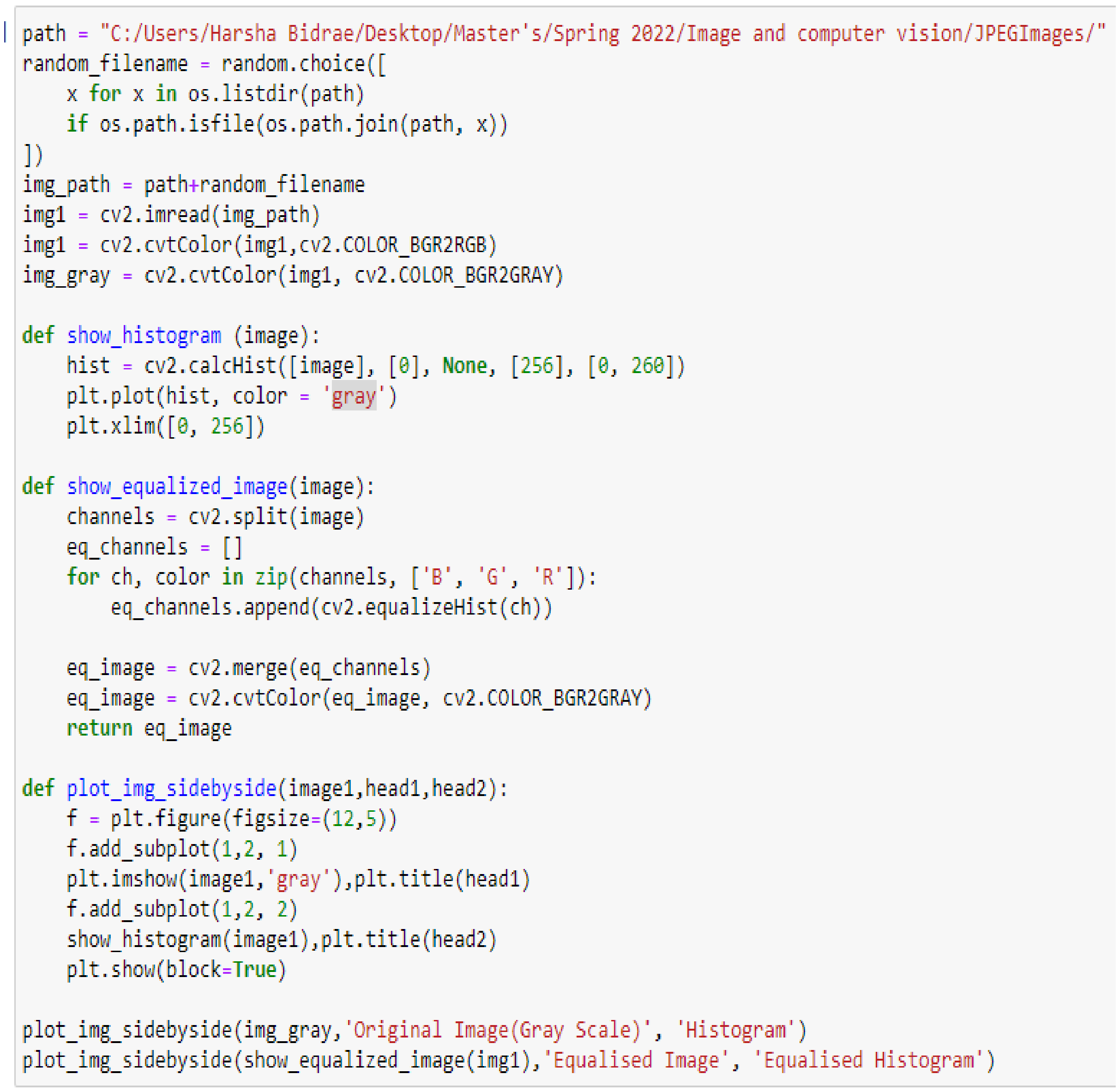
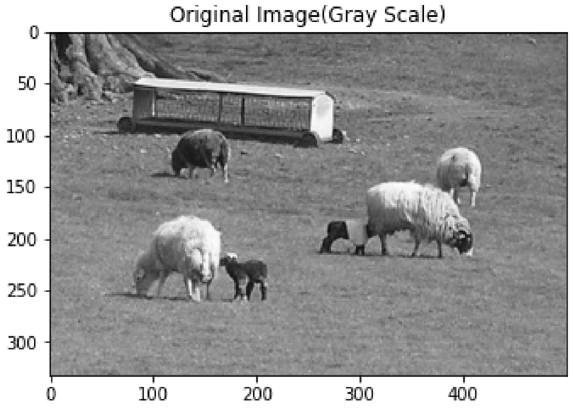
Image-Analysis-and-Computer-Vision-CS-898BA

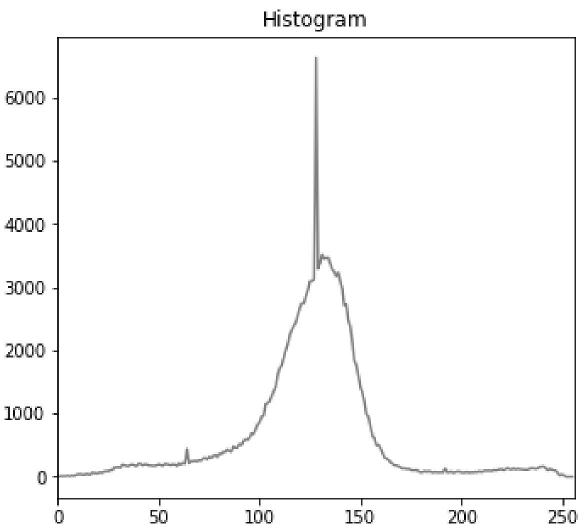
Name: **Harsha Siddapura Gnaneshwara.**
Wsu Id: **w786p696.**

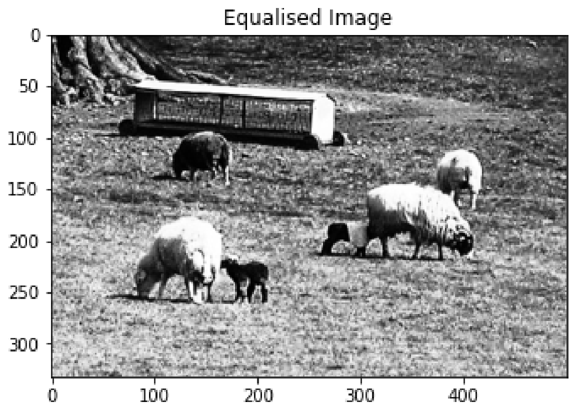
1) On a randomly selected three images from PASCAL VOC dataset compute (a) image statistics, (b) histogram, (c) histogram equalization and (d) binarize it.

