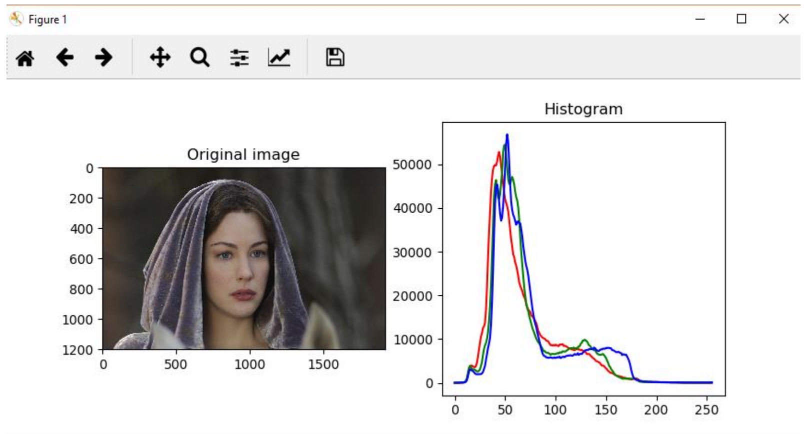
Q1

a. Histogram Computation

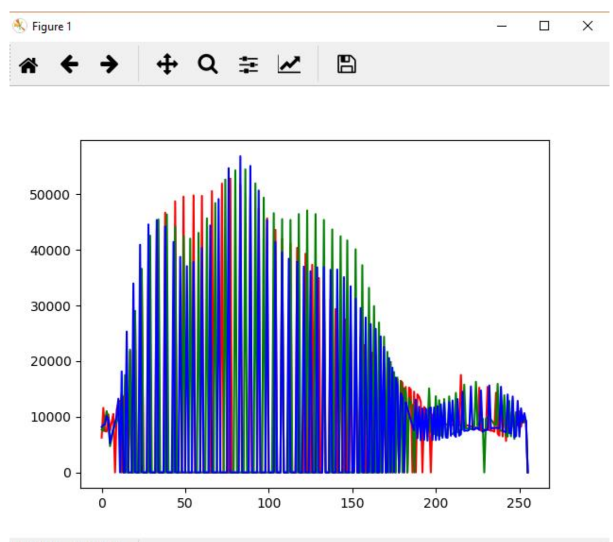
```
import cv2 as cv
from matplotlib import pyplot as plt
def histogram(img):
    global j
    global m
    global n
    j += 1
    print(j)
    colors = ['r','g','b']
    histList = []
    for i in range(256):

histList.append((img.ravel().tolist().count(i)))
    axarr[1].plot(histList, color=colors[j-1])
    axarr[1].set_title('Histogram')
j = 0
img = cv.imread('images/im02.png',
cv.IMREAD_COLOR)
red, green, blue =cv.split(img)
img = cv.cvtColor(img,cv.COLOR_RGB2BGR)
m = img.shape[0]
n = img.shape[1]
f, axarr = plt.subplots(1, 2)
axarr[0].imshow(img)
axarr[0].set_title("Original image")
histogram(red)
histogram(green)
histogram(blue)
plt.show()
```



Histogram is calculated using histogram function, and plotted the histograms for each color channels in subplots.

