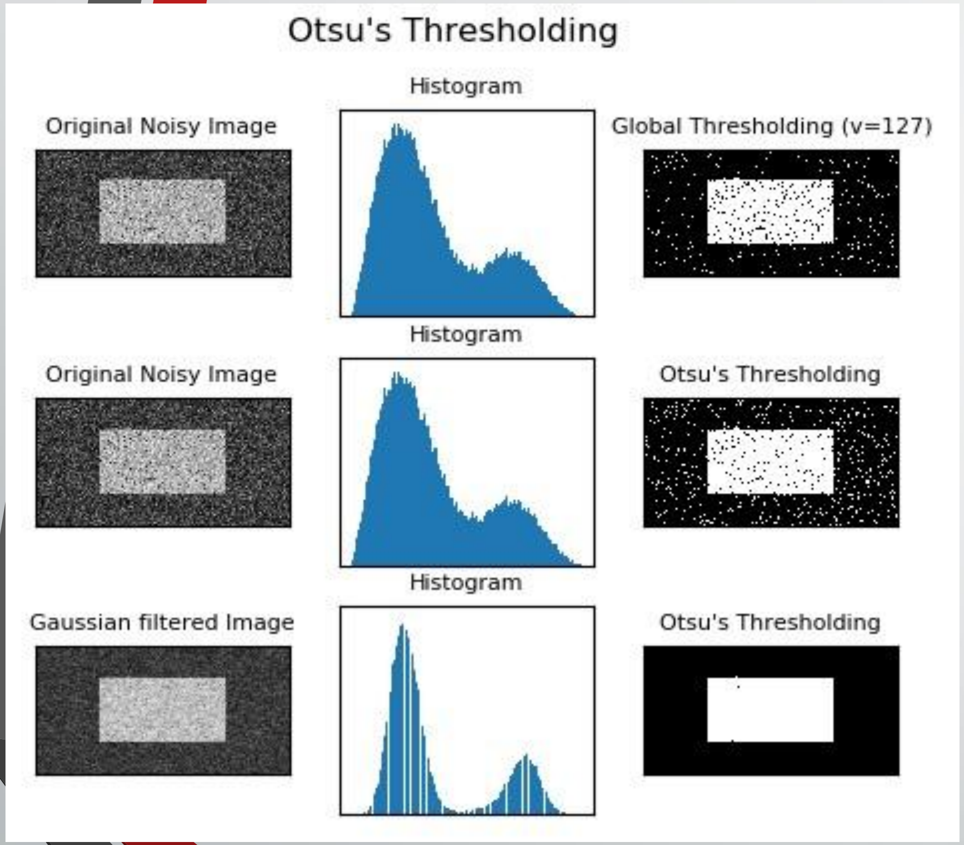
Adaptive Thresholding

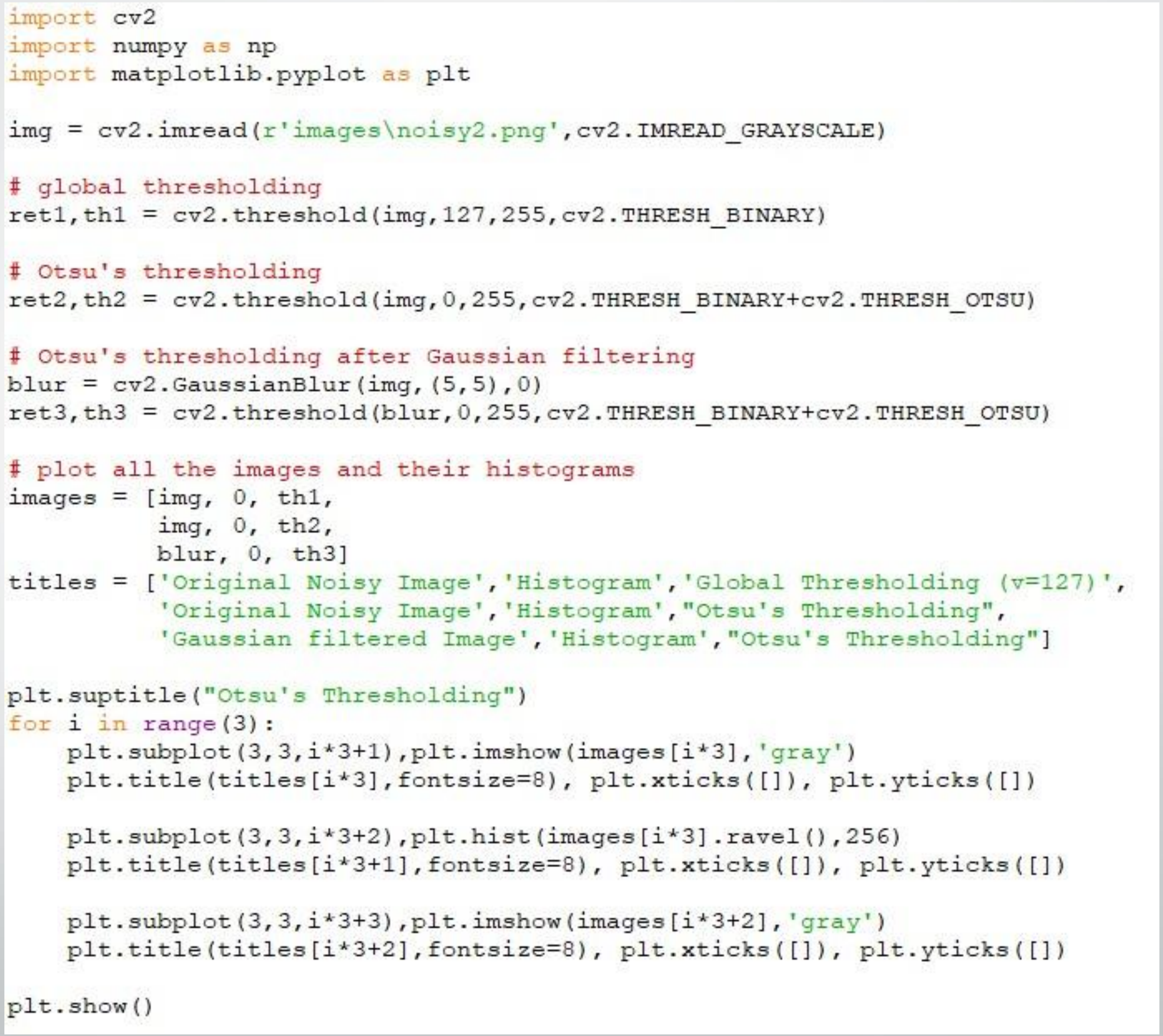


```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r'images\sudoku-original.jpg',cv2.IMREAD_GRAYSCALE)

img = cv2.medianBlur(img,5)
ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
            cv2.THRESH_BINARY,21,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,21,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']

images = [img, th1, th2, th3]

plt.suptitle("Adaptive Thresholding")
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i],fontsize=8)
    plt.xticks([]),plt.yticks([])
plt.show()
```


