






CSC 3141

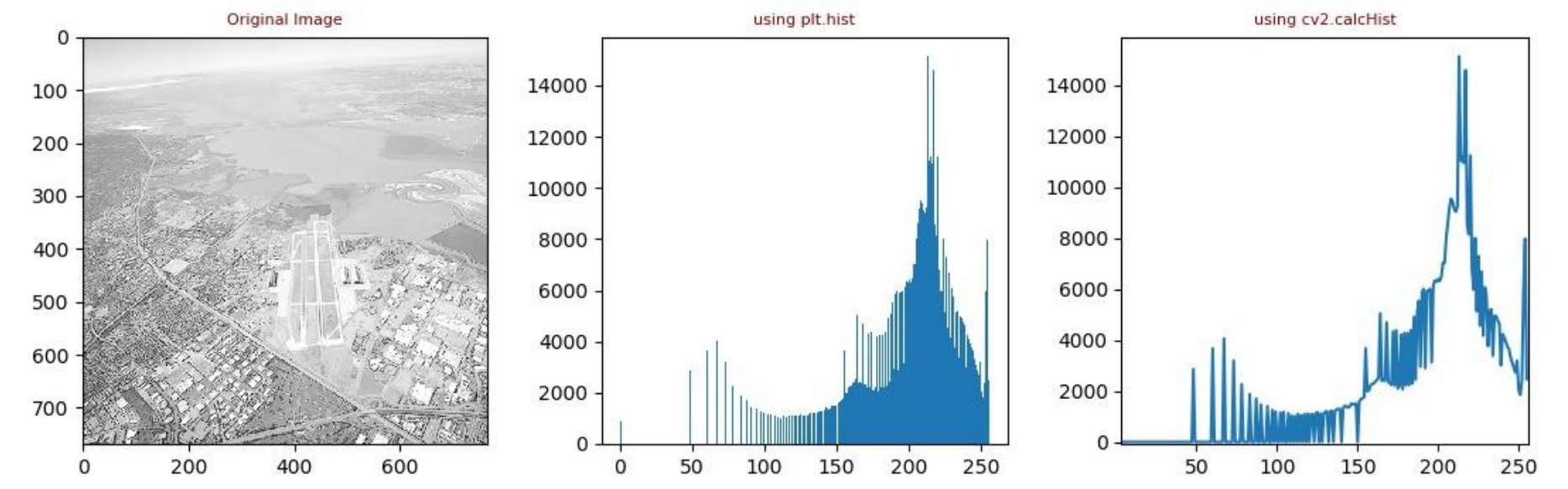
IMAGE PROCESSING LABORATORY

06 – Image Enhancements

Histogram Processing

Image Histograms

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

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

#using plt.hist
plt.hist(img.ravel(),256,[0,256])