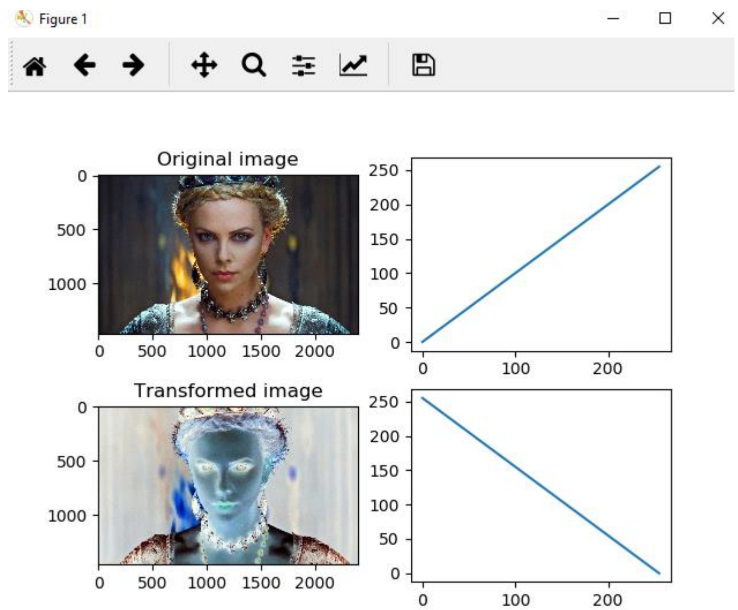
b. Histogram equalization

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
image = cv.imread('images/im02.png', cv.IMREAD_COLOR)
red, green, blue = cv.split(image)  # splitting image in to three
red2 = np.reshape(red, (1, np.product(red.shape)))[0]
green2 = np.reshape(green, (1, np.product(green.shape)))[0]  #
making 1D array of the image
blue2 = np.reshape(blue, (1, np.product(blue.shape)))[0]
imagemap = [red2, green2, blue2]
mn = image.shape[0] * image.shape[1]
for count in range(0, 3):
    unique, nk = np.unique(imagemap[count], return_counts=True)
    prnk = nk / mn
    newvalue = np.zeros(256)
    cumvalue = 0
    for i, x in enumerate(prnk):
        cumvalue += x
        newvalue[i] = 255 * cumvalue
    for x in range(0, image.shape[0]):
        for y in range(0, image.shape[1]):
            pval = image[:, :, count][x, y]
            image[:, :, count][x, y] = newvalue[pval]
color = ("r", "g", "b")
for e, c in enumerate(color):
    hist = cv.calcHist([image[:, :, e]], [0], None, [256], [0, 256])
    plt.plot(hist, color=c)

plt.show()
```



In histogram equalization for each three color channels equalization is done and plotted in the same graph.

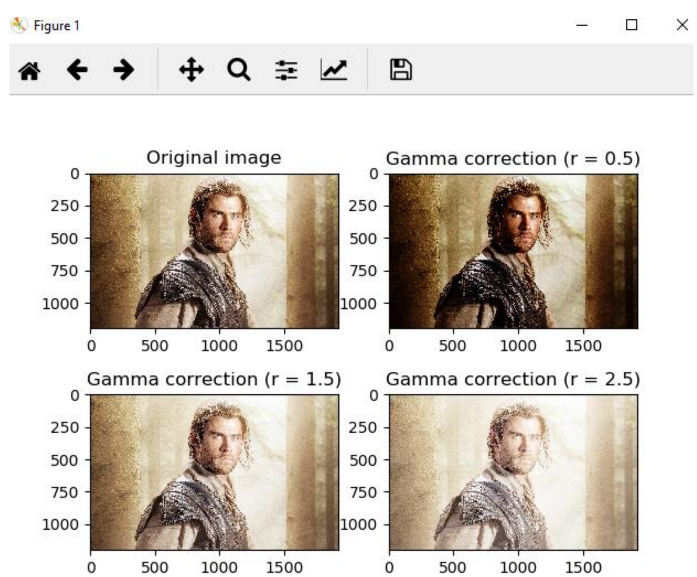
c. Intensity Transformation




```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img1= cv.imread('images/im03.png', cv.IMREAD_COLOR)
img = cv.cvtColor(img1,cv.COLOR_RGB2BGR)
f, axarr = plt.subplots(2, 2)
axarr[0, 0].imshow(img, cmap='gray')
axarr[0, 0].set_title("Original image")
x = np.arange(0, 256)
axarr[0, 1].plot(x)
y = np.arange(255,-1,-1)
axarr[1, 1].plot(y)
axarr[1, 0].imshow(cv.LUT(img, y),cmap='gray')
axarr[1, 0].set_title("Transformed image")
plt.show()
```

The intensity of the original image has been inverted and got a negative effect

d. Gamma Correction