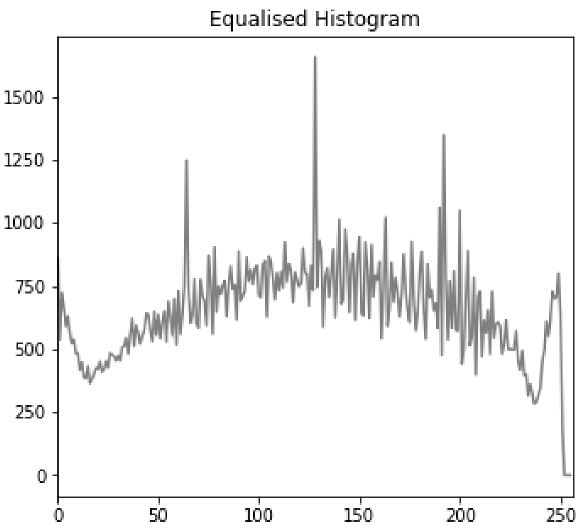


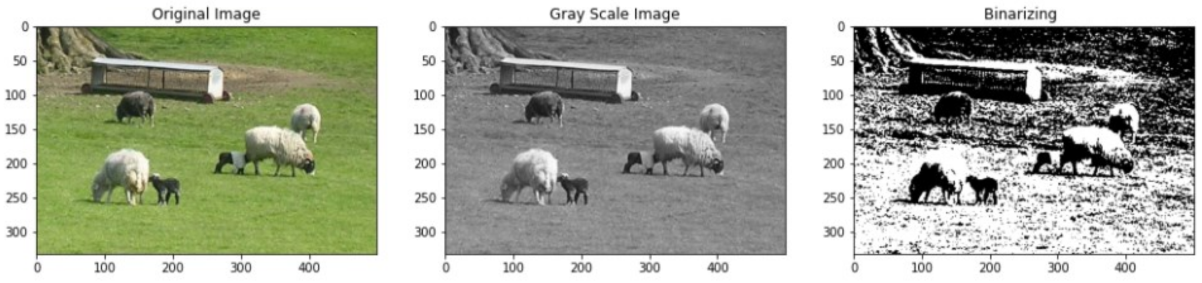
```python
path = "C:/Users/Harsha Bidrae/Desktop/Master's/Spring 2022/Image and computer vision/JPEGImages/"
random_filename = random.choice([
    x for x in os.listdir(path)
    if os.path.isfile(os.path.join(path, x))
])
img_path = path+random_filename
img1 = cv2.imread(img_path)
img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

def show_histogram (image):
    hist = cv2.calcHist([image], [0], None, [256], [0, 260])
    plt.plot(hist, color = 'gray')
    plt.xlim([0, 256])

def show_equalized_image(image):
    channels = cv2.split(image)
    eq_channels = []
    for ch, color in zip(channels, ['B', 'G', 'R']):
        eq_channels.append(cv2.equalizeHist(ch))

    eq_image = cv2.merge(eq_channels)
    eq_image = cv2.cvtColor(eq_image, cv2.COLOR_BGR2GRAY)
    return eq_image

def plot_img_sidebyside(image1,head1,head2):
    f = plt.figure(figsize=(12,5))
    f.add_subplot(1,2, 1)
    plt.imshow(image1,'gray'),plt.title(head1)
    f.add_subplot(1,2, 2)
    show_histogram(image1),plt.title(head2)
    plt.show(block=True)

plot_img_sidebyside(img_gray,'Original Image(Gray Scale)', 'Histogram')
plot_img_sidebyside(show_equalized_image(img1),'Equalised Image', 'Equalised Histogram')
```

