Code:

```
# importing libraries
from PIL import Image
import cv2 as cv
import matplotlib.pyplot as plt

# Read Image file using OpenCV
image_path = 'fielder.jpeg'
img = cv.imread(image_path)
print(img.shape)

# showing image (By default OpenCV read image in BGR format)
plt.imshow(img)
plt.title('Image in BGR format')
plt.show()

#converting BGR format image to RGB format
img_rgb = cv.cvtColor(img,cv.COLOR_BGR2RGB)
plt.imshow(img_rgb)
plt.title('Image in RGB format')
plt.show()

# Resizing image and showing the resized image
resized_img_rgb = cv.resize(img_rgb,(128,128),cv.INTER_LINEAR)
plt.imshow(resized_img_rgb)
plt.title('Resized Image in RGB format')
plt.show()

print(resized_img_rgb.shape)
print(type(img))

# converting rgb image into Numpy array and showing the image
import numpy as np
img_data = np.array(img_rgb)
print(img_data.shape)
print(type(img_data))

new_img = Image.fromarray(img_data)
print(type(new_img))
plt.imshow(new_img)
plt.title('Image form numpy array')
plt.show()
```

Output:

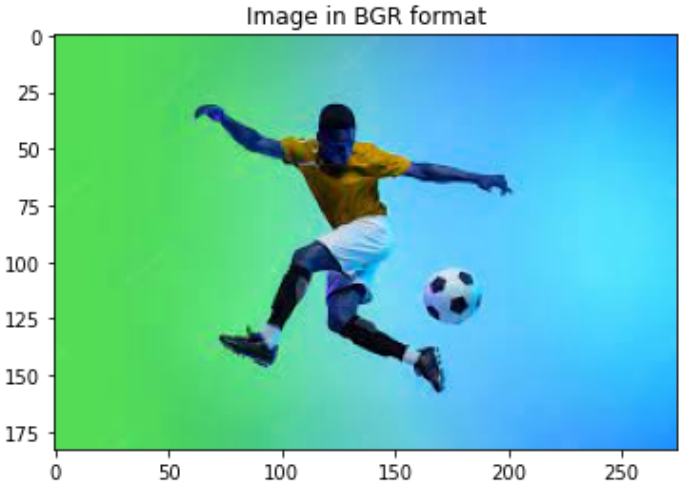


Image in BGR format
0
25
50
75
100
125
150
175
0
50
100
150
200
250

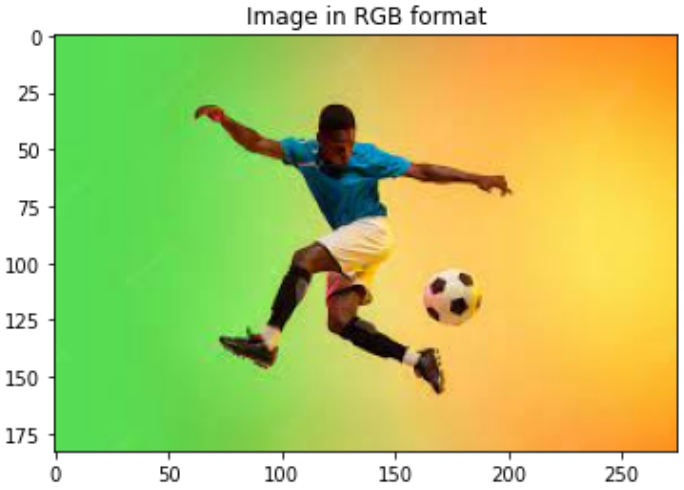
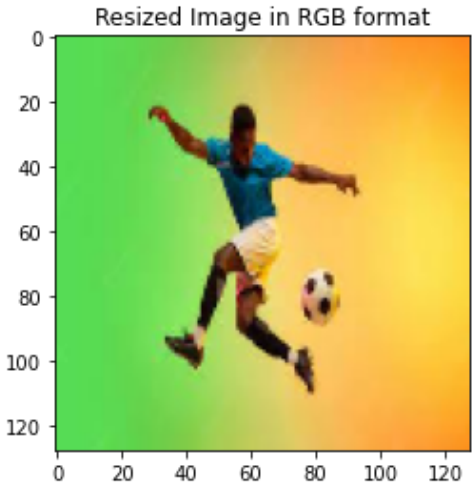


