**Digital Image Processing Lab**

**CEL-445**

**Lab Journal: 4**



Name: OMAR

Class: BCE-07

Enrollment No: 01-132182-024

TASKS:

1. Explanation of Used techniques.

2. Explanation of Input code and Analysis of the Output.

3. Home Task code and explanation.

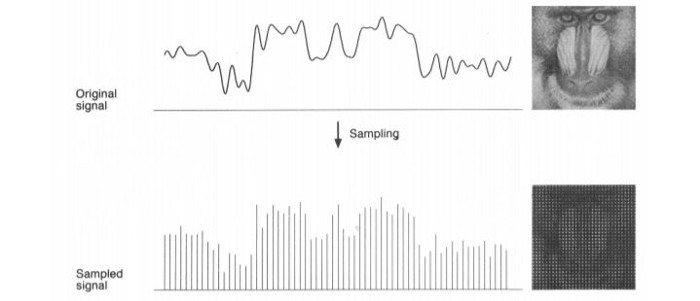
**Sampling:TASK-1:**

Since an analogue image is continuous not just in its co-ordinates (x axis), but also in its amplitude (y axis), so the part that deals with the digitizing of co-ordinates is known as sampling. In digitizing sampling is done on independent variable. In case of equation y = sin(x), it is done on x variable.

When looking at this image, we can see there are some random variations in the signal caused by noise. In sampling we reduce this noise by taking samples. It is obvious that more samples we take, the quality of the image would be better, the noise would be more removed and same happens vice versa. However, if you take sampling on the x axis, the signal is not converted to digital format, unless you take sampling of the y-axis too which is known as quantization.

Sampling has a relationship with image pixels. The total number of pixels in an image can be calculated as Pixels = total no of rows \* total no of columns. For example, let’s say we have total of 36 pixels, that means we have a square image of 6X 6. As we know in sampling, that more samples eventually result in more pixels. So it means that of our continuous signal, we have taken 36 samples on x axis. That refers to 36 pixels of this image. Also the number sample is directly equal to the number of sensors on CCD array.

Here is an example for image sampling and how it can be represented using a graph.