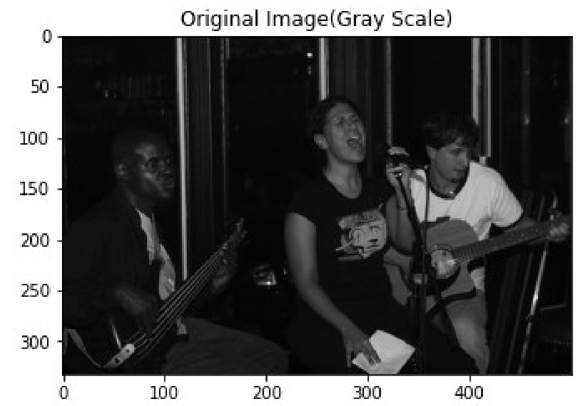

Original Image(Gray Scale)
0
50
100
150
200
250
300
0
100
200
300
400

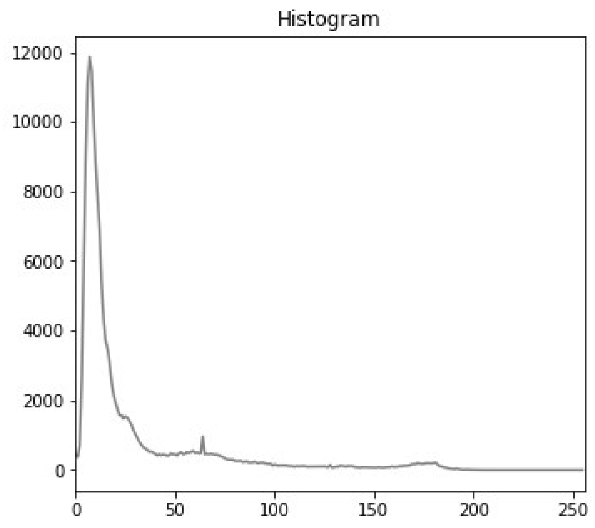

Histogram
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250

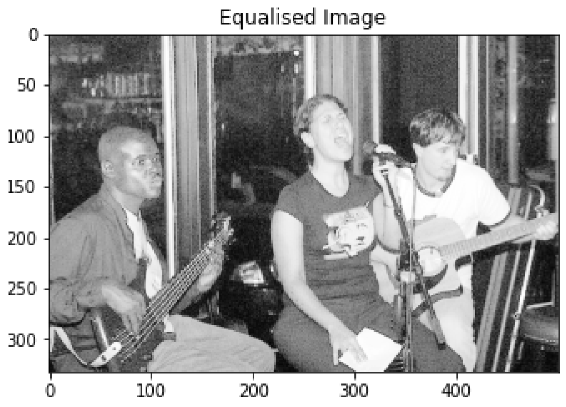

Equalised Image
0
50
100
150
200
250
300
0
100
200
300
400

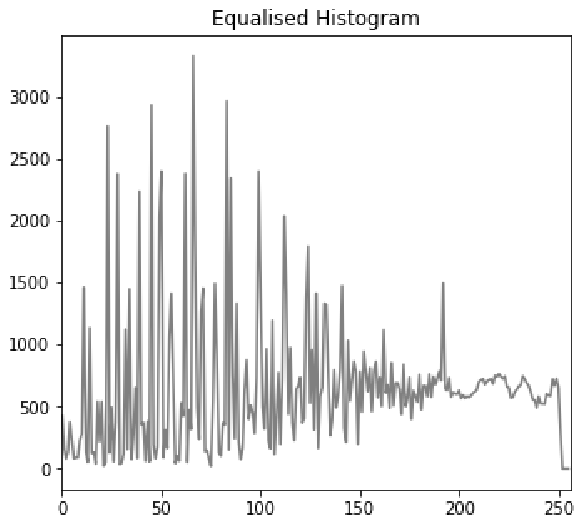

Equalised Histogram
1500
1250
1000
750
500
250
0
0
50
100
150
200
250

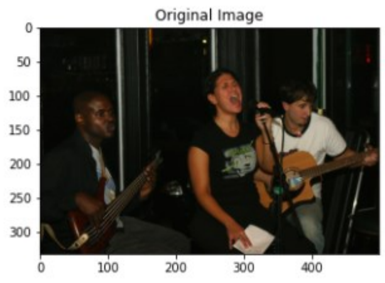

Original Image
Gray Scale Image
Binarizing
0
50
100
150
200
250
300
0
100
200
300
400

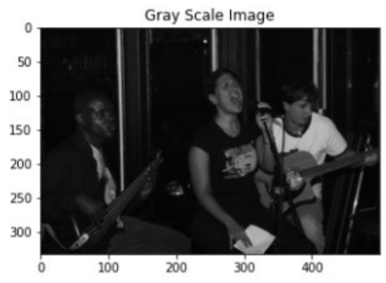

Original Image(Gray Scale)