```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('images/im05.png', cv.IMREAD_COLOR)
img1 = cv.cvtColor(img, cv.COLOR_RGB2BGR)
f, axarr = plt.subplots(2, 2)
axarr[0, 0].imshow(img1, cmap='gray')
axarr[0, 0].set_title("Original image")
gamma = 0.5
table = np.array([(i/255.0)**(1/gamma)*255.0 for i in
(np.arange(0, 256))]).astype('uint8')
axarr[0, 1].imshow(cv.LUT(img1, table), cmap='gray')
axarr[0, 1].set_title("Gamma correction (r = 0.5)")
gamma = 1.5
table = np.array([(i/255.0)**(1/gamma)*255.0 for i in
(np.arange(0, 256))]).astype('uint8')
axarr[1, 0].imshow(cv.LUT(img1, table), cmap='gray')
axarr[1, 0].set_title("Gamma correction (r = 1.5)")
gamma = 2.5
table = np.array([(i/255.0)**(1/gamma)*255.0 for i in
(np.arange(0, 256))]).astype('uint8')
axarr[1, 1].imshow(cv.LUT(img1, table), cmap='gray')
axarr[1, 1].set_title("Gamma correction (r = 2.5)")
plt.show()
```

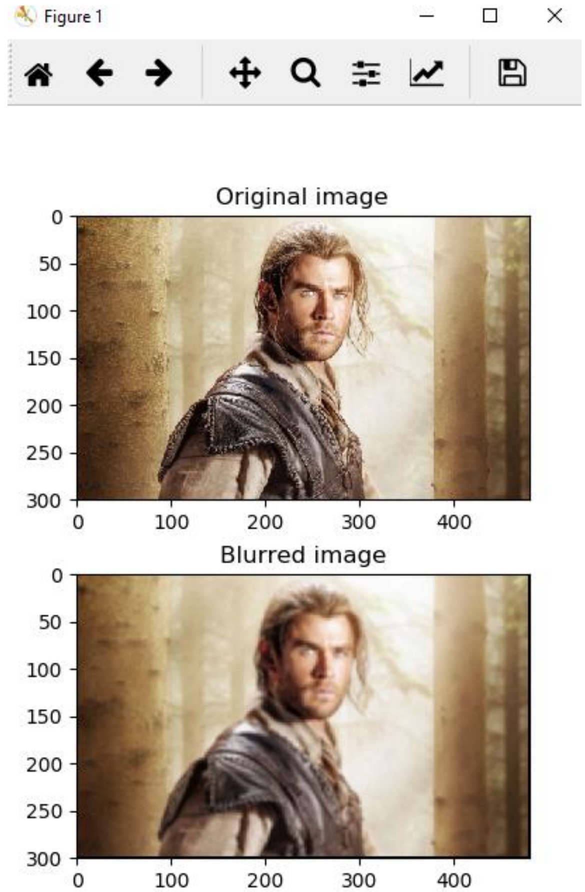
Gamma correction is done for r = 0.5, r =1.5 and r = 2. When r = 1, the original image can be obtained.

e. Gaussian Smoothing

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
import math
def gaussianKernel(size, sigma):
    gaussKernel = np.zeros((size, size), dtype=np.float32)
    for i in range(0, size):
        for j in range(0, size):
            x = i - (math.floor(size / 2))
            y = j - (math.floor(size / 2))
```

