Part-I

Define the following in two sentences or less:

a. Digital Image: An image may be defined as a two-dimensional function, f(x, y), where x and y are spatial (plane) coordinates, and the amplitude of f, at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point. When x, y, and the intensity values of f are all finite, discrete quantities, we call the image a digital image.

b) Pixel: a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are called picture elements, image elements, pels, and pixels. Pixel is the term used most widely to denote the elements of a digital image.

c) Sampling: Digitizing the coordinate values is called sampling.

d) Quantization: Digitizing the amplitude values is called quantization.

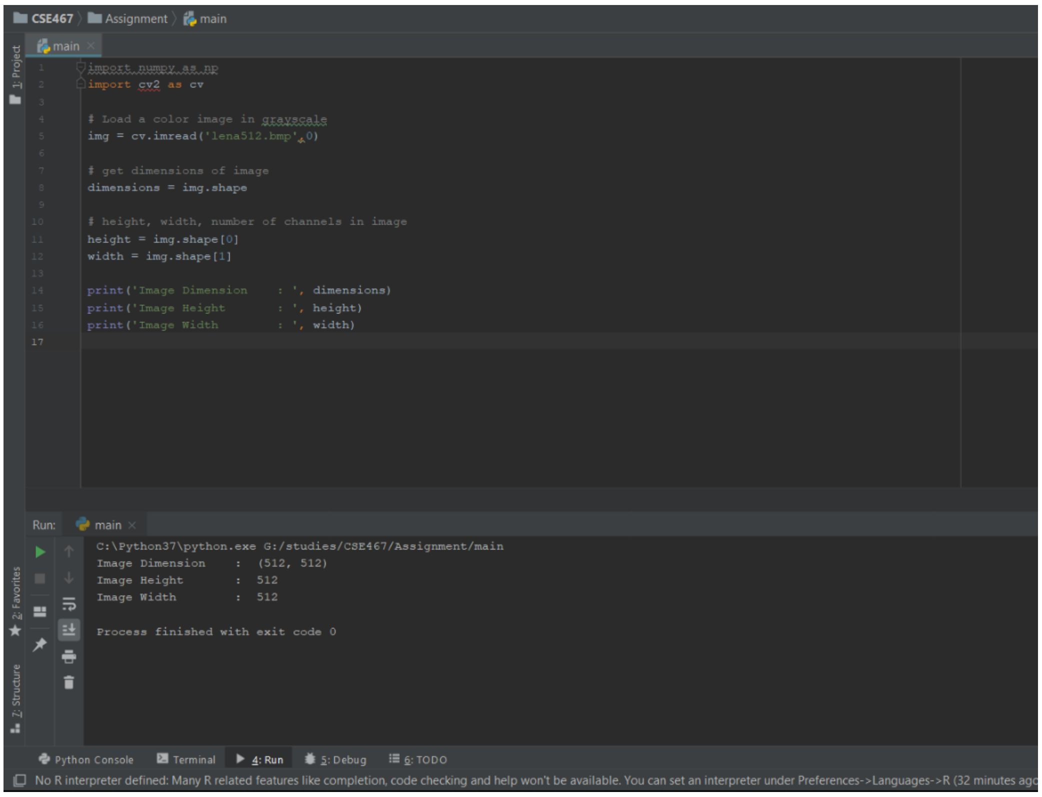
e) Spatial Resolution: spatial resolution is a measure of the smallest discernible detail in an image. Quantitatively, spatial resolution can be stated in several ways, with line pairs per unit distance, and dots (pixels) per unit distance being common measures.

f) Intensity Resolution: refers to the smallest discernible change in intensity level.

Part-II

a. Load the image of Lena in grayscale. What is the dimensionality of this image? How many bits are used to encode a single pixel in this image?