Image in RGB format
0
25
50
75
100
125
150
175
0
50
100
150
200
250


Resized Image in RGB format
0
20
40
60
80
100
120
0
20
40
60
80
100
120

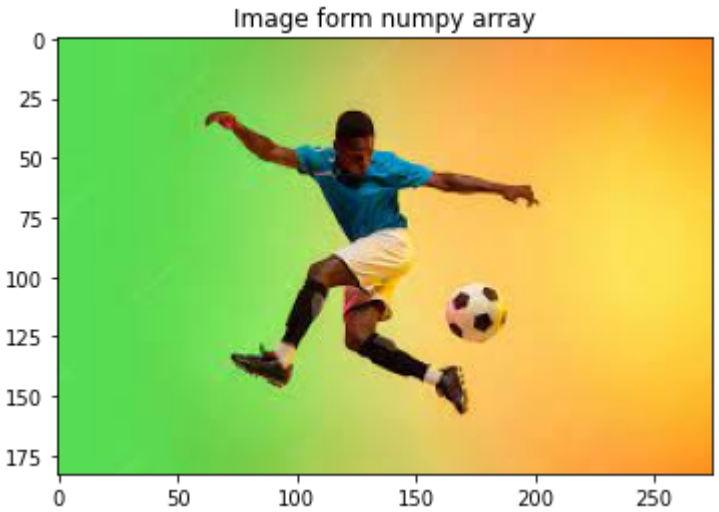


Image form numpy array
0
25
50
75
100
125
150
175
0
50
100
150
200
250

Code:

```
# importing libraries
from PIL import Image
import cv2 as cv
from matplotlib import pyplot as plt
import numpy as np

# reading image using OpenCV
image_path = 'Lenna.png'
img = cv.imread(image_path)
print(type(img))
print(img.shape)
plt.imshow(img)
plt.title('Original image in BGR format')
plt.axis('off')
plt.savefig('BGR_img.png')
plt.show()

# Converting BGR to GRAY
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

plt.imshow(img_gray, cmap= 'gray')
plt.title('Original image in grayscale')
plt.axis('off')
plt.savefig('Grayscale.jpg')
plt.show()

# Rotating Image in 90 degree clockwise
r_img = cv.rotate(img_gray, cv.ROTATE_90_CLOCKWISE)

plt.imshow(r_img, cmap= 'gray')
plt.title('90 degree rotated image')
plt.savefig('90_degree.jpg')
plt.show()

# Arbitrary angle rotation
h,w = img_gray.shape
rotation_matrix = cv.getRotationMatrix2D((h/2, w/2), 55, 1)
rotated_img = cv.warpAffine(img_gray,rotation_matrix,(h,w),
borderValue=100)

plt.imshow(rotated_img, cmap= 'gray')
```

```
plt.title('55 degree rotated image')
plt.axis('off')
plt.savefig('arbitrary rotated.jpg')
plt.show()
# Image Translation
tx = w/4; ty= h/4
translation_matrix = np.array([ [1,0,tx], [0,1,ty] ], dtype=np.float32)
img_translated = cv.warpAffine(img_gray, translation_matrix, (h,w))

plt.imshow(img_translated, cmap= 'gray')
plt.title('Translated Image')
plt.axis('off')
plt.savefig('Translated.jpg')
plt.show()

# Binarizing image using built-in OTSU-Method
thresh, binary_img = cv.threshold(img_gray, 154,255,
cv.THRESH_BINARY+cv.THRESH_OTSU)
print(thresh)

plt.imshow(binary_img, cmap= 'gray')
plt.title('Binarizing image using built-in OTSU-Method')
plt.axis('off')
plt.savefig('Binary image.jpg')
plt.show()
```

Output:

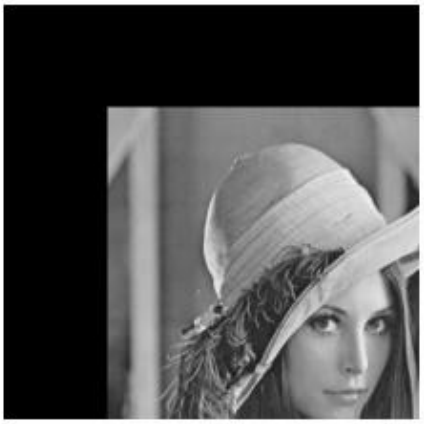
Original image in BGR format

Original image in grayscale

90 degree rotated image

55 degree rotated image

Translated Image



Binarizing image using built-in OTSU-Method

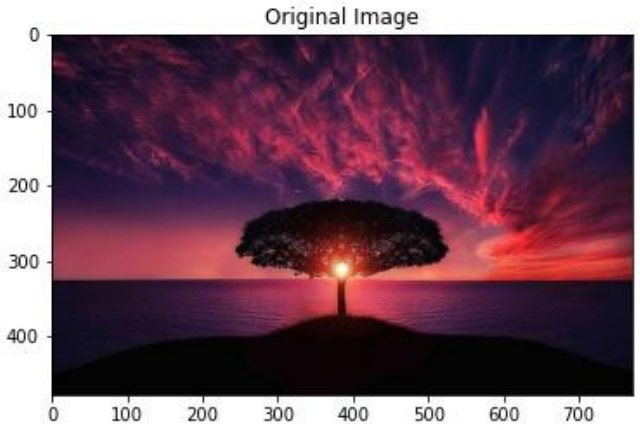
Code:

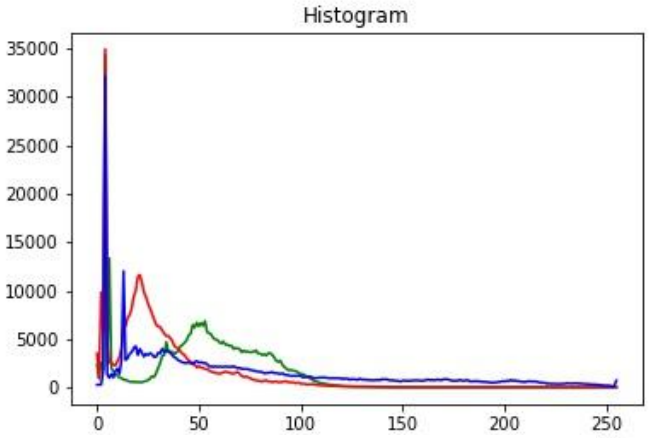
```
#importing libraries
import cv2 as cv
from matplotlib import pyplot as plt
from matplotlib.gridspec import GridSpec
import numpy as np

img_path = 'test.jpg'
img = cv.imread(img_path)
plt.title("Original Image")
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
plt.savefig('1.jpg')
plt.show()

# plotting histogram
colors = {'b','g','r'}
for i,color in enumerate(colors):
   hist = cv.calcHist([img],[i],None,[256],[0,256])
   plt.plot(hist,color = color)
plt.savefig('2.jpg')
plt.title('Histogram')
plt.show()

# Erosion
img_gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
thresh,bin_img=cv.threshold(img_gray,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

kernel=np.ones((5,5),np.float32)
eroded_img=cv.erode(bin_img,kernel,iterations=2)

hist_grayscale = cv.calcHist([img_gray], [0], None, [256], [0,256])
hist_otsu = cv.calcHist([bin_img], [0], None, [256], [0,256])
hist_eroded = cv.calcHist([eroded_img], [0], None, [256], [0,256])

row,col=2,3
fig=plt.figure(figsize=(15,10))
gs=GridSpec(row,col)

fig.add_subplot(gs[0,0])
plt.title('Original Grayscale Image')
plt.imshow(img_gray,cmap='gray')
```

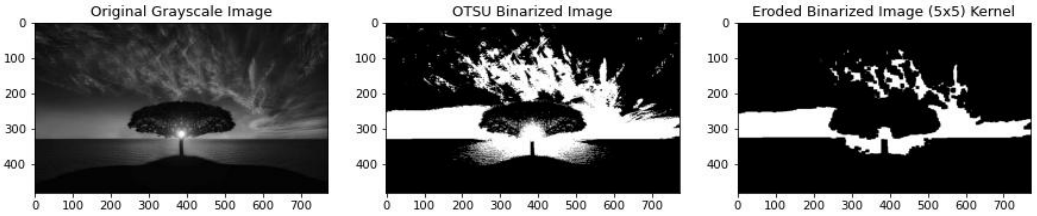
```
fig.add_subplot(gs[0,1])
plt.title('OTSU Binarized Image')
plt.imshow(bin_img,cmap='gray')

fig.add_subplot(gs[0,2])
plt.title('Eroded Binarized Image (5x5) Kernel')
plt.imshow(eroded_img,cmap='gray')

fig.add_subplot(gs[1,0])
plt.title('Histogram')
plt.plot(hist_grayscale)

fig.add_subplot(gs[1,1])
plt.title('Histogram')
plt.plot(hist_otsu)

fig.add_subplot(gs[1,2])
plt.title('Histogram')
plt.plot(hist_eroded)
plt.savefig('3.jpg')
plt.show()

# Dilation
dilated_img=cv.dilate(bin_img,kernel,iterations=1)
hist_dilated = cv.calcHist([dilated_img], [0], None, [256], [0,256])

row,col=2,3
fig=plt.figure(figsize=(15,10))
gs=GridSpec(row,col)
fig.add_subplot(gs[0,0])
plt.title('Original Grayscale Image')
plt.imshow(img_gray,cmap='gray')

fig.add_subplot(gs[0,1])
plt.title('OTSU Binarized Image')
plt.imshow(bin_img,cmap='gray')

fig.add_subplot(gs[0,2])
plt.title('Dilated Binarized Image (5x5) Kernel')
plt.imshow(dilated_img,cmap='gray')

fig.add_subplot(gs[1,0])
plt.title('Histogram')
```

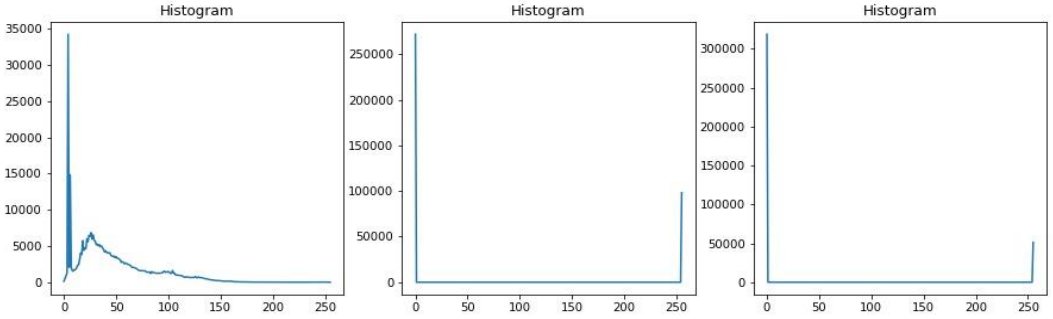
```
plt.plot(hist_grayscale)
fig.add_subplot(gs[1,1])
plt.title('Histogram')
plt.plot(hist_otsu)

fig.add_subplot(gs[1,2])
plt.title('Histogram')
plt.plot(hist_eroded)
plt.savefig('4.jpg')
plt.show()

# Gradient
grad_img = cv.morphologyEx(bin_img, cv.MORPH_GRADIENT, kernel)
hist_gradient = cv.calcHist([grad_img], [0], None, [256], [0,256])

row,col=2,3
fig=plt.figure(figsize=(15,10))
gs=GridSpec(row,col)

fig.add_subplot(gs[0,0])
plt.title('Original Grayscale Image')
plt.imshow(img_gray,cmap='gray')

fig.add_subplot(gs[0,1])
plt.title('OTSU Binarized Image')
plt.imshow(bin_img,cmap='gray')

fig.add_subplot(gs[0,2])
plt.title('Gradient Binarized Image (5x5) Kernel')
plt.imshow(grad_img,cmap='gray')

fig.add_subplot(gs[1,0])
plt.title('Histogram')
plt.plot(hist_grayscale)

fig.add_subplot(gs[1,1])
plt.title('Histogram')
plt.plot(hist_otsu)
fig.add_subplot(gs[1,2])
plt.title('Histogram')
plt.plot(hist_gradient)
plt.savefig('5.jpg')
plt.show()
```