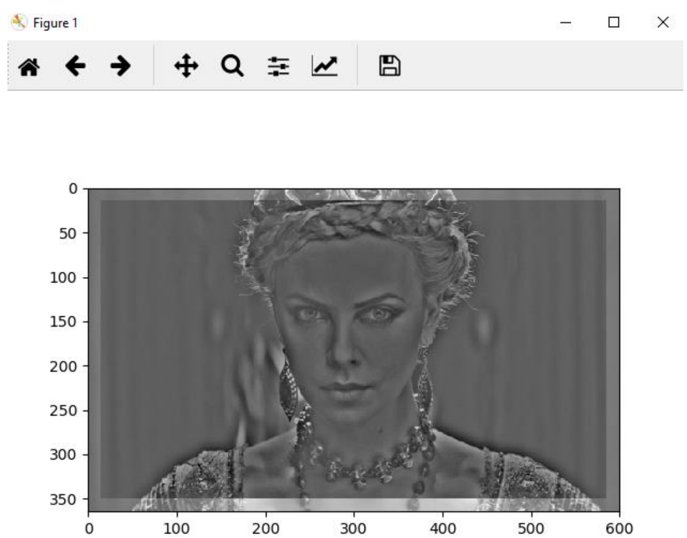
The Gaussian kernel has been generated in gaussianKernal function and the image is convolved with the kernel in filter function. Here kernel size = 5 and sigma = 1

```
            gaussKernel[i][j] = (1 / (2 * math.pi * (sigma ** 2))) *
math.exp((-1 / (2 * math.pi * (sigma ** 2))) * ((x ** 2) + (y **
2)))
    return gaussKernel
def filter(image, kernel):
    assert kernel.shape[0] % 2 == 1 and kernel.shape[1] % 2 == 1
    k_hh, k_hw = math.floor(kernel.shape[0] / 2),
math.floor(kernel.shape[1] / 2)
    h, w = image.shape
    image_float = cv.normalize(image.astype('float'), None, 0.0,
1.0, cv.NORM_MINMAX)
    result = np.zeros(image.shape, 'float')
    for i in range(k_hh, h - k_hh):
        for j in range(k_hw, w - k_hw):
            result[i, j] = np.dot(image_float[i - k_hh: i + k_hh +
1, j - k_hw: j + k_hw + 1].flatten(),
                                  kernel.flatten())
    result = cv.normalize(result.astype('float'), None, 0.0, 1.0,
cv.NORM_MINMAX)
    result = result * 255.0
    return result.astype(np.uint8)
img = cv.imread('images/im05small.png', cv.IMREAD_COLOR)
blue, red, green = cv.split(img)
kernel = gaussianKernel(5, 1)
imgb = filter(blue, kernel)
imgr = filter(red,kernel)
imgg = filter(green,kernel)
filtered_img = cv.merge((imgb, imgr, imgg))
f, axarr = plt.subplots(2)
axarr[0].imshow(cv.cvtColor(img,cv.COLOR_RGB2BGR))
axarr[0].set_title("Original image")
axarr[1].imshow(cv.cvtColor(filtered_img,cv.COLOR_RGB2BGR))
axarr[1].set_title("Blurred image")
plt.show()
```