#using cv2.calcHist
histr = cv2.calcHist([img],[0],None,[256],[0,256]),plt.plot(histr)
```

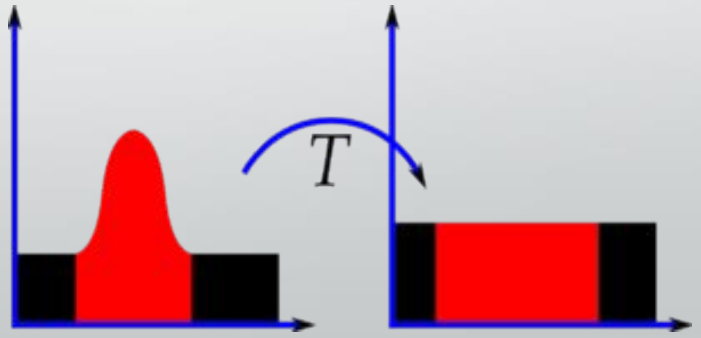


Histogram Equalization

This method usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values. This allows for areas of lower local contrast to gain a higher contrast.

Histogram equalization accomplishes this by effectively spreading out the highly populated intensity values which are used to degrade image contrast.

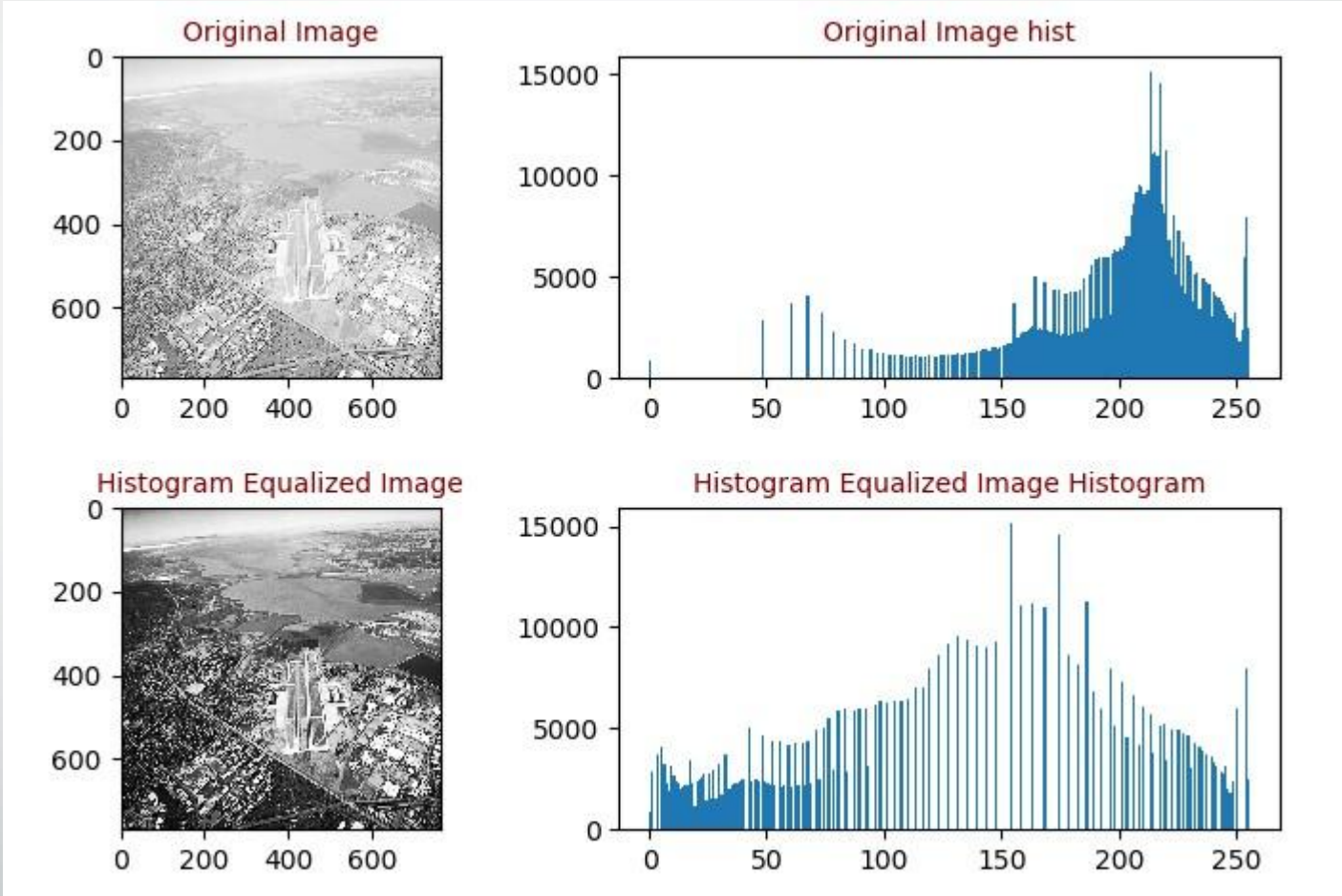
Useful in images with backgrounds and foregrounds that are both bright or both dark. This normally improves the contrast of the image.



Histograms of an image before and after equalization

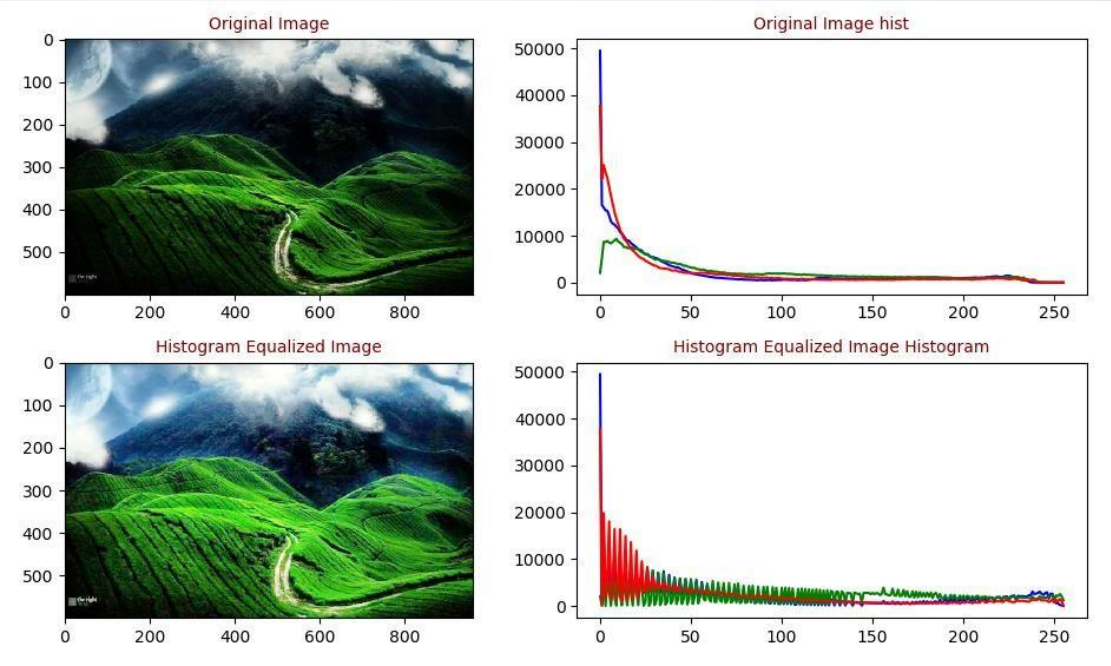
Histogram Equalization - Grayscale

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

FONT_SIZE = 10
COLOR = 'maroon'

img = cv2.imread(r'images\Aerial.tif',0)
hist_eq = cv2.equalizeHist(img)

plt.subplot(221)
plt.title('Original Image',
          c=COLOR,fontsize=FONT_SIZE)
plt.imshow(img,'gray')

plt.subplot(222)
plt.title('Original Image hist',
          c=COLOR,fontsize=FONT_SIZE)
plt.hist(img.ravel(),256,[0,256])

plt.subplot(223)
plt.title('Histogram Equalized Image',
          c=COLOR,fontsize=FONT_SIZE)
plt.imshow(hist_eq,'gray')

plt.subplot(224)
plt.title('Histogram Equalized Image Histogram',
          c=COLOR,fontsize=FONT_SIZE)
plt.hist(hist_eq.ravel(),256,[0,256])

plt.show()
```