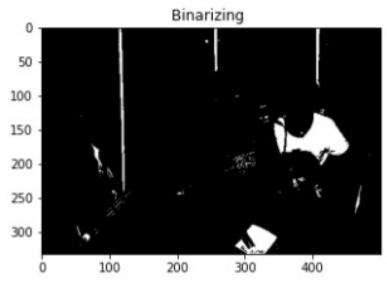

Histogram

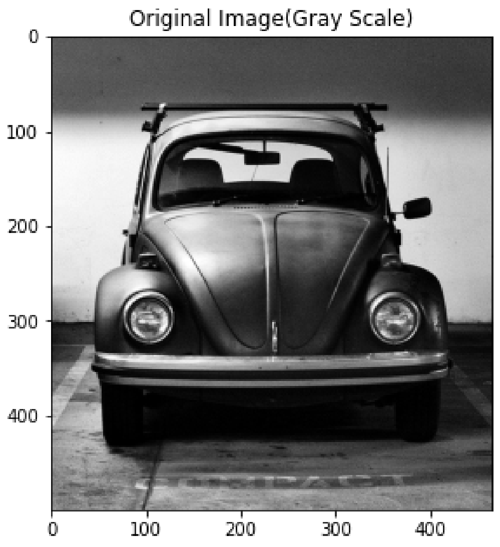

Equalised Image

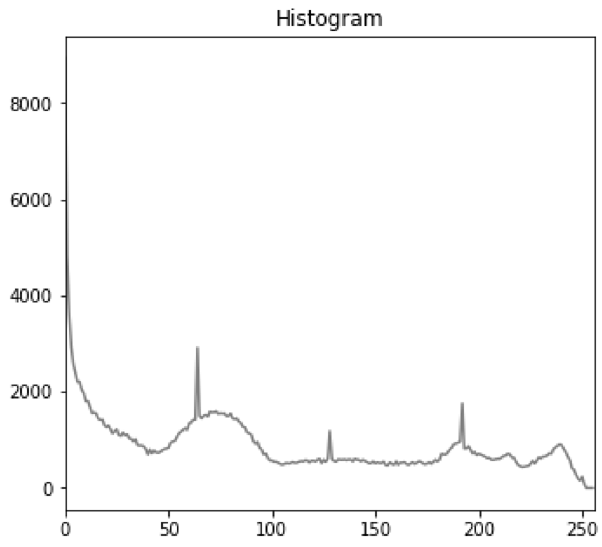

Equalised Histogram

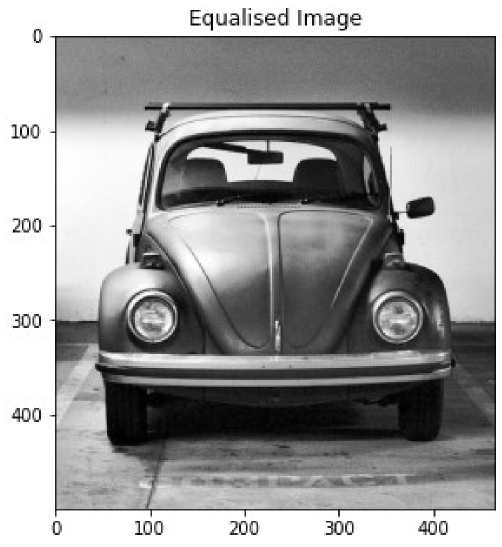

Original Image

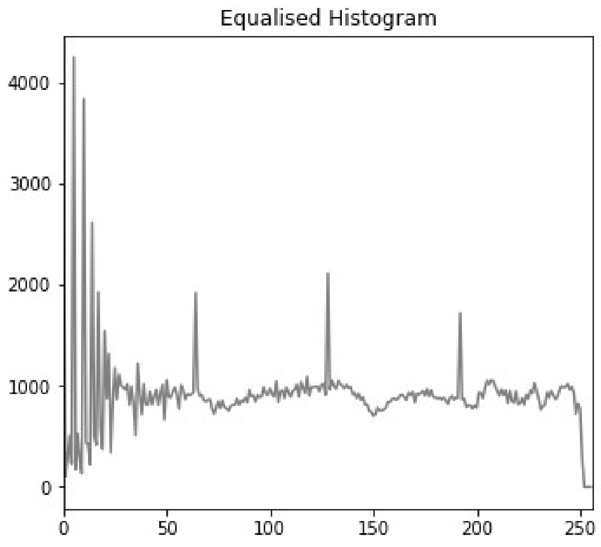

Gray Scale Image

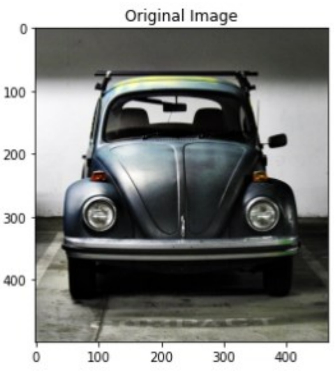

Binarizing

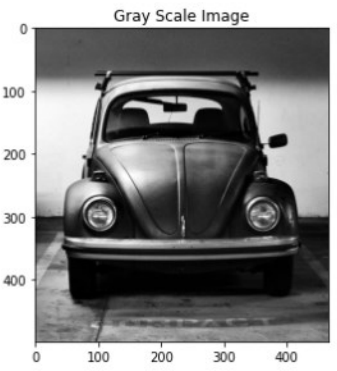

Original Image(Gray Scale)