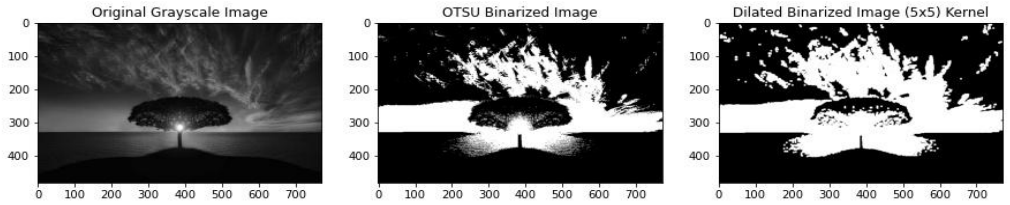
Output:

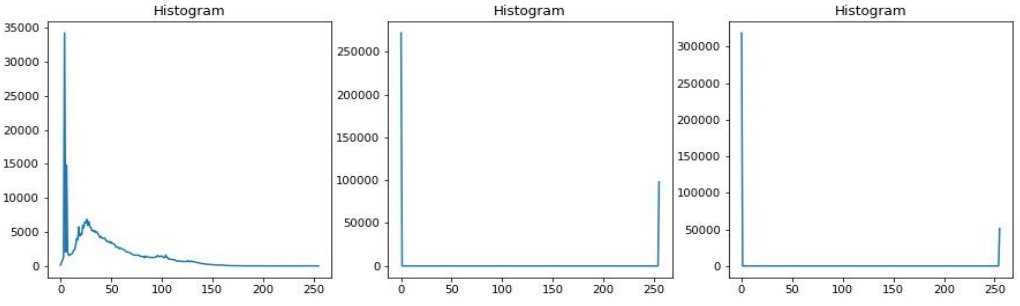

Original Image

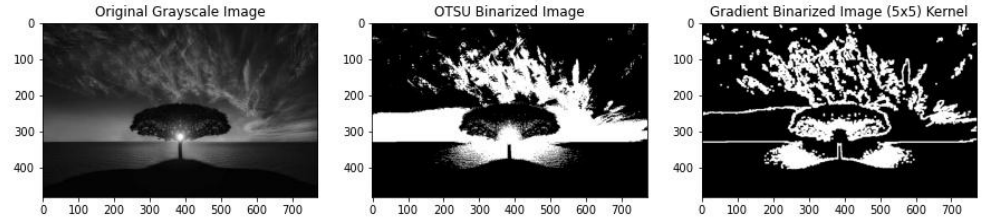

Histogram

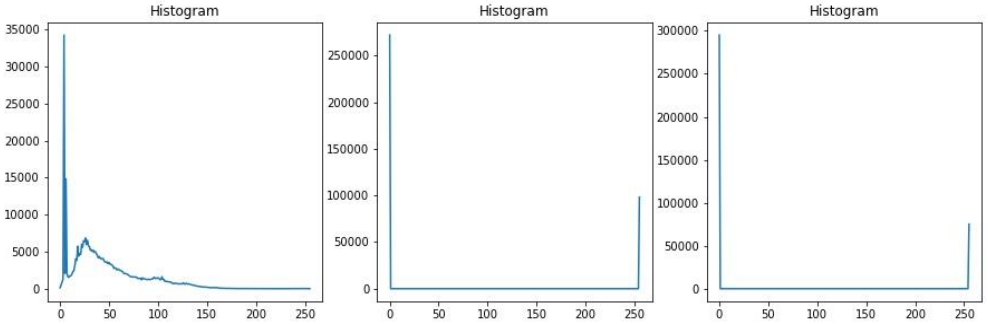

Original Grayscale Image
OTSU Binarized Image
Eroded Binarized Image (5x5) Kernel