Histogram Equalization - Color

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

FONT_SIZE = 10
COLOR = 'maroon'
img = cv2.imread(r'images\1.jpg',1)

def equalize_hist_color_hsv(img):
    H, S, V = cv2.split(cv2.cvtColor(img,cv2.COLOR_BGR2HSV))
    eq_V = cv2.equalizeHist(V)
    eq_image = cv2.cvtColor(cv2.merge([H, S, eq_V]),
                            cv2.COLOR_HSV2BGR)
    return eq_image

def histograms_for_color_img(img):
    hists = []
    for i in range(img.shape[2]):
        hist = cv2.calcHist([img],[i],None,[256],[0,256])
        hists.append(hist)
    return hists


hist_eq = equalize_hist_color_hsv(img)
hists_ = histograms_for_color_img(img)

hists_eq = histograms_for_color_img(hist_eq)
```



Original Image

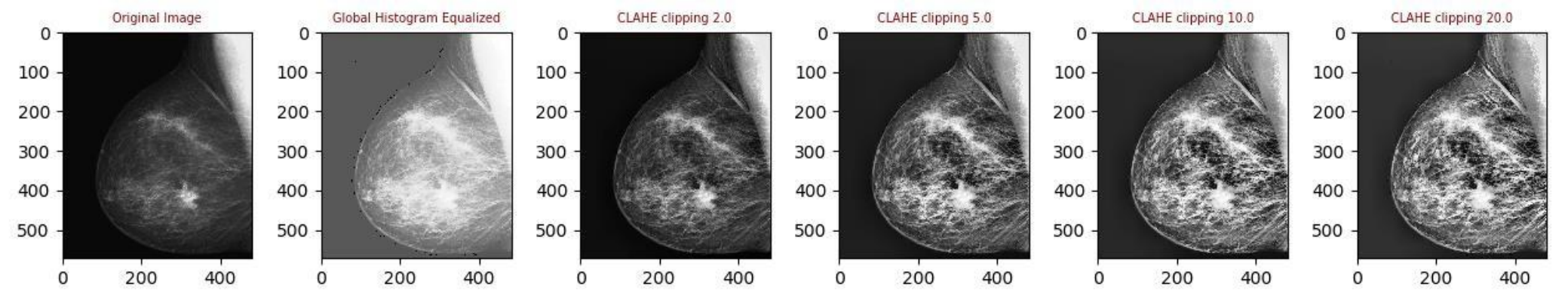
Histogram Equalized Image

Contrast Limited Adaptive Histogram Equalization

General Histogram Equalization considers the global contrast of the image. In many cases, it is not appropriate. (**Figure 1b – Global Histogram Equalized**).

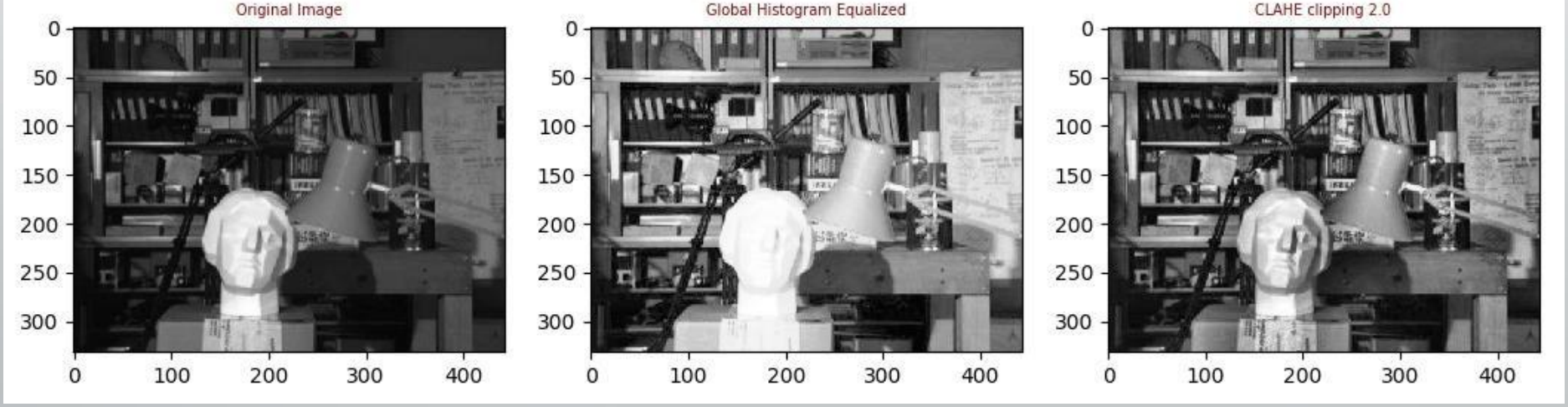
So to solve this problem, adaptive histogram equalization is used by dividing the image into small blocks called "tiles" (tileGridSize is 8x8 by default in OpenCV).

Then each of these blocks are histogram equalized. So in a small area, histogram would confine to a small region (unless there is noise). If noise is there, it will be amplified. To avoid this, contrast limiting is applied (**CLAHE**). If any histogram bin is above the specified contrast limit (by default 40 in OpenCV), those pixels are clipped and distributed uniformly to other bins before applying histogram equalization.



CLAHE

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

img = cv2.imread(r'images\clahe_1.jpg',0)

#global
hist_eq_img = cv2.equalizeHist(img)

# creating a CLAHE object
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
gray_image_clahe = clahe.apply(img)
```