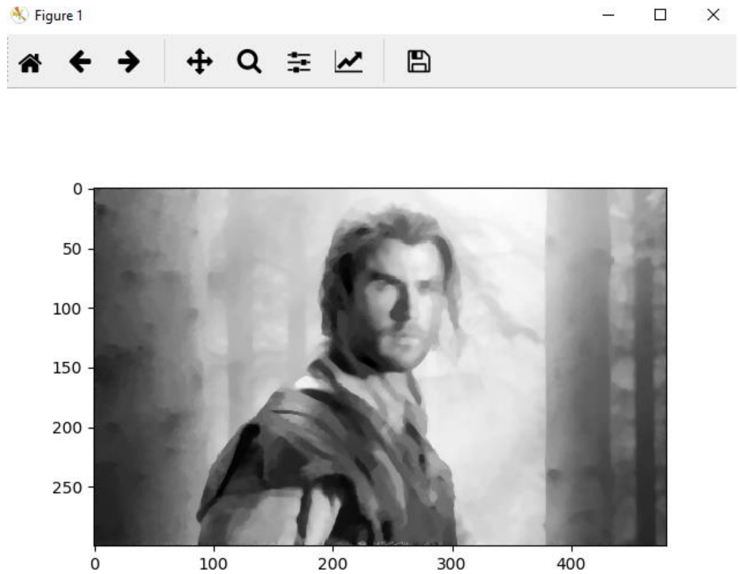
f. Unsharp masking

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
import math
image = cv.imread('images/im03small.png',cv.IMREAD_GRAYSCALE)
def filter2d(image, kernel):
    assert kernel.shape[0]%2 == 1 and kernel.shape[1]%2 == 1
    k_hh, k_hw = math.floor(kernel.shape[0]/2),
math.floor(kernel.shape[1]/2)
    h, w = image.shape
    image_float = cv.normalize(image.astype('float'), None,
0.0, 1.0, cv.NORM_MINMAX)
    result = np.zeros(image.shape, 'float')
    for i in range(k_hh, h - k_hh):
        for j in range(k_hw, w - k_hw):
            result[i,j] = np.dot(image_float[i-k_hh:i + k_hh
+ 1, j - k_hw : j + k_hw + 1].flatten(), kernel.flatten())
    return result
def gaus(size,sigma):
    shape=(size,size)
    x, y = [edge /2 for edge in shape]
    x=int(x)
    y=int(y)
    grid = np.array([[((i**2+j**2)/(2.0*sigma**2)) for i in range(-x, x+1)] for j in range(-y, y+1)])
    g_filter = np.exp(-grid)/(2*np.pi*sigma**2)
    g_filter /= np.sum(g_filter)
    return g_filter
g_filter=gaus(31,10)
bimage=filter2d(image,g_filter)
image = cv.normalize(image.astype('float'), None, 0.0, 1.0, cv.NORM_MINMAX)
bimage = cv.normalize(bimage.astype('float'), None, 0.0, 1.0, cv.NORM_MINMAX)
image3=image-bimage
image3=image3*255.0
image3.astype(np.uint8)
plt.imshow(image3,cmap="gray")
plt.show()
```



First the original image is blurred using Gaussian smoothing method. Then the original image is subtracted by the blurred image.

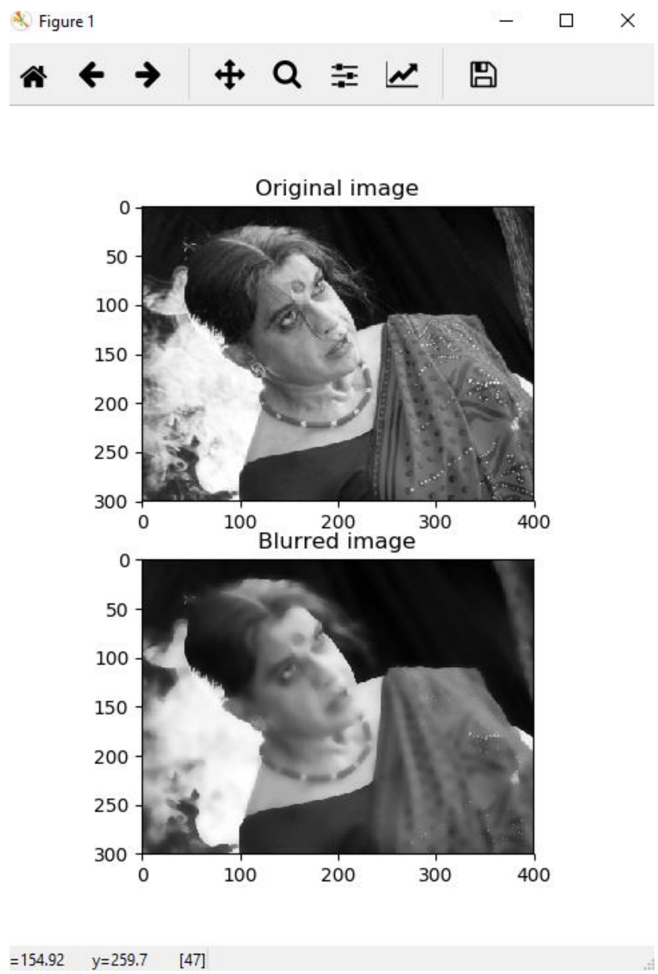
g. Median Filter