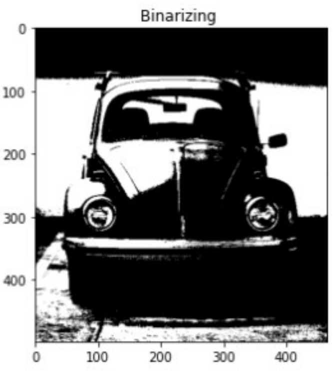

Histogram


Equalised Image


Equalised Histogram

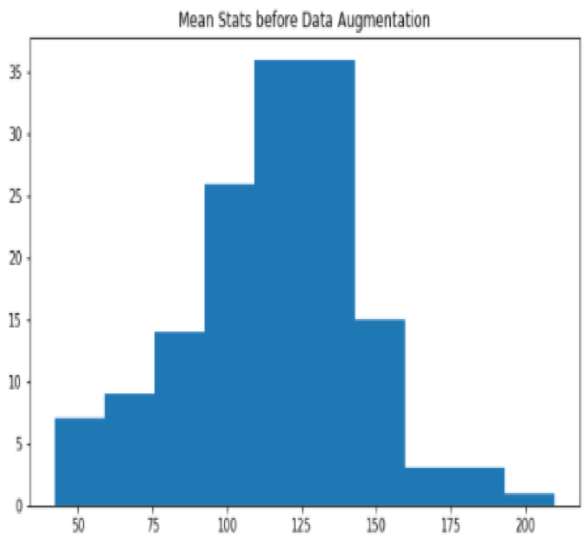

Original Image

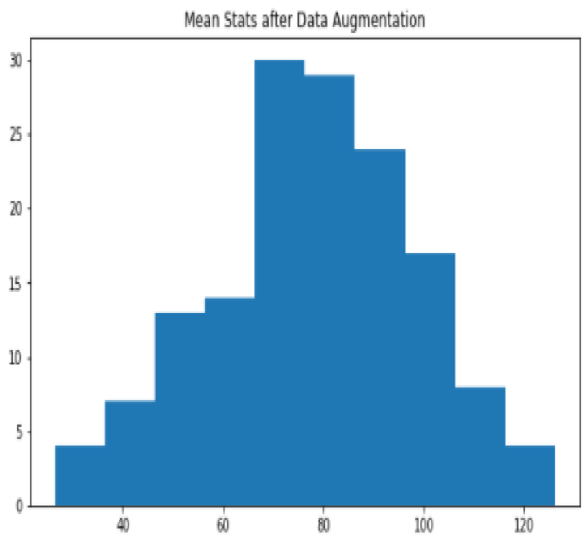

Gray Scale Image


Binarizing

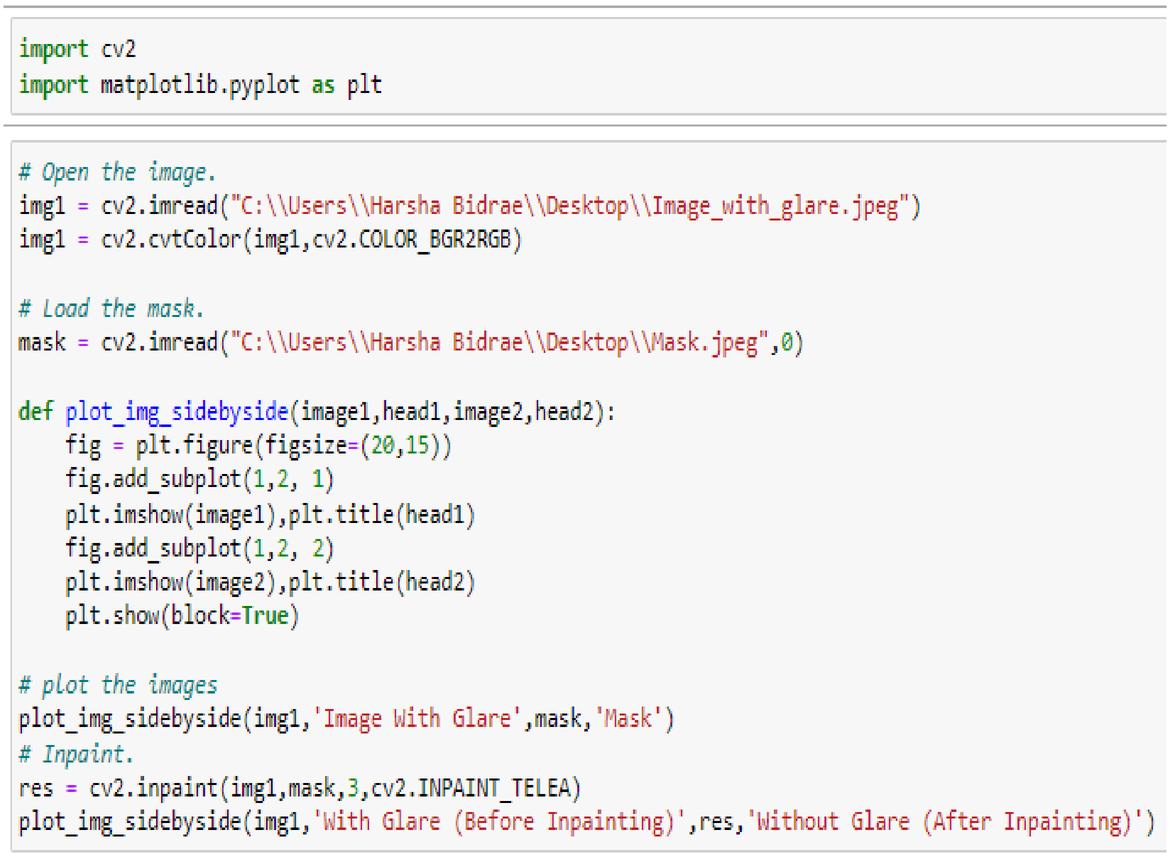
- **1B** On a subset of 150 images from PASCAL VOC dataset, compute mean statistics and plot the histogram. Perform data augmentation on these 150 images by applying affine transformation to the images (translation, rotation, scaling etc..), compute mean statistics and plot the histogram of mean statistic again. Compare the histograms before and after the data augmentation

```
import random
mean_val_before = []
mean_val_after = []
mean_var = []
for i in range(150):
    n = random.randint(0,17125)
    mean_var.append(jp + onlyfiles[n])
#print(mean_var)
for img in mean_var:
    pic = cv2.imread(img, 0)
    mean_val_before.append(np.mean(pic))
print(len(mean_val_before))
for img in mean_var:
    raw_img = cv2.imread(img)
    img = cv2.cvtColor(raw_img, cv2.COLOR_BGR2RGB)
    rows, cols, idx = img.shape
    trans = np.float32([[1,0,100],[0,1,50]])
    dst_trans = cv2.warpAffine(img,trans,(cols,rows))
    mean_val_after.append(np.mean(dst_trans))
print(len(mean_val_after))

f = plt.figure(figsize = (20,5))
f.add_subplot(121),plt.hist(mean_val_before),plt.title("Mean Stats before Data Augmentation")
f.add_subplot(122),plt.hist(mean_val_after),plt.title("Mean Stats after Data Augmentation")
```





2) Apply inpainting operation based on the algorithm Fast Marching to remove glare from the below image. The corresponding mask is also given.



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

```
# Open the image.
img1 = cv2.imread("C:\\Users\\Harsha Bidrae\\Desktop\\Image_with_glare.jpeg")
img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)

# Load the mask.
mask = cv2.imread("C:\\Users\\Harsha Bidrae\\Desktop\\Mask.jpeg",0)

def plot_img_sidebyside(image1,head1,image2,head2):
    fig = plt.figure(figsize=(20,15))
    fig.add_subplot(1,2, 1)
    plt.imshow(image1),plt.title(head1)
    fig.add_subplot(1,2, 2)
    plt.imshow(image2),plt.title(head2)
    plt.show(block=True)

# plot the images
plot_img_sidebyside(img1,'Image With Glare',mask,'Mask')
# Inpaint.
res = cv2.inpaint(img1,mask,3,cv2.INPAINT_TELEA)
plot_img_sidebyside(img1,'With Glare (Before Inpainting)',res,'Without Glare (After Inpainting)')
```