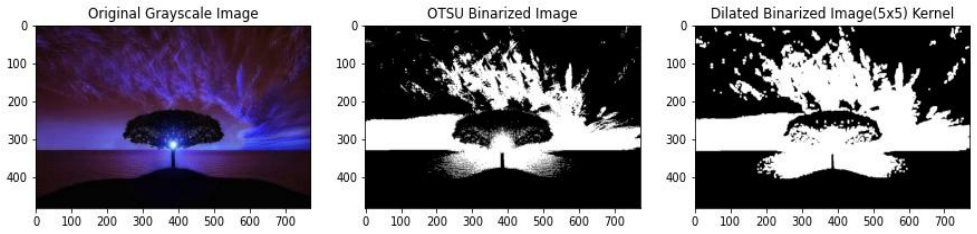

Histogram
Histogram
Histogram

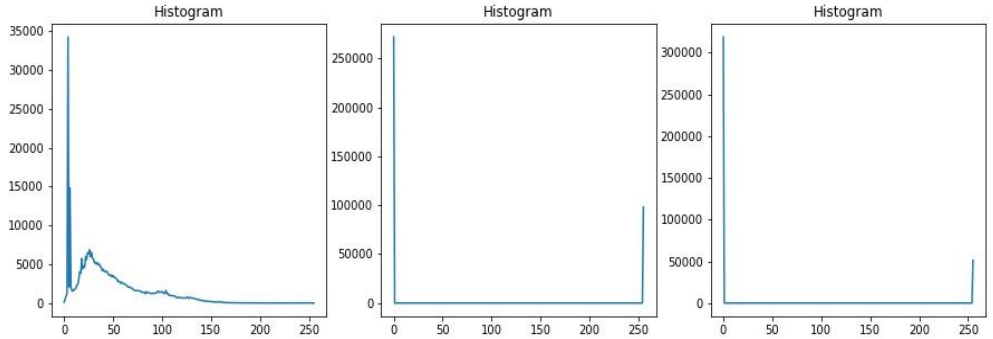

Original Grayscale Image
OTSU Binarized Image
Dilated Binarized Image (5x5) Kernel


Histogram
Histogram
Histogram


Original Grayscale Image
OTSU Binarized Image
Gradient Binarized Image (5x5) Kernel


Histogram
Histogram
Histogram


Original Grayscale Image
OTSU Binarized Image
Dilated Binarized Image(5x5) Kernel


Histogram
Histogram
Histogram

Code:

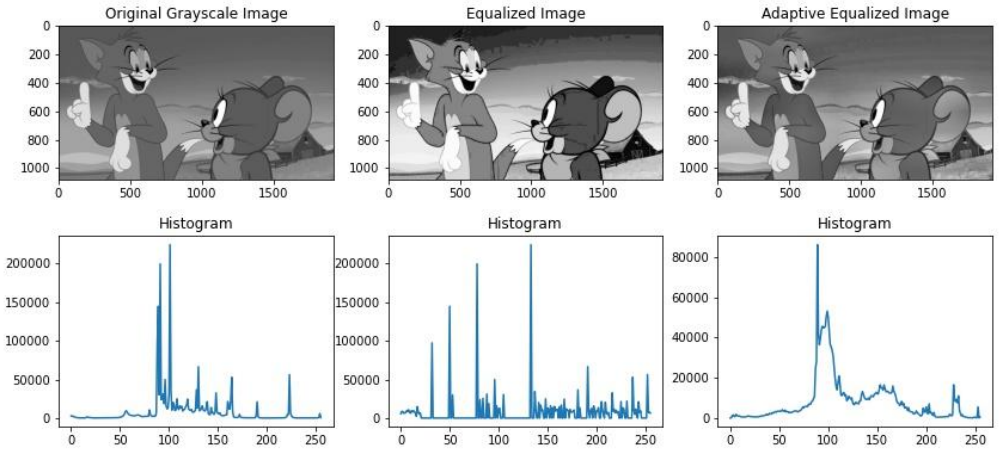
```
# importing libraries
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

# reading image
img = cv.imread('jerry.jpg')
img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)
img_gray =  cv.cvtColor(img_rgb, cv.COLOR_BGR2GRAY)

# histogram calculation
hist = cv.calcHist([img_gray],[0],None,[256],[0,256])
img_eq = cv.equalizeHist(img_gray)
hist_eq = cv.calcHist([img_eq],[0],None,[256],[0,256])

cl = cv.createCLAHE(3.0,(8,8))
img_ad_eq = cl.apply(img_gray)
hist_ad_eq = cv.calcHist([img_ad_eq],[0],None,[256],[0,256])

row,col = 3,3
fig = plt.figure(figsize=(14,10))
gs = GridSpec(row,col)

fig.add_subplot(gs[0,0])
plt.title('Original Grayscale Image')
plt.imshow(img_gray,cmap='gray')

fig.add_subplot(gs[0,1])
plt.title('Equalized Image')
plt.imshow(img_eq,cmap='gray')

fig.add_subplot(gs[0,2])
plt.title('Adaptive Equalized Image')
plt.imshow(img_ad_eq,cmap='gray')

fig.add_subplot(gs[1,0])
plt.title('Histogram')
plt.plot(hist)

fig.add_subplot(gs[1,1])
```