```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
image = cv.imread('images/im05small.png',
cv.IMREAD_GRAYSCALE)
kernal = 5
hs = int(kernal / 2)
for x in range(hs, image.shape[0] - hs - 1):
    for y in range(hs, image.shape[1] - hs - 1):
        l = []
        for a in range(x - hs, x + hs + 1):
            for b in range(y - hs, y + hs + 1):
                l.append(image[a, b])
        l.sort()
        image[x, y] = l[int(len(l) / 2)]
plt.imshow(image, cmap="gray")
plt.show()
```

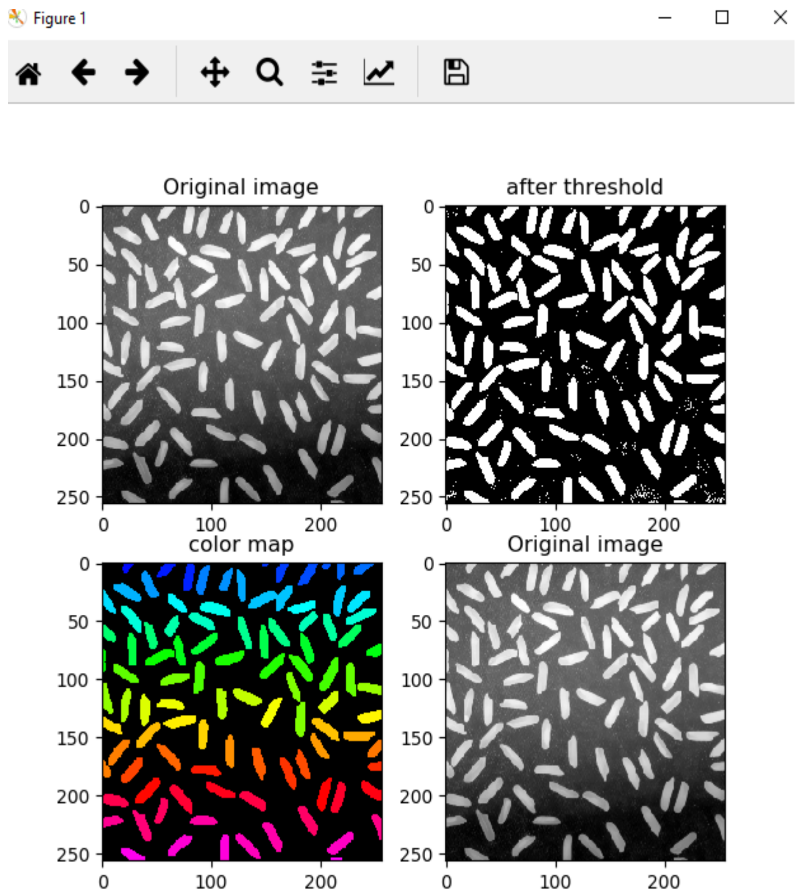
h. Bilateral filter

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
image=cv.imread('images/im09small.png',cv.IMREAD_GR
AYSCALE)
planes=cv.split(image)
def gaussian_filter(shape, sigma):
    x=shape//2
    y=x
    grid = np.array([[((i**2+j**2)/(2.0*sigma**2))
for i in range(-x, x+1)] for j in range(-y, y+1)])
    g_filter = np.exp(-grid)/(2*np.pi*sigma**2)
    g_filter /= np.sum(g_filter)
    return g_filter
def BilateralFilter(image,sigma1,sigma2,size):
    mid=size//2
    kernelSpace=gaussian_filter(size,sigma1)

FilteredImage=np.zeros((len(image),len(image[0])))
    for i in range(mid,len(image)-mid-1):
        for j in range(mid,len(image[i])-mid-1):
            kernelCol=np.zeros((size,size))
            kernelCol=np.abs(image[i][j]-image[i-
mid:i+mid+1,j-mid:j+mid+1])
            kernelCol=np.exp(-
(np.square(kernelCol)/(2*sigma2**2)))

bil_kernel=np.multiply(kernelSpace,kernelCol)

FilteredImage[i][j]=(np.sum(np.multiply(bil_kernel,
image[i-mid:i+mid+1,j-
mid:j+mid+1])))/np.sum(bil_kernel)
    return FilteredImage.astype('uint8')
f, axarr = plt.subplots(2)
axarr[0].imshow(cv.cvtColor(image,cv.COLOR_RGB2BGR))
axarr[0].set_title("Original image")
axarr[1].imshow(cv.cvtColor(newimage,cv.COLOR_RGB2BGR))
axarr[1].set_title("Blurred image")
plt.show()
```