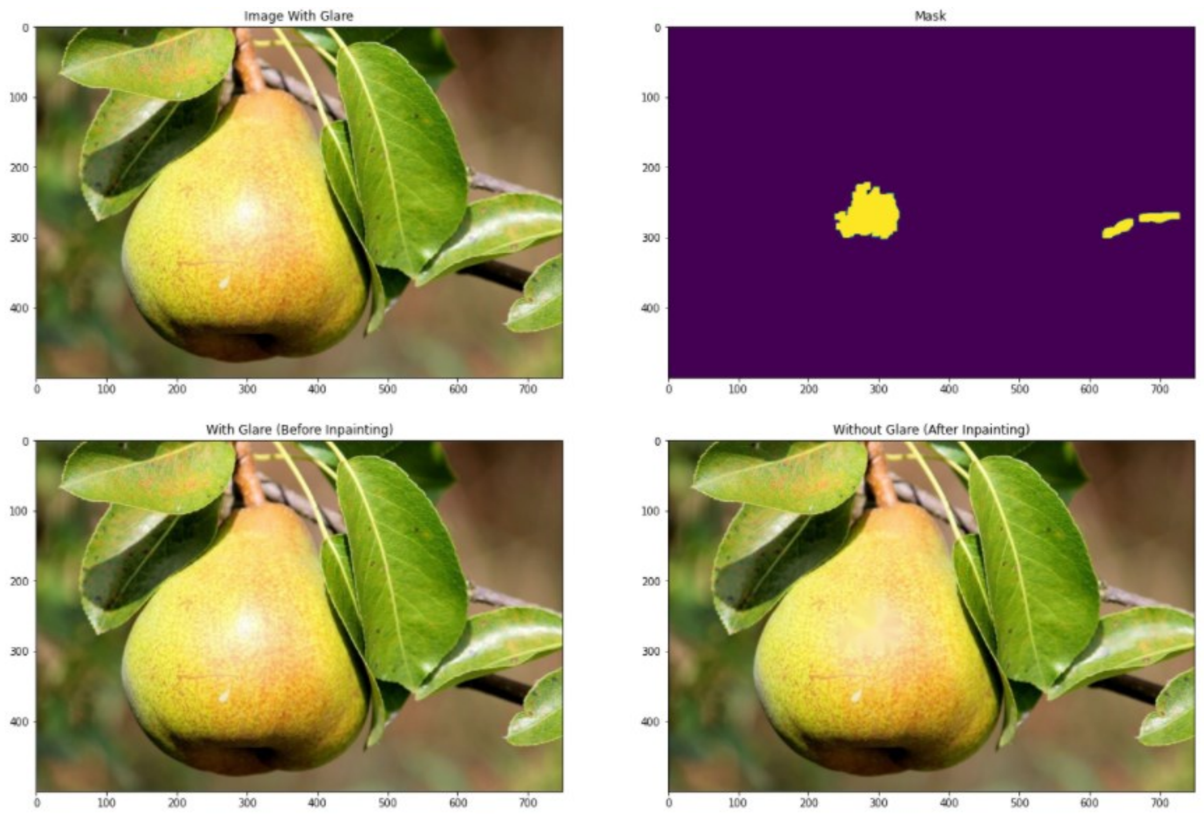
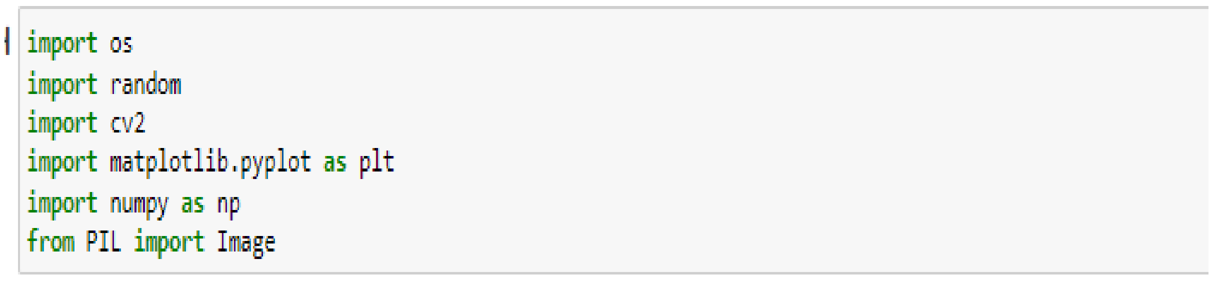


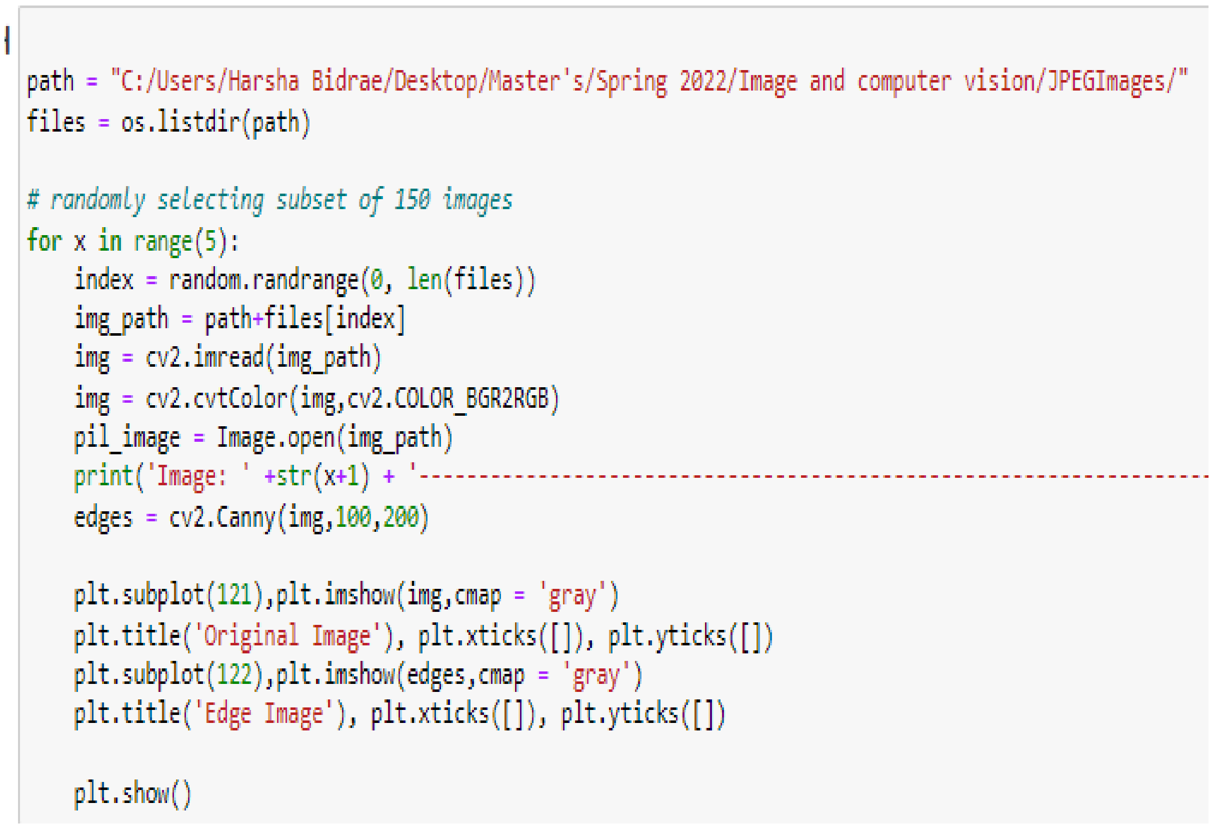
Image With Glare
Mask
With Glare (Before Inpainting)
Without Glare (After Inpainting)
0
100
200
300
400
500
600
700

3) Explain Canny edge detector, apply it on three random images from PASCAL VOC dataset and show the output.