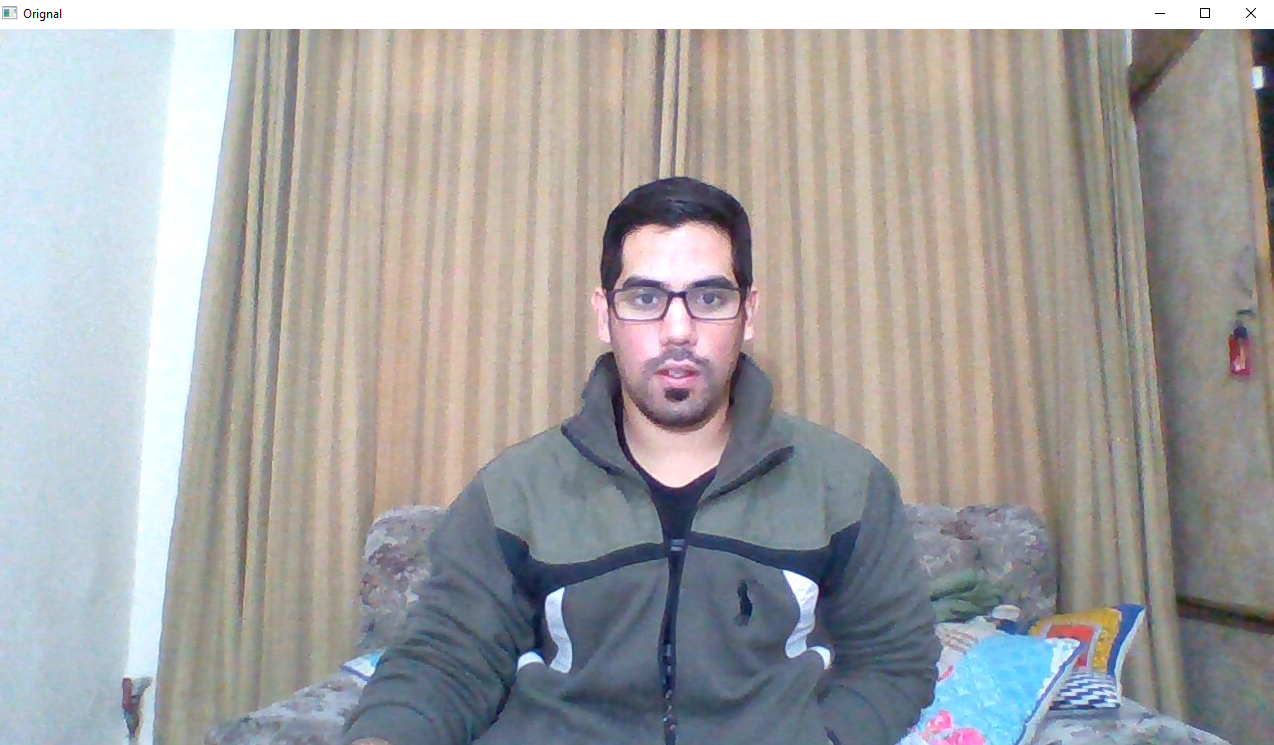


Code:

import numpy as np  
import cv2  
img=cv2.imread("im.jpg")  
cv2.imshow("Orignal",img)  
cv2.waitKey(0)  
  
# to get the height and image  
height=img.shape[0]  
width=img.shape[1]  
print(height,width)  
dimension=img.shape  
print(dimension)  
#to convert sample into 16\*16 area  
numheight=int(height/16)  
numwidth=int(width/16)  
###################################  
#create an image#very imp  
new\_img=np.zeros((height,width,3),np.uint8)  
#to create x-axis of nem\_img--x & y are arrays(1D)  
for i in range(16):  
 y=i\*numheight  
for j in range(16):  
 x=j\*numwidth  
#yeh hum original image sy colors ki value ly rhy hain#loop not required as python understands  
b=img[y,x][0]  
g=img[y,x][1]  
r=img[y,x][2]  
##created image main hum pixel ki values dal rhy hain .  
for n in range(numheight):  
 for m in range(numwidth):  
 new\_img[y+n,x+m][0]=np.uint8(b)  
 new\_img[y + n, x + m][1] = np.uint8(g)  
 new\_img[y + n, x + m][2] = np.uint8(r)  
#display created image  
cv2.imshow("original",img)  
cv2.imshow("sampling(created image)",new\_img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

OUTPUT:

ORIGINAL IMAGE:



SAMPLED IMAGE(16\*16):



SAMPLED IMAGE(32\*32):



SAMPLED IMAGE(64\*64):



***Each time sample resolution is changed, a new type of image is created.***

**Quantization:TASK#2**

Quantization is opposite to sampling because it is done on “y axis” while sampling is done on “x axis”. Quantization is a process of transforming a real valued sampled image to one taking only a finite number of distinct values. Under quantization process the amplitude values of the image are digitized. In simple words, when you are quantizing an image, you are actually dividing a signal into quanta(partitions).

Now let’s see how quantization is done. Here we assign levels to the values generated by sampling process. In the image showed in sampling explanation, although the samples has been taken, but they were still spanning vertically to a continuous range of gray level values. In the image shown below, these vertically ranging values have been quantized into 5 different levels or partitions. Ranging from 0 black to 4 white. This level could vary according to the type of image you want.

There is a relationship between Quantization with gray level resolution. The above quantized image represents 5 different levels of gray and that means the image formed from this signal, would only have 5 different colors. It would be a black and white image more or less with some colors of gray.

When we want to improve the quality of image, we can increase the levels assign to the sampled image. If we increase this level to 256, it means we have a gray scale image.