Q2. Rice Grain

```python
import numpy as np
import cv2 as cv
import math

import matplotlib.pyplot as plt
image = cv.imread('images/rice.png',
cv.IMREAD_GRAYSCALE)
image=cv.bilateralFilter(image,9,7,7)
thr = cv.adaptiveThreshold(image, 255,
cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, 61, 1)

kernel = np.ones((3,3), np.uint8)
imgEro = cv.erode(thr, kernel, iterations=1)
imgDil = cv.dilate(imgEro, kernel,
iterations=1)

obj, labels = cv.connectedComponents(imgDil)
rice_count=obj-1 #remove background
#give different colors to each object
hue=np.uint8(150*labels/np.max(labels))
blank=255*np.ones_like(hue)

labeled=cv.merge([hue,blank,blank])
labeled=cv.cvtColor(labeled,cv.COLOR_HSV2BGR)
labeled[hue==0]=0

f, axarr = plt.subplots(2, 2)
axarr[0, 0].imshow(image,cmap='gray')
axarr[0, 0].set_title("Original image")

axarr[0, 1].imshow(thr,cmap='gray')
axarr[0, 1].set_title("after threshold")

axarr[1, 0].imshow(labeled,cmap='gray')
axarr[1, 0].set_title("color map")

axarr[1, 1].imshow(image,cmap='gray')
axarr[1, 1].set_title("Original image")
plt.show()
```



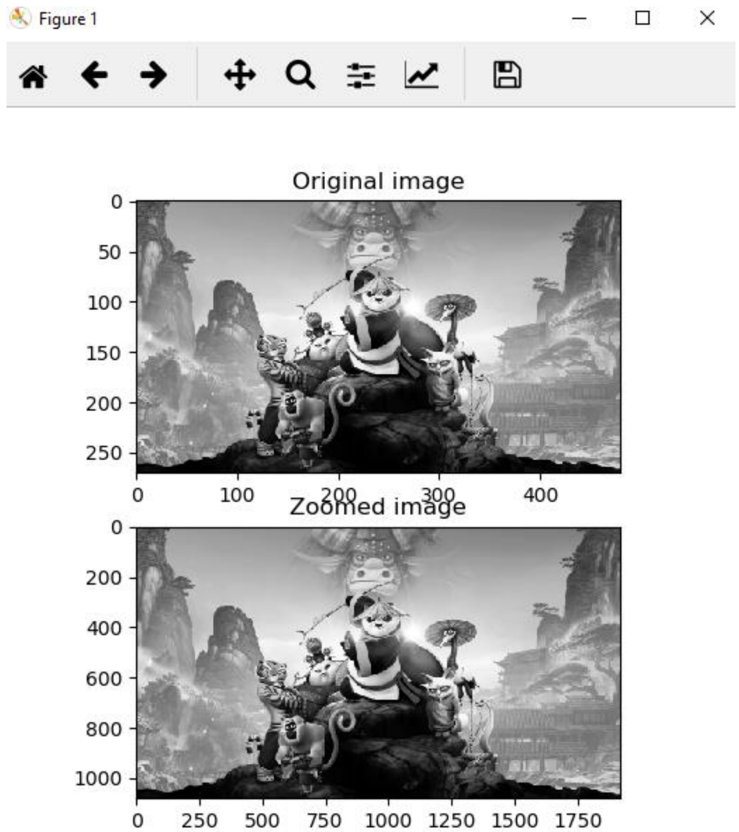
First the image is filtered using the bilateral function to reduce the noise in the image. Then the image is turned into a black and white image by using a threshold intensity value. The color mapping is done and counts the rice grains.

The rice grain count becomes **97** in this method.