CLAHE - Color

Create a function to apply CLAHE on color images and visualize outputs for clip limits 2.0, 5.0, 10.0 and 20.0. Use the image “1.jpg”

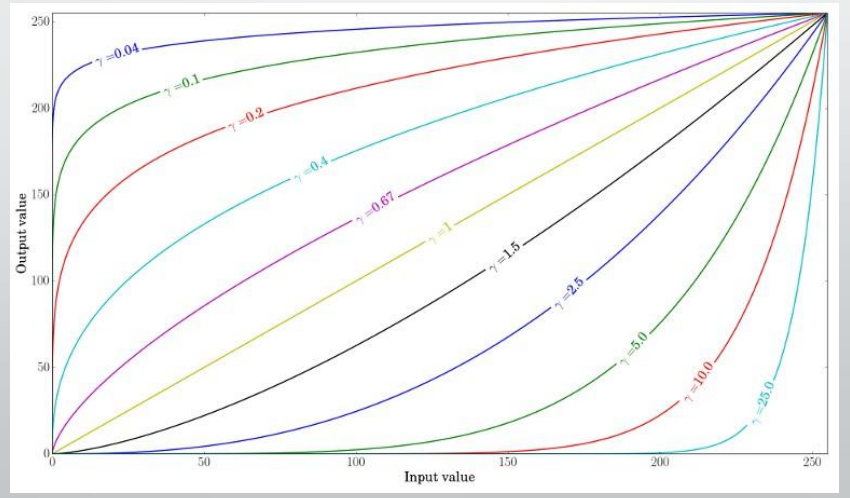
Power Law Transformations

Gamma Correction

Power-law (gamma) transformations can be mathematically expressed as ,

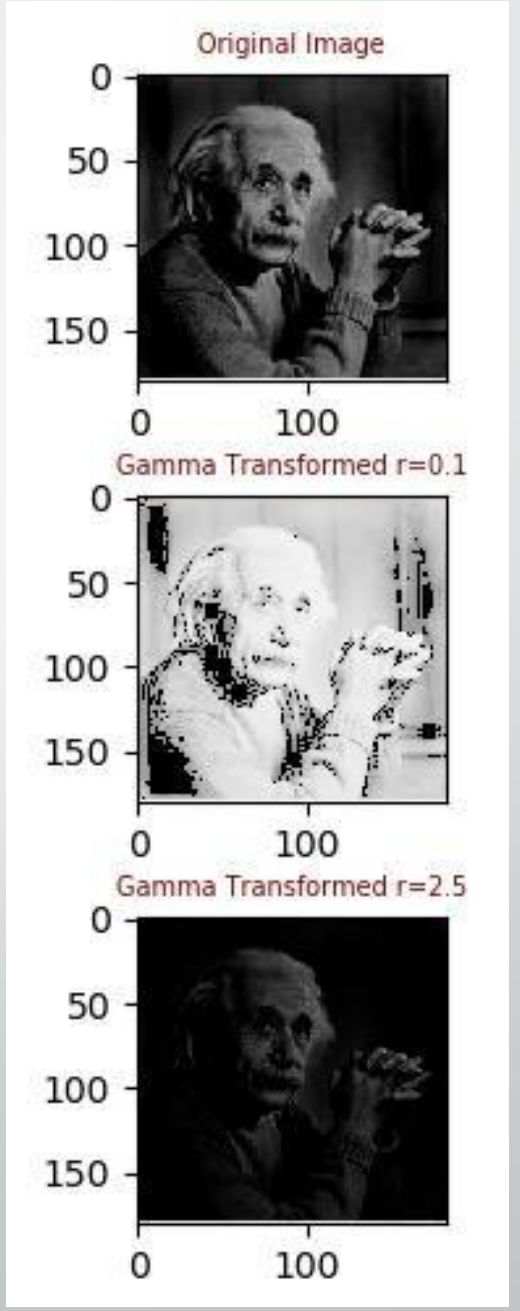
$$S = Imax \left(\frac{r}{I_{max}}\right)^{\gamma}$$

Where I_{max} is the maximum intensity value and γ is a positive constant, S and r are output and input pixel values respectively. Gamma correction is important for **displaying images on a screen correctly,** to prevent bleaching or darkening of images when viewed from different types of monitors with different display settings.