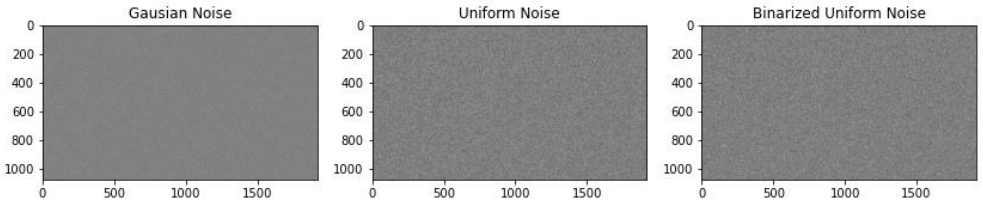
```
plt.title('Histogram')
plt.plot(hist_eq)

fig.add_subplot(gs[1,2])
plt.title('Histogram')
plt.plot(hist_ad_eq)

plt.savefig('1.jpg')
plt.show()

# Gausian Noise
gaussian_noise = np.zeros((img_gray.shape[0],img_gray.shape[1]),dtype =
np.uint8)
cv.randn(gaussian_noise,128,20)

gaussian_noise = (gaussian_noise*0.7).astype(np.uint8)
noise_img = cv.add(img_gray,gaussian_noise)

# Uniform Noise
uniform_noise = np.zeros((img_gray.shape[0],img_gray.shape[1]),dtype =
np.uint8)
cv.randu(uniform_noise,0,255)
uniform_noise = (uniform_noise*0.7).astype(np.uint8)
uf_noise_img = cv.add(img_gray,uniform_noise)

# Binarized Uniform Noise
new_img = uniform_noise.copy()
ret,new_img = cv.threshold(uniform_noise,100,200,cv.THRESH_BINARY)
uf_bin_new = cv.add(img_gray,new_img)

row, col = 3, 3
fig = plt.figure(figsize=(14, 12))
gs = GridSpec(row, col)

fig.add_subplot(gs[0, 0])
plt.title("Gausian Noise")
plt.imshow(gaussian_noise, cmap='gray')

fig.add_subplot(gs[0, 1])
plt.title('Uniform Noise')
plt.imshow(uniform_noise, cmap='gray')

fig.add_subplot(gs[0, 2])
```

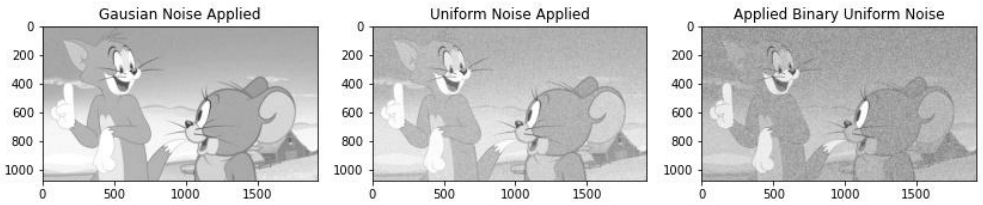
```
plt.title('Binarized Uniform Noise')
plt.imshow(uniform_noise, cmap='gray')

fig.add_subplot(gs[1, 0])
plt.title('Gausian Noise Applied')
plt.imshow(noise_img, cmap='gray')

fig.add_subplot(gs[1, 1])
plt.title('Uniform Noise Applied')
plt.imshow(uf_noise_img, cmap='gray')

fig.add_subplot(gs[1, 2])
plt.title('Applied Binary Uniform Noise')
plt.imshow(uf_bin_new, cmap='gray')

plt.savefig('2.jpg')
plt.show()

row, col = 3, 3
fig = plt.figure(figsize=(14, 12))
gs = GridSpec(row, col)

fig.add_subplot(gs[0, 0])
plt.title("Gausian Noise")
plt.imshow(gaussian_noise, cmap='gray')

fig.add_subplot(gs[0, 1])
plt.title('Uniform Noise')
plt.imshow(uniform_noise, cmap='gray')

fig.add_subplot(gs[0, 2])
plt.title('Binarized Uniform Noise')
plt.imshow(uniform_noise, cmap='gray')


fig.add_subplot(gs[1, 0])
plt.title('Gausian Noise Applied')
plt.imshow(noise_img, cmap='gray')

fig.add_subplot(gs[1, 1])
plt.title('Uniform Noise Applied')
plt.imshow(uf_noise_img, cmap='gray')
```

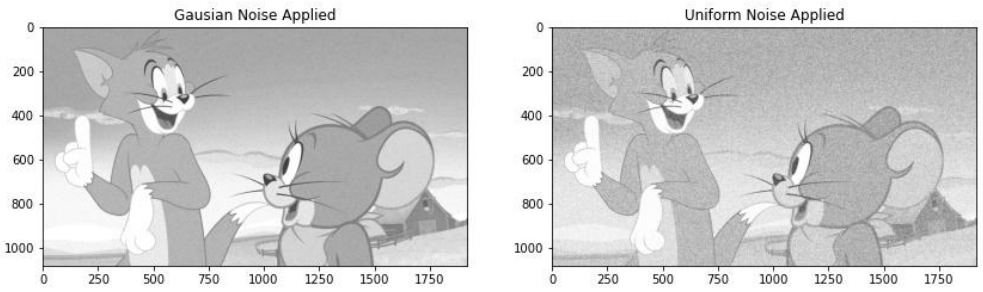
```
fig.add_subplot(gs[1, 2])
plt.title('Applied Binary Uniform Noise')
plt.imshow(uf_bin_new, cmap='gray')

plt.savefig('2.jpg')
plt.show()

row, col = 2, 2
fig = plt.figure(figsize=(14,10))
gs = GridSpec(row, col)

fig.add_subplot(gs[0, 0])
plt.title('Gausian Noise Applied')
plt.imshow(noise_img, cmap='gray')

fig.add_subplot(gs[0, 1])
plt.title('Uniform Noise Applied')
plt.imshow(uf_noise_img, cmap='gray')

fig.add_subplot(gs[1, 0])
plt.title('Gausian Blurre image')
plt.imshow(g_blur, cmap='gray')

fig.add_subplot(gs[1, 1])
plt.title('median Blured image')
plt.imshow(m_blur, cmap='gray')

plt.savefig('3.jpg')
plt.show()
```