Q3

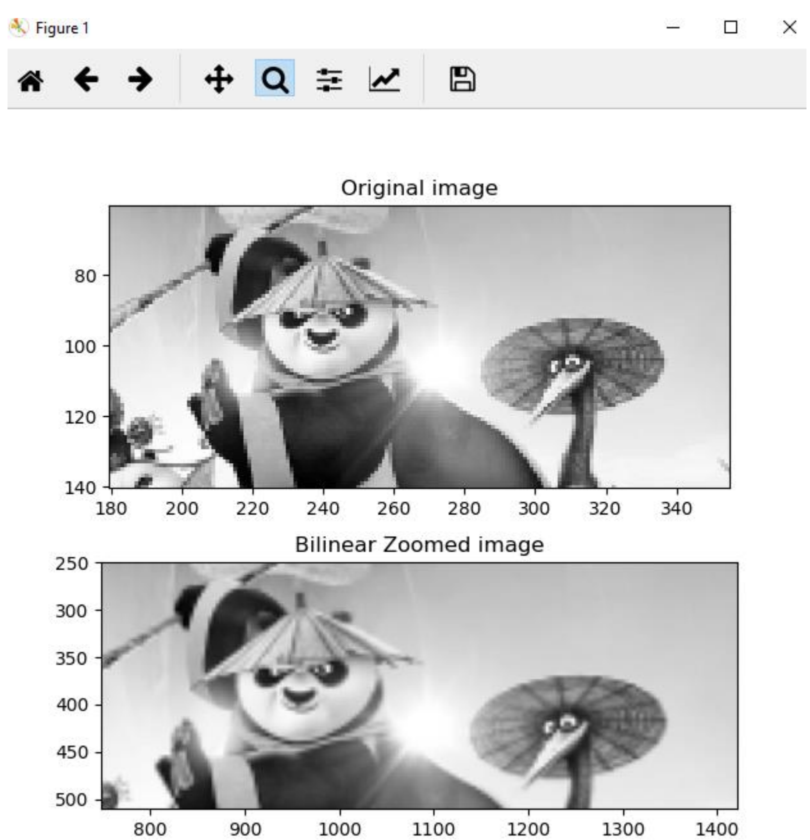
a. Nearest-neighbor

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread("images/im01small.png",
cv.IMREAD_GRAYSCALE)
print(img.shape)
n = img.shape[0]
m = img.shape[1]
scaleFactor = 4
zeroImg = np.zeros((n*scaleFactor,
m*scaleFactor))
for i in range(n):
    for j in range(m):
        for k in range(scaleFactor):
            for l in range(scaleFactor):
                zeroImg[scaleFactor*i+k,
scaleFactor*j+l] = img[i, j]
f, axarr = plt.subplots(2)
axarr[0].imshow(img, cmap='gray')
axarr[0].set_title("Original image")
axarr[1].imshow(zeroImg, cmap='gray')
axarr[1].set_title("Zoomed image")
plt.show()
```



b. Bilinear interpolation

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
img = cv.imread("images/im06small.png",
cv.IMREAD_GRAYSCALE)
factor = 4
n = img.shape[1]
m = img.shape[0]
newImg = np.zeros((m * factor, n * factor))
for y in range(0, m):
    for x in range(0, n):
        newImg[y * factor, x * factor] =
img[y, x]
for y in range(0, m * factor, factor):
    for x in range(0, (n - 1) * factor,
factor):
        for a in range(x + 1, x + factor):
            newImg[y, a] = int(newImg[y, x] +
(newImg[y, x + factor] - newImg[y, x]) /
factor * (a - x))
for x in range(0, n * factor):
    for y in range(0, (m - 1) * factor,
factor):
        for a in range(y + 1, y + factor):
            newImg[a, x] = int(newImg[y, x] +
(newImg[y + factor, x] - newImg[y, x]) /
factor * (a - y))
f, axarr = plt.subplots(2)
axarr[0].imshow(img, cmap='gray')
axarr[0].set_title("Original image")
axarr[1].imshow(newImg, cmap='gray')
axarr[1].set_title("Bilinear Zoomed image")
plt.show()
```



The bilinear interpolation has given a better quality image than the image created with nearest-neighbor method.