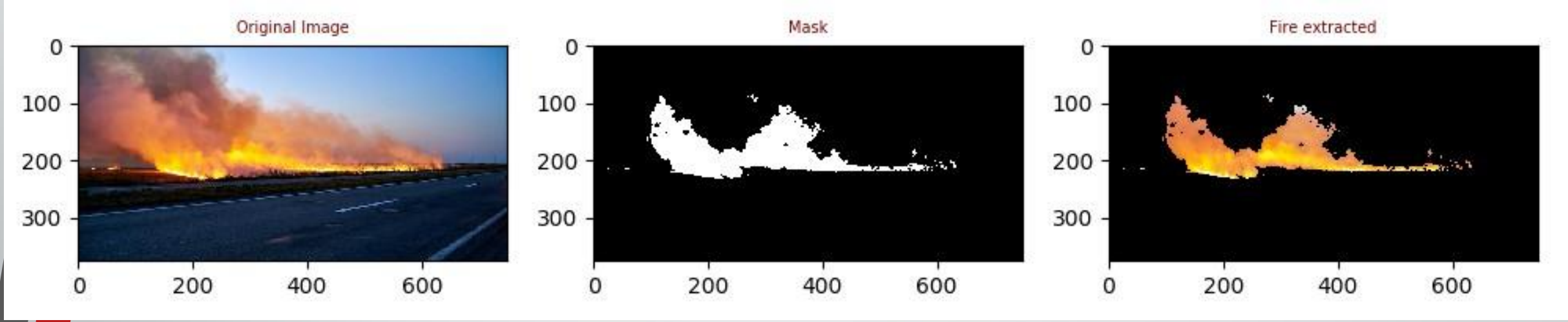


Gamma Correction - steps

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

img = cv2.imread(r'images\graylevel6.jpg',0)

def gamma_transform_(img, gamma):
    #I = np.zeros(img.shape, dtype=img.dtype)
    if(len(img.shape) == 3):
        for c in range(img.shape[2]):
            img[:,:,c] = 255 * (img[:,:,c]/255) ** (gamma)
    else:
        img = 255 * ((img/255) ** (gamma))

    return img

gm_1 = gamma_transform_(img,0.1)
gm_2 = gamma_transform_(img,2.5)
```



Assignment – Fire Extraction



– END –