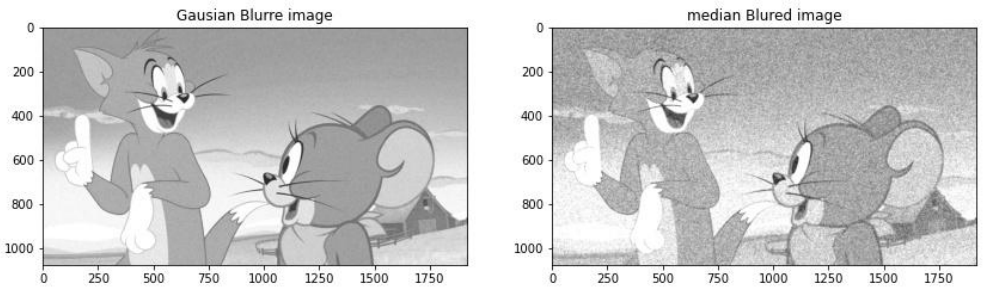
Output:


Original Grayscale Image
Equalized Image
Adaptive Equalized Image
Histogram
Histogram
Histogram


Gausian Noise
Uniform Noise
Binarized Uniform Noise


Gausian Noise Applied
Uniform Noise Applied
Applied Binary Uniform Noise


Gausian Noise Applied
Uniform Noise Applied


Gausian Blurre image
median Blured image

Code:

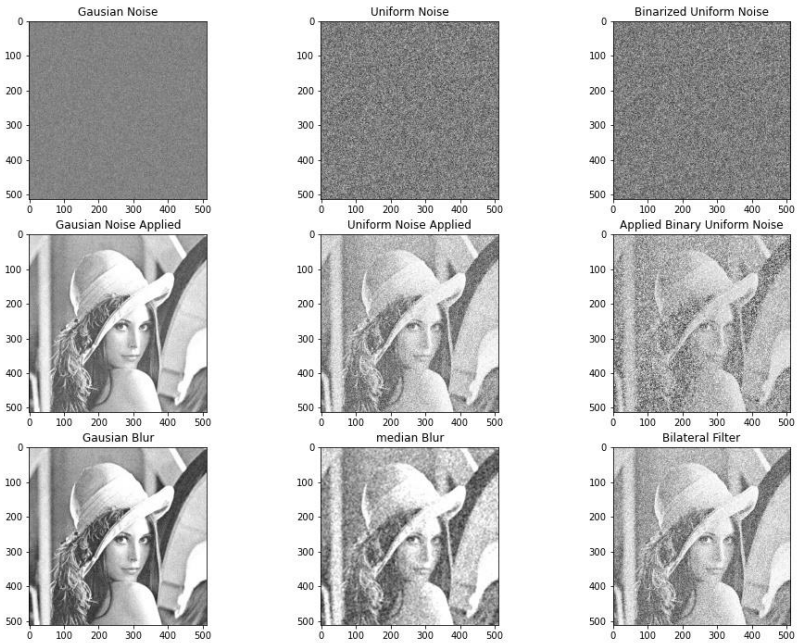
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

img = cv.imread('Lenna.png')
img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)
img_gray =  cv.cvtColor(img_rgb, cv.COLOR_BGR2GRAY)
hist = cv.calcHist([img_gray],[0],None,[256],[0,256])

img_eq = cv.equalizeHist(img_gray)
hist_eq = cv.calcHist([img_eq],[0],None,[256],[0,256])

cl = cv.createCLAHE(3.0,(8,8))
img_ad_eq = cl.apply(img_gray)
hist_ad_eq = cv.calcHist([img_ad_eq],[0],None,[256],[0,256])

gaussian_noise = np.zeros((img_gray.shape[0],img_gray.shape[1]),dtype =
np.uint8)
cv.randn(gaussian_noise,128,20)

gaussian_noise = (gaussian_noise*0.7).astype(np.uint8)
noise_img = cv.add(img_gray,gaussian_noise)
uniform_noise = np.zeros((img_gray.shape[0],img_gray.shape[1]),dtype =
np.uint8)
cv.randu(uniform_noise,0,255)
uniform_noise = (uniform_noise*0.7).astype(np.uint8)
uf_noise_img = cv.add(img_gray,uniform_noise)

g_blur = cv.GaussianBlur(noise_img,(5,5),0.5)
m_blur = cv.medianBlur(uf_noise_img,5)