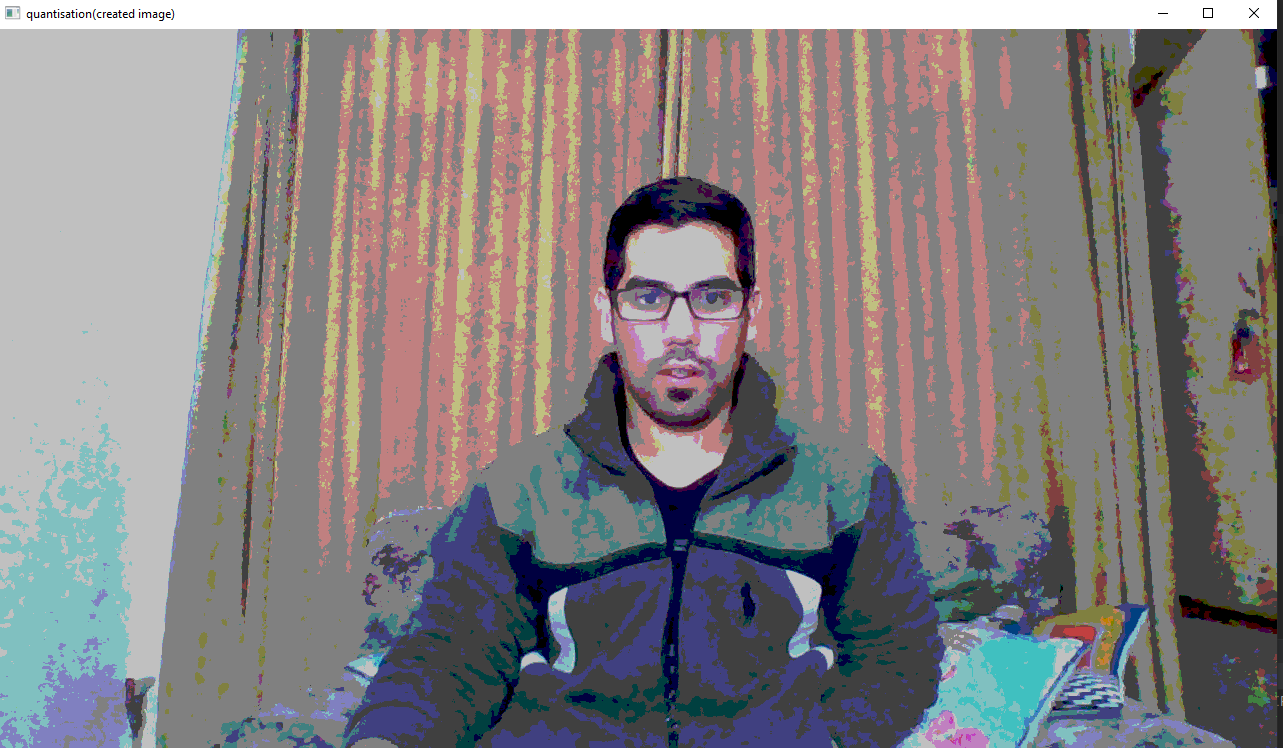
Code:Level2-Quantization

import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
img=cv2.imread("im.jpg",1)  
cv2.imshow("Orignal",img)  
cv2.waitKey(0)  
  
# to get the height and image  
height=img.shape[0]  
width=img.shape[1]  
new\_img=np.zeros((height,width,3),np.uint8)  
for i in range(height):  
 for j in range(width):  
 for k in range(3):  
 if img[i,j][k]<128:  
 gray=0  
 else:  
 gray=128  
 new\_img[i,j][k]=np.uint8(gray)  
  
cv2.imshow("original",img)  
cv2.imshow("quantisation(created image)",new\_img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()