Adaptive Thresholding
Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

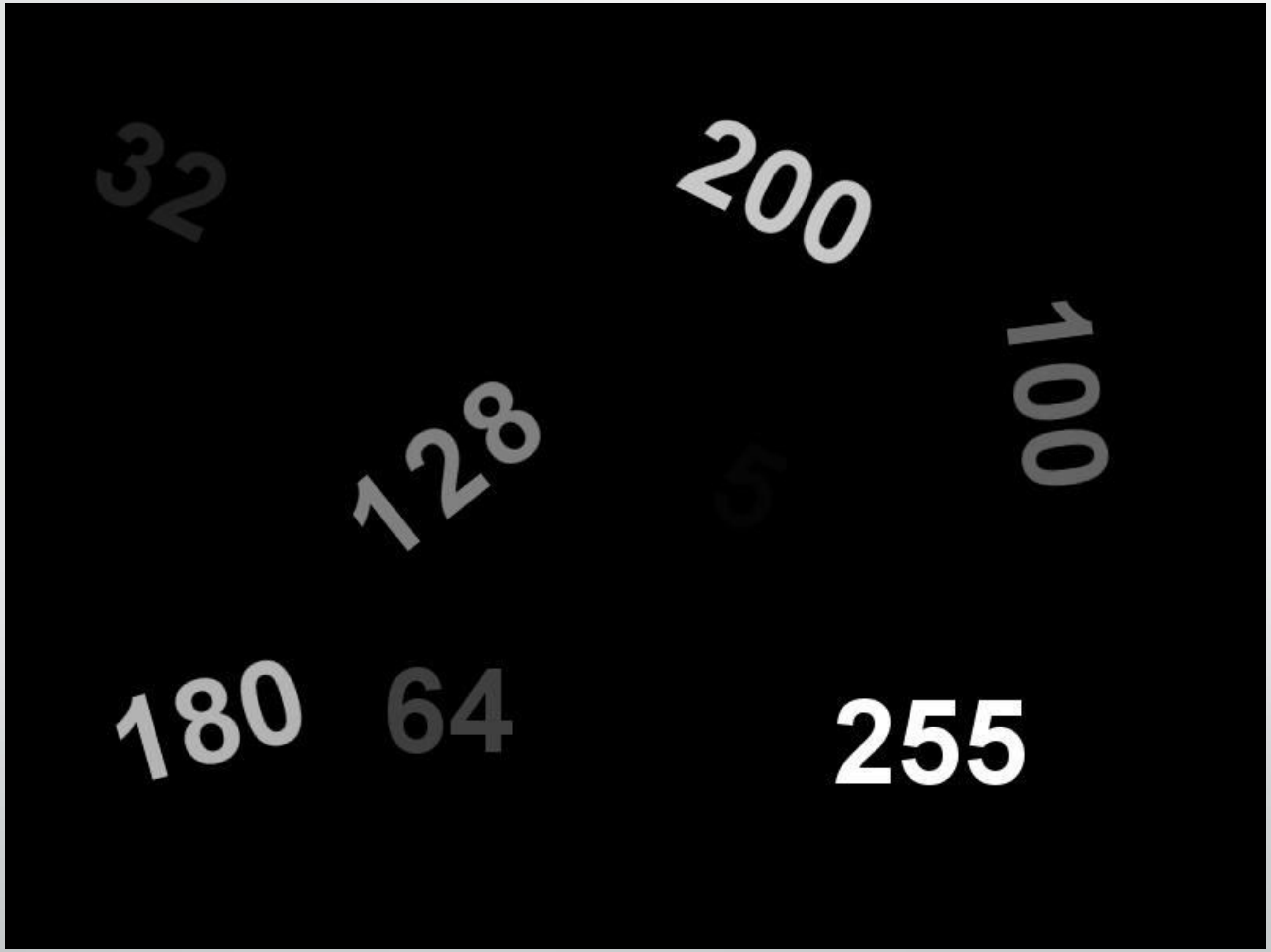
Otzu's Thresholding



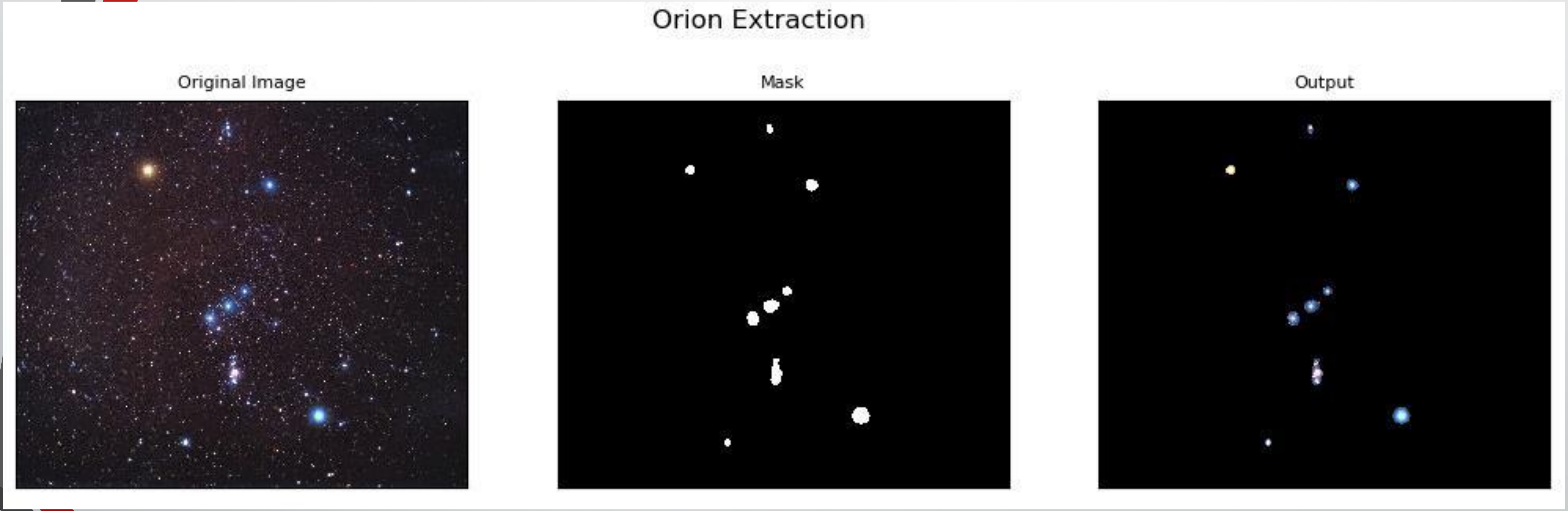


```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r'images\noisy2.png',cv2.IMREAD_GRAYSCALE)

# global thresholding
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv2.GaussianBlur(img,(5,5),0)
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

plt.suptitle("Otsu's Thresholding")
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3],fontsize=8), plt.xticks([]), plt.yticks([])

    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1],fontsize=8), plt.xticks([]), plt.yticks([])

    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2],fontsize=8), plt.xticks([]), plt.yticks([])

plt.show()
```

Numbers Extraction



Orion Extraction



– END –