#Bilateral Filter
new_img = uniform_noise.copy()
ret,new_img = cv.threshold(uniform_noise,88,225,cv.THRESH_BINARY)
uf_bin_new = cv.add(img_gray,new_img)
uf_bsm = cv.bilateralFilter(uf_noise_img,3,15,15)

row,col = 3,3
fig = plt.figure(figsize=(16,12))
gs = GridSpec(row,col)
```

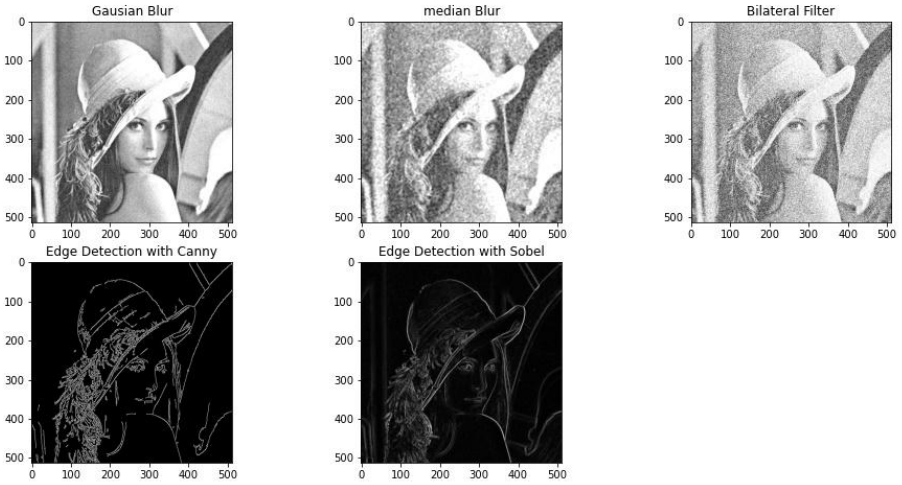
```
fig.add_subplot(gs[0,0])
plt.title("Gausian Noise")
plt.imshow(gaussian_noise,cmap='gray')

fig.add_subplot(gs[0,1])
plt.title('Uniform Noise')
plt.imshow(uniform_noise,cmap='gray')

fig.add_subplot(gs[0,2])
plt.title('Binarized Uniform Noise')
plt.imshow(uniform_noise,cmap='gray')

fig.add_subplot(gs[1,0])
plt.title('Gausian Noise Applied')
plt.imshow(noise_img,cmap='gray')

fig.add_subplot(gs[1,1])
plt.title('Uniform Noise Applied')
plt.imshow(uf_noise_img,cmap='gray')

fig.add_subplot(gs[1,2])
plt.title('Applied Binary Uniform Noise')
plt.imshow(uf_bin_new,cmap='gray')

fig.add_subplot(gs[2,0])
plt.title('Gausian Blur')
plt.imshow(g_blur,cmap='gray')

fig.add_subplot(gs[2,1])
plt.title('median Blur')
plt.imshow(m_blur,cmap='gray')

fig.add_subplot(gs[2,2])
plt.title('Bilateral Filter')
plt.imshow(uf_bsm,cmap='gray')
plt.savefig('1.jpg')
plt.show()

# Canny Edge Detection
canny_img = cv.Canny(img_gray,89,190)

#Sobel
grad_x = cv.Sobel(img_gray,cv.CV_64F,1,0)
```

```
grad_y = cv.Sobel(img_gray,cv.CV_64F,0,1)
grad = np.sqrt(grad_x**2 + grad_y**2)
grad_norm = (grad*255 / grad.max()).astype(np.uint8)

row,col = 3,3
fig = plt.figure(figsize=(16,12))
gs = GridSpec(row,col)


fig.add_subplot(gs[0,0])
plt.title('Gausian Blur')
plt.imshow(g_blur,cmap='gray')

fig.add_subplot(gs[0,1])
plt.title('median Blur')
plt.imshow(m_blur,cmap='gray')

fig.add_subplot(gs[0,2])
plt.title('Bilateral Filter')
plt.imshow(uf_bsm,cmap='gray')

fig.add_subplot(gs[1,0])
plt.title('Edge Detection with Canny')
plt.imshow(canny_img,cmap='gray')

fig.add_subplot(gs[1,1])
plt.title('Edge Detection with Sobel')
plt.imshow(grad_norm,cmap='gray')

plt.savefig('2.jpg')
plt.show()

# Template Matching
foot_path = 'football.jpg'
fielder_path = 'fl1.jpeg'

img1 = cv.imread(foot_path)
img2 = cv.imread(fielder_path)

h,w,c = img1.shape

img1 = cv.cvtColor(img1,cv.COLOR_BGR2GRAY)
img2 = cv.cvtColor(img2,cv.COLOR_BGR2GRAY)
```

```
img2c = img2.copy()

result = cv.matchTemplate(img2c,img1,cv.TM_CCOEFF)
min_val,max_val,min_loc,max_loc = cv.minMaxLoc(result)

print(f'min_location: {min_loc}\nmax_location: {max_loc}')

topleft = max_loc
bottom_right = [topleft[0] + w, topleft[1] + h]
cv.rectangle(img2c,topleft,bottom_right,255,3)
plt.imshow(img2c,cmap='gray')
plt.savefig('3.jpg')
plt.show()
```

Output:




Gausian Blur
median Blur
Bilateral Filter
Edge Detection with Canny
Edge Detection with Sobel