- Canny edge detection is a technique for extracting relevant structural information from various visual objects while reducing the amount of data to be processed considerably. It's been used in a variety of computer vision systems. Canny discovered that the criteria for using edge detection on a variety of vision systems are remarkably similar. As a result, an edge detection solution that meets these requirements can be used in a variety of scenarios.
- Edge detection has a low error rate, which indicates that the detection should catch as many of the image's edges as possible.
- The edge point recognized by the operator should be accurate in locating the edge's centre.

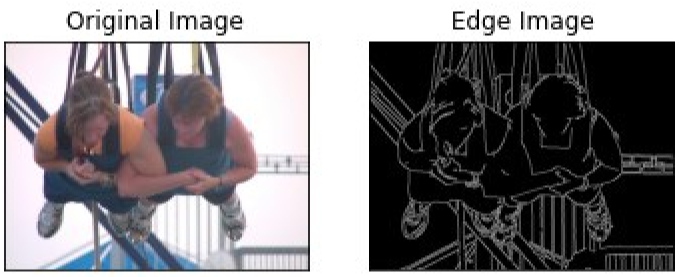


```
import os
import random
import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
```

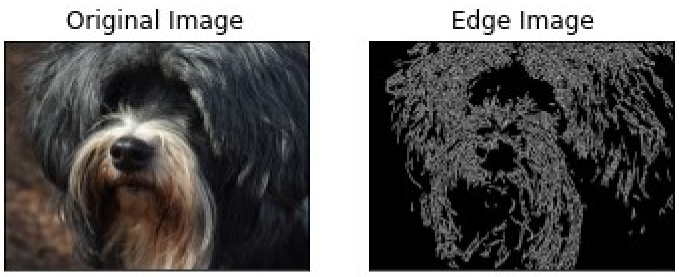


```

path = "C:/Users/Harsha Bidrae/Desktop/Master's/Spring 2022/Image and computer vision/JPEGImages/"
files = os.listdir(path)

# randomly selecting subset of 150 images
for x in range(5):
    index = random.randrange(0, len(files))
    img_path = path+files[index]
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    pil_image = Image.open(img_path)
    print('Image: ' +str(x+1) + '-----------------------------------------------------------------
    edges = cv2.Canny(img,100,200)

    plt.subplot(121),plt.imshow(img,cmap = 'gray')
    plt.title('Original Image'), plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(edges,cmap = 'gray')
    plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

    plt.show()
```

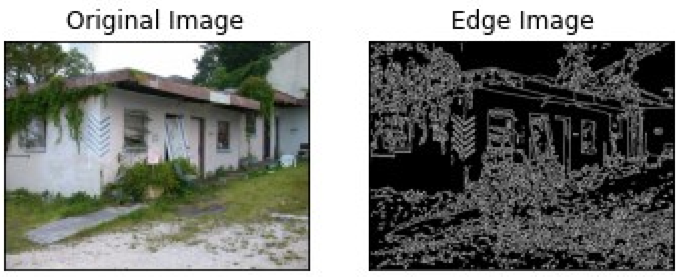
```
Image: 1----------------------------------------
```



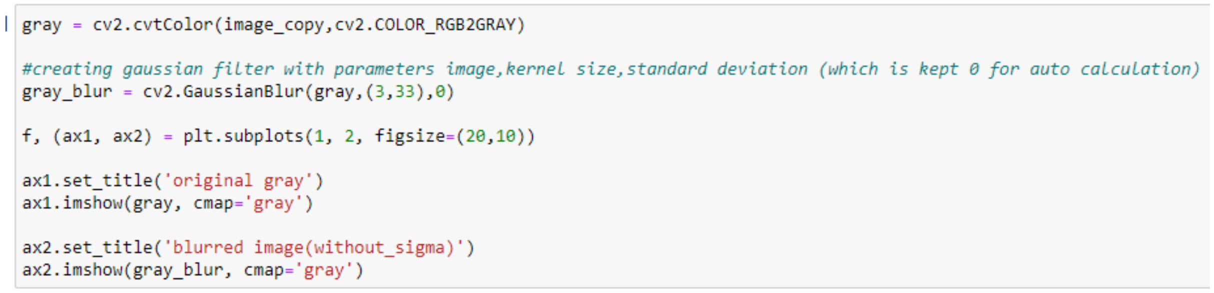
```
Image: 2----------------------------------------
```