Ans:



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

# Load a color image in grayscale
img = cv.imread('lena512.bmp',0)

# get dimensions of image
dimensions = img.shape

# height, width, number of channels in image
height = img.shape[0]
width = img.shape[1]

print('Image Dimension    : ', dimensions)
print('Image Height       : ', height)
print('Image Width        : ', width)
```

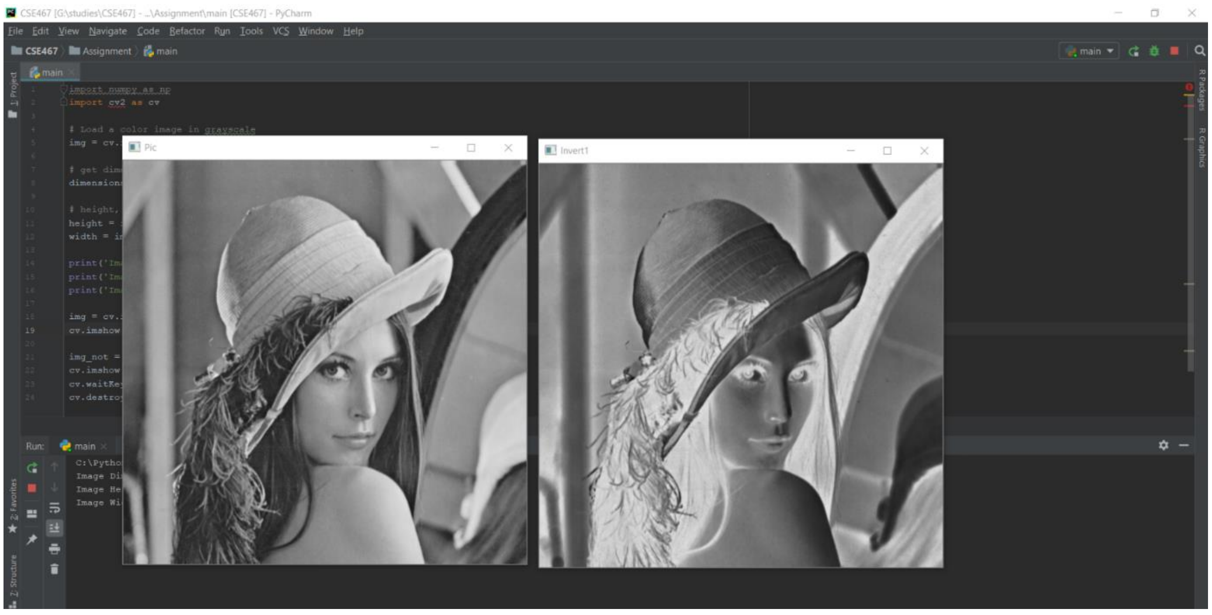
```
C:\Python37\python.exe G:/studies/CSE467/Assignment/main
Image Dimension    :  (512, 512)
Image Height       :  512
Image Width        :  512

Process finished with exit code 0
```

No R interpreter defined: Many R related features like completion, code checking and help won't be available. You can set an interpreter under Preferences->Languages->R (32 minutes ago

8 bits are used to encode a single pixel in any image.

b. Generate an Image negative of Lena.

Ans:

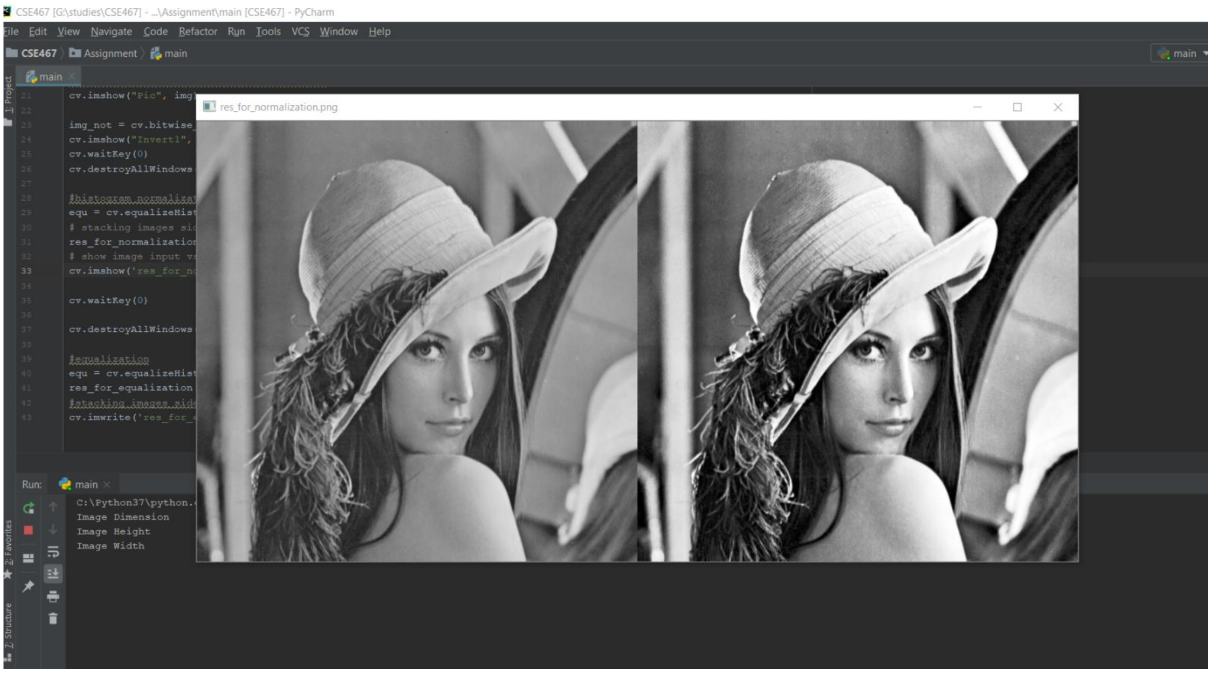

CSE467 [G:\studies\CSE467] - ...\Assignment\main [CSE467] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSE467 Assignment main
Pic
Invert1
Run: main

```
import numpy as np
import cv2 as cv

# Load a color image in grayscale
img = cv.

# get dim
dimension

# height,
height =
width = i

print('Im
print('Im
print('Im

img = cv.
cv.imshow

img_not =
cv.imshow
cv.waitKe
cv.destro
```

```
C:\Pytho
Image Di
Image He
Image Wi
```

c. Generate the normalized image histogram for Lena, and perform histogram equalization on the image. Comment on the difference in the image after histogram equalization, if any.

Ans:

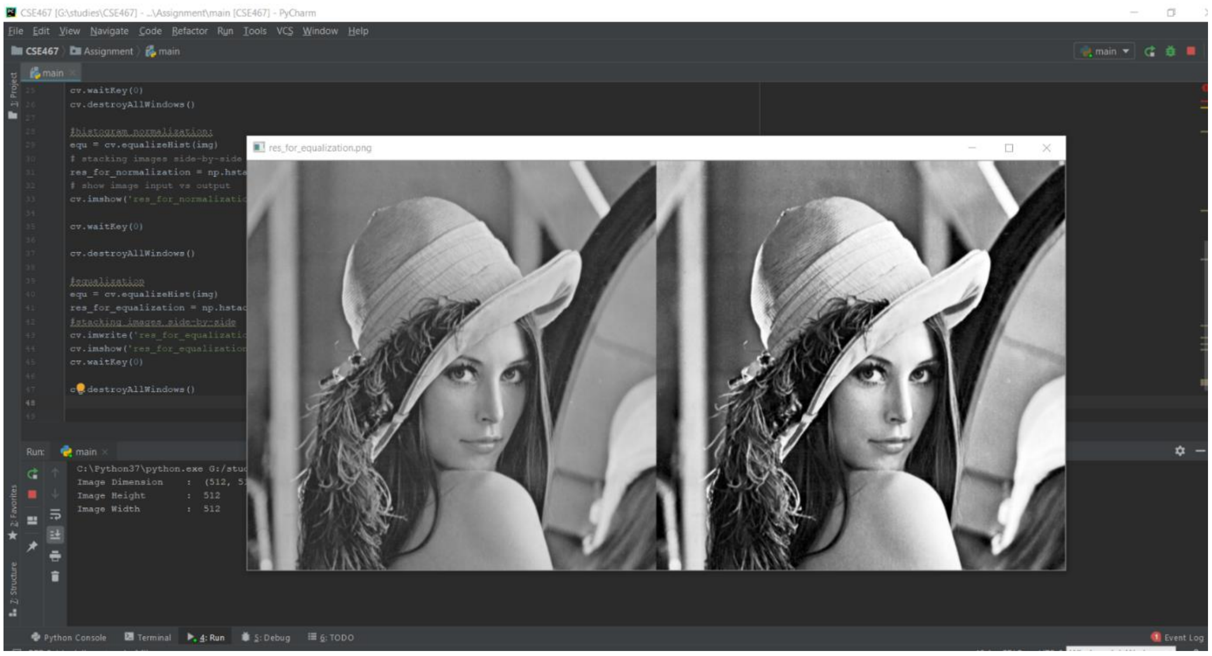

CSE467 [G:\studies\CSE467] - ...\Assignment\main [CSE467] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSE467 Assignment main
res_for_normalization.png
Run: main

```
cv.imshow("Pic", img

img_not = cv.bitwise
cv.imshow("Invert1",
cv.waitKey(0)
cv.destroyAllWindows

#histogram normaliza
equ = cv.equalizeHist
# stacking images si
res_for_normalizatio
# show image input v
cv.imshow('res_for_n

cv.waitKey(0)

cv.destroyAllWindows

#equalization
equ = cv.equalizeHist
res_for_equalization
#stacking images side
cv.imwrite('res_for_
```

```
C:\Python37\python.
Image Dimension
Image Height
Image Width
```

-Normalization

They both produce similar results but use different techniques. The normalize is quite simple, it looks for the maximum intensity pixel (we will use a grayscale example here) and a minimum intensity and then will determine a factor that scales the min intensity to black and the max intensity to white. This is applied to every pixel in the image which produces the final result.

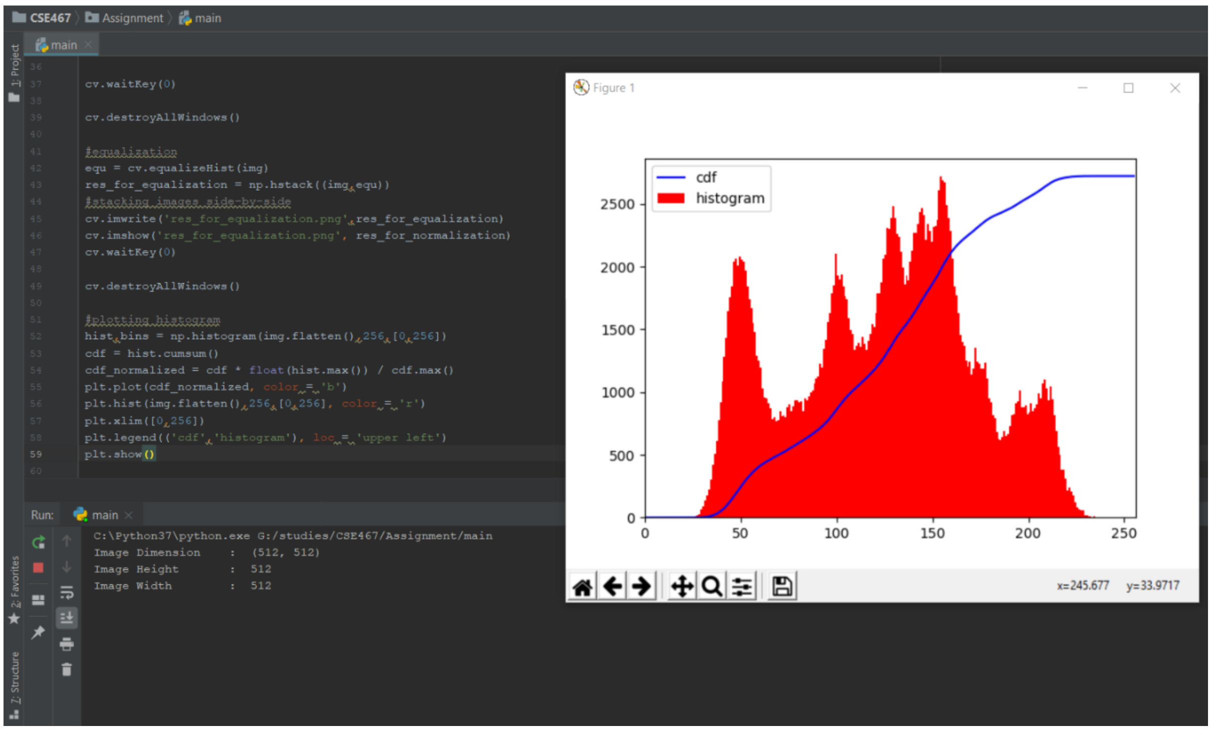

CSE467 [G:\studies\CSE467] - ...\Assignment\main [CSE467] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSE467 Assignment main
res_for_equalization.png
Python Console Terminal 4: Run 5: Debug 6: TODO Event Log

```
cv.waitKey(0)
cv.destroyAllWindows()

#histogram normalization:
equ = cv.equalizeHist(img)
# stacking images side-by-side
res_for_normalization = np.hsta
# show image input vs output
cv.imshow('res_for_normalizatio

cv.waitKey(0)

cv.destroyAllWindows()

#equalization
equ = cv.equalizeHist(img)
res_for_equalization = np.hstac
#stacking images side-by-side
cv.imwrite('res_for_equalizatio
cv.imshow('res_for_equalization
cv.waitKey(0)

cv.destroyAllWindows()
```

```
C:\Python37\python.exe G:/stuc
Image Dimension     :  (512, 5
Image Height        :  512
Image Width         :  512
```

-Equalization

The equalize will attempt to produce a histogram with equal amounts of pixels in each intensity level. This can produce unrealistic images since the intensities can be radically distorted but can also produce images very similar to normalization which preserves relative levels which the equalization process does not.



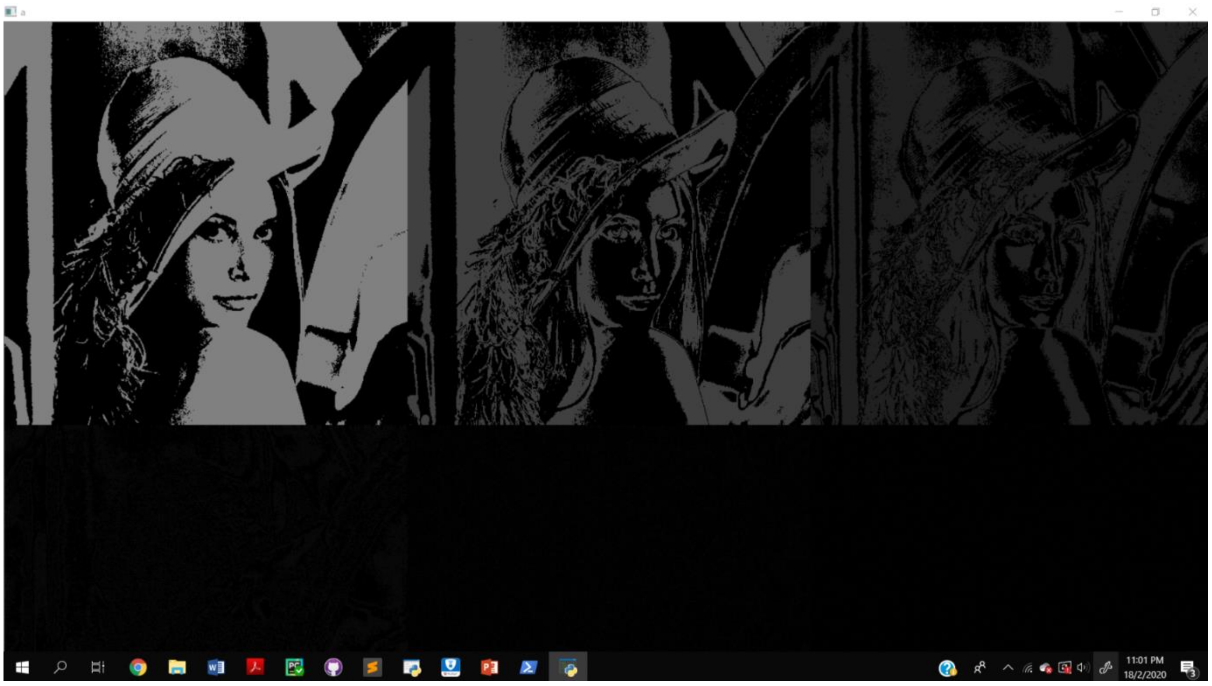
```
cv.waitKey(0)

cv.destroyAllWindows()

#equalization
equ = cv.equalizeHist(img)
res_for_equalization = np.hstack((img,equ))
#stacking images side-by-side
cv.imwrite('res_for_equalization.png',res_for_equalization)
cv.imshow('res_for_equalization.png', res_for_normalization)
cv.waitKey(0)

cv.destroyAllWindows()

#plotting histogram
hist,bins = np.histogram(img.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()
```

```
C:\Python37\python.exe G:/studies/CSE467/Assignment/main
Image Dimension     :  (512, 512)
Image Height        :  512
Image Width         :  512
```

-The histogram

d. Write a function that takes an 8-bit grayscale image as input, and generates 8 1-bit plane images as discussed in lecture. Demonstrate the functionality of your code using the Lena image – your submission should include images of all bit planes. Which bit planes hold the most information, and why?

Ans:



The higher-order planes are typically those that contains the most relevant visual data.

Code:

```
import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

from PIL import Image

from pylab import *




# Load a color image in grayscale

img = cv.imread('lena512.bmp',0)


# get dimensions of image

dimensions = img.shape


# height, width, number of channels in image

height = img.shape[0]

width = img.shape[1]
```

```
print('Image Dimension    : ', dimensions)

print('Image Height       : ', height)

print('Image Width        : ', width)


#finding the invert/ negetive of teh image

cv.imshow("Pic", img)


img_not = cv.bitwise_not(img)

cv.imshow("Invert1", img_not)

cv.waitKey(0)

cv.destroyAllWindows()


#histogram normalization:

equ = cv.equalizeHist(img)

# stacking images side-by-side

res_for_normalization = np.hstack((img, equ))

# show image input vs output

cv.imshow('res_for_normalization.png', res_for_normalization)


cv.waitKey(0)


cv.destroyAllWindows()


#equalization

equ = cv.equalizeHist(img)

res_for_equalization = np.hstack((img,equ))

#stacking images side-by-side

cv.imwrite('res_for_equalization.png',res_for_equalization)

cv.imshow('res_for_equalization.png', res_for_normalization)

cv.waitKey(0)
cv.destroyAllWindows()


#plotting histogram

hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()

cdf_normalized = cdf * float(hist.max()) / cdf.max()

plt.plot(cdf_normalized, color = 'b')

plt.hist(img.flatten(),256,[0,256], color = 'r')

plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()


cdf_m = np.ma.masked_equal(cdf,0)

cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())

cdf = np.ma.filled(cdf_m,0).astype('uint8')




#part c


lst = []

for i in range(img.shape[0]):

   for j in range(img.shape[1]):

       lst.append(np.binary_repr(img[i][j], width=8))  # width = no. of bits


# We have a list of strings where each string represents binary pixel value. To extract bit planes we need to iterate over the strings and store the characters corresponding to bit planes into lists.

# Multiply with 2^(n-1) and reshape to reconstruct the bit image.

eight_bit_img = (np.array([int(i[0]) for i in lst], dtype=np.uint8) * 128).reshape(img.shape[0], img.shape[1])
```

```
seven_bit_img = (np.array([int(i[1]) for i in lst], dtype=np.uint8) * 64).reshape(img.shape[0],
img.shape[1])

six_bit_img = (np.array([int(i[2]) for i in lst], dtype=np.uint8) * 32).reshape(img.shape[0],
img.shape[1])

five_bit_img = (np.array([int(i[3]) for i in lst], dtype=np.uint8) * 16).reshape(img.shape[0],
img.shape[1])

four_bit_img = (np.array([int(i[4]) for i in lst], dtype=np.uint8) * 8).reshape(img.shape[0],
img.shape[1])

three_bit_img = (np.array([int(i[5]) for i in lst], dtype=np.uint8) * 4).reshape(img.shape[0],
img.shape[1])

two_bit_img = (np.array([int(i[6]) for i in lst], dtype=np.uint8) * 2).reshape(img.shape[0],
img.shape[1])

one_bit_img = (np.array([int(i[7]) for i in lst], dtype=np.uint8) * 1).reshape(img.shape[0],
img.shape[1])


# Concatenate these images for ease of display using cv2.hconcat()

finalr = cv.hconcat([eight_bit_img, seven_bit_img, six_bit_img, five_bit_img])

finalv = cv.hconcat([four_bit_img, three_bit_img, two_bit_img, one_bit_img])


# Vertically concatenate

final = cv.vconcat([finalr, finalv])


# Display the images

cv.imshow('a', final)

cv.waitKey(0)

# Combining 4 bit planes

new_img = eight_bit_img+seven_bit_img+six_bit_img+five_bit_img

# Display the image

cv.imshow('a',new_img)

cv.waitKey(0)
```