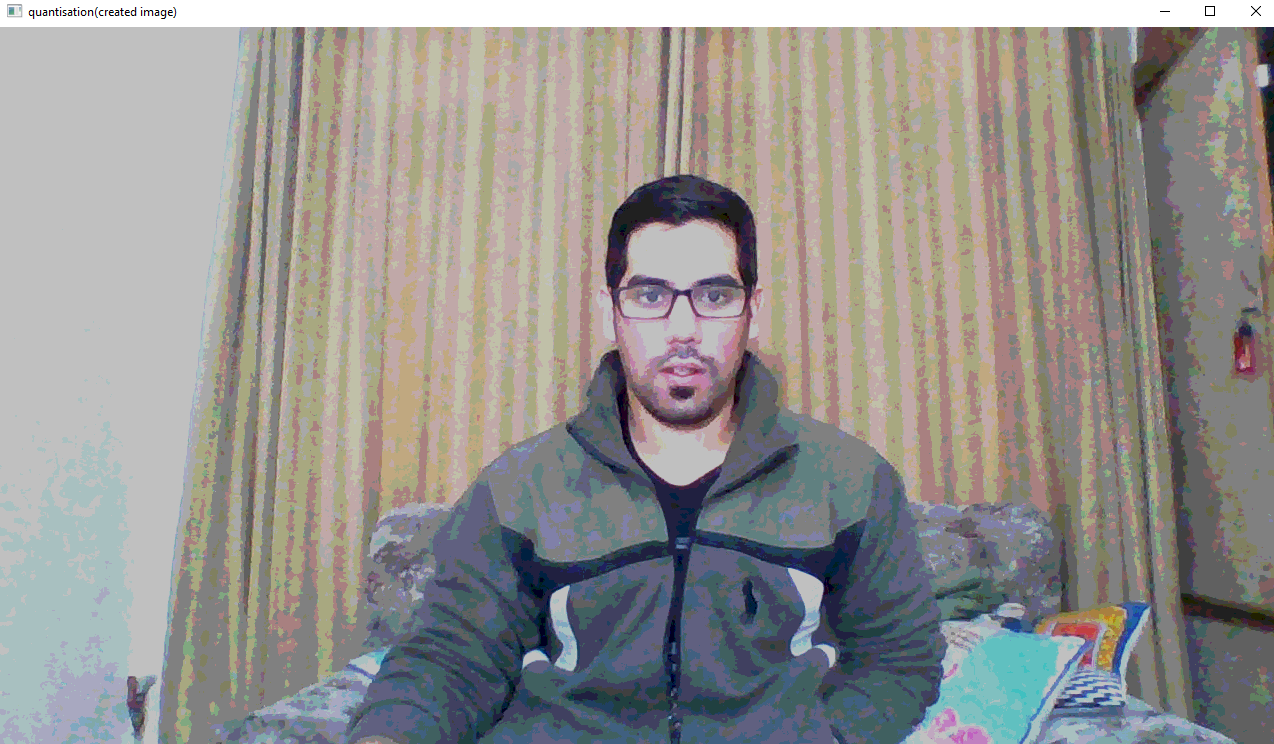
Code:Level4-Quantization

import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
img=cv2.imread("im.jpg",1)  
cv2.imshow("Orignal",img)  
cv2.waitKey(0)  
  
# to get the height and image  
height=img.shape[0]  
width=img.shape[1]  
new\_img=np.zeros((height,width,3),np.uint8)  
for i in range(height):  
 for j in range(width):  
 for k in range(3):  
 if img[i,j][k]<64:  
 gray=0  
 elif img[i,j][k]<128:  
 gray=64  
 elif img[i,j][k]<192:  
 gray=128  
 else:  
 gray=192  
 new\_img[i,j][k]=np.uint8(gray)  
  
cv2.imshow("original",img)  
cv2.imshow("quantisation(created image)",new\_img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()



Code:Level8-Quantization

import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
img=cv2.imread("im.jpg",1)  
cv2.imshow("Orignal",img)  
cv2.waitKey(0)  
  
# to get the height and image  
height=img.shape[0]  
width=img.shape[1]  
new\_img=np.zeros((height,width,3),np.uint8)  
for i in range(height):  
 for j in range(width):  
 for k in range(3):  
 if img[i,j][k]<32:  
 gray=0  
 elif img[i,j][k]<64:  
 gray=32  
 elif img[i,j][k]<96:  
 gray=64  
 elif img[i,j][k]<128:  
 gray=96  
 elif img[i,j][k]<160:  
 gray=128  
 elif img[i,j][k]<192:  
 gray=168  
 elif img[i,j][k]<224:  
 gray=192  
 else:  
 gray=192  
 new\_img[i,j][k]=np.uint8(gray)  
  
cv2.imshow("original",img)  
cv2.imshow("quantisation(created image)",new\_img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()



**Bit plane slicing:Task#3**

Bit plane slicing is the conversion of image into multilevel binary image. These binary images are then compressed using different algorithm. With this technique, the valid bits from gray scale images can be separated, and it will be useful for processing these data in very less time complexity.

The three main goals of bit plane slicing is:

Converting a gray level image to a binary image.

Representing an image with fewer bits and corresponding the image to a smaller size

Enhancing the image by focusing.

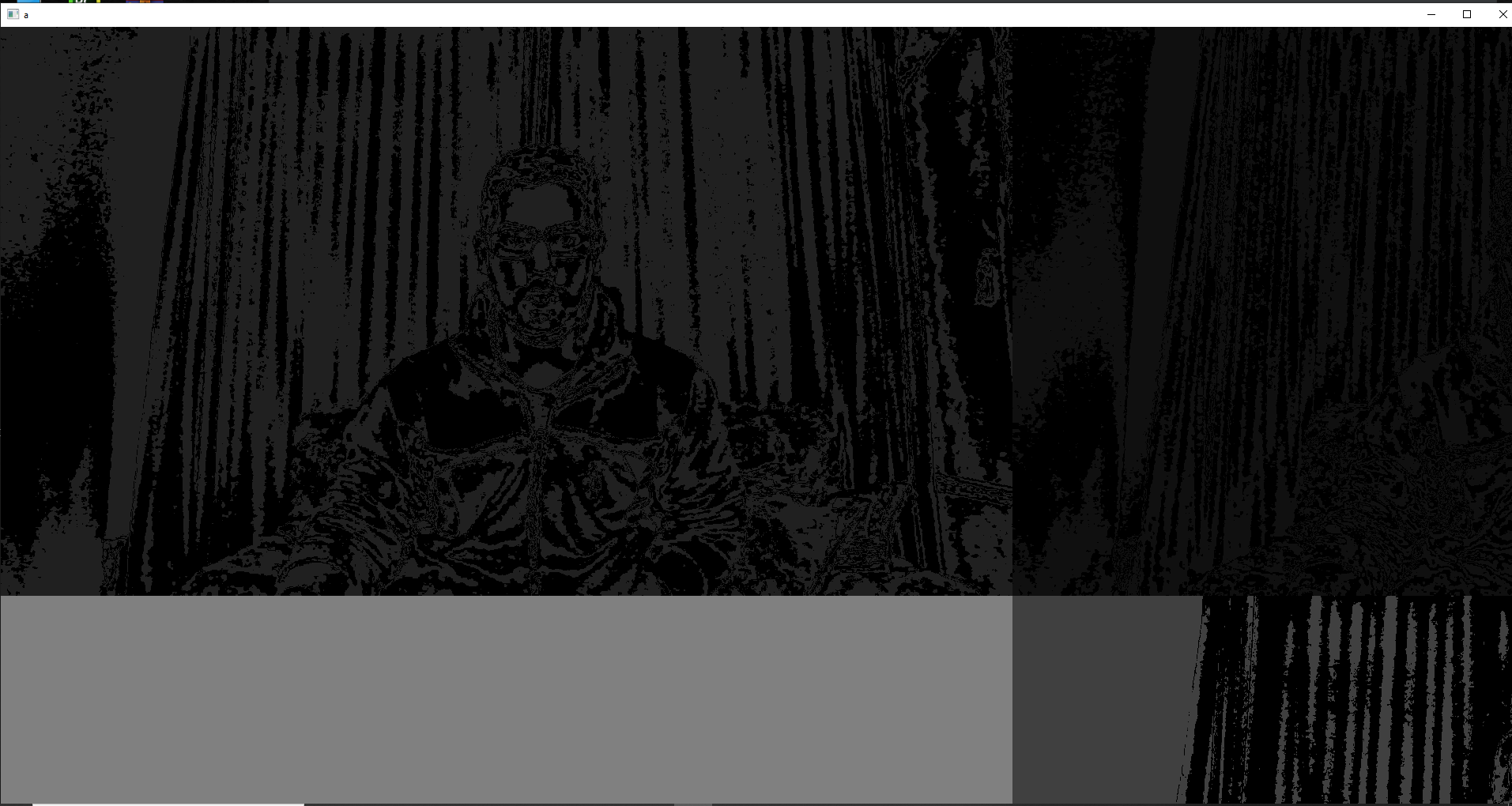
8-bit image slicing:

import numpy as np  
import cv2  
img=cv2.imread("im.jpg",0)  
list1=[]  
#iterate over each pixel and change pixel value to binary using np.binary\_repr() and store it in a list  
for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 list1.append(np.binary\_repr(img[i][j],width=8))#width=no of bits  
  
eight\_bit\