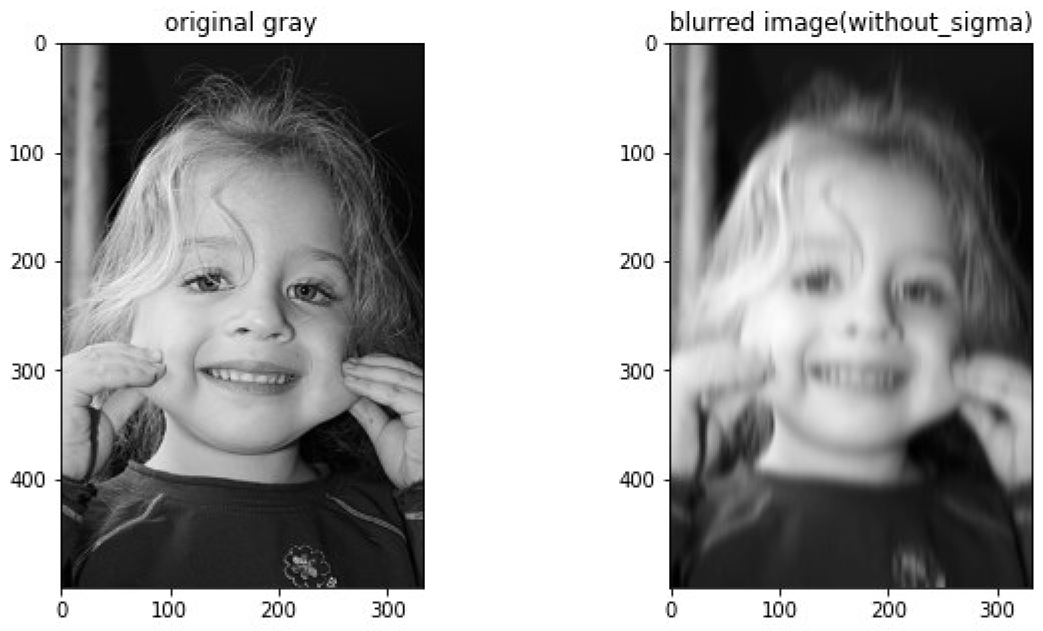
```
Image: 3----------------------------------------
```

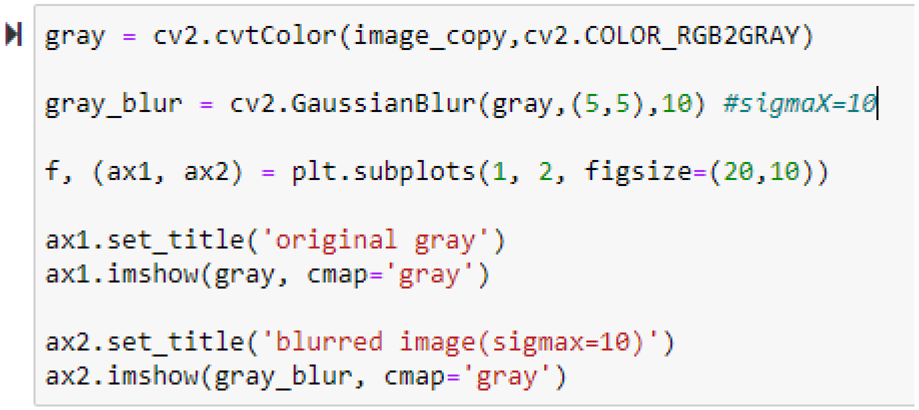


4) Apply Gaussian blur filter on a randomly selected image and choose the optimum value of sigma through empirical evidence. Show all the outputs obtained on varying sigma.

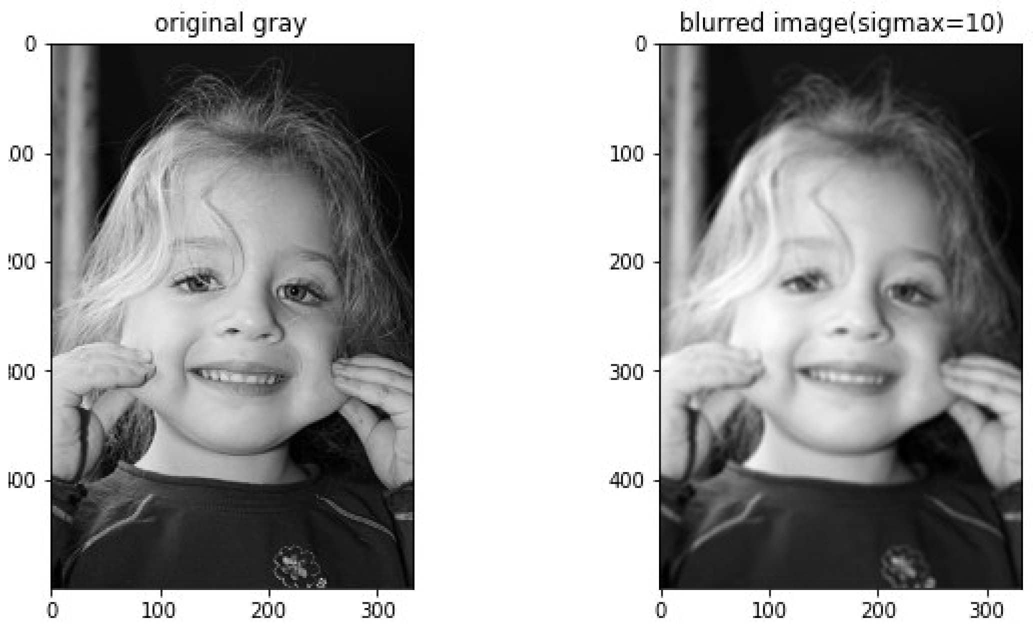


```
gray = cv2.cvtColor(image_copy,cv2.COLOR_RGB2GRAY)

#creating gaussian filter with parameters image,kernel size,standard deviation (which is kept 0 for auto calculation)
gray_blur = cv2.GaussianBlur(gray,(3,33),0)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))

ax1.set_title('original gray')
ax1.imshow(gray, cmap='gray')

ax2.set_title('blurred image(without_sigma)')
ax2.imshow(gray_blur, cmap='gray')
```





```
gray = cv2.cvtColor(image_copy,cv2.COLOR_RGB2GRAY)

gray_blur = cv2.GaussianBlur(gray,(5,5),10) #sigmaX=10

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))

ax1.set_title('original gray')
ax1.imshow(gray, cmap='gray')

ax2.set_title('blurred image(sigmax=10)')
ax2.imshow(gray_blur, cmap='gray')
```



```
gray = cv2.cvtColor(image_copy,cv2.COLOR_RGB2GRAY)

gray_blur = cv2.GaussianBlur(gray,(5,5),8, 15) #sigmaX=10 sigmay=15

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

ax1.set_title('original gray')
ax1.imshow(gray, cmap='gray')

ax2.set_title('blurred image(sigmax=10), sigmay=15')
ax2.imshow(gray_blur, cmap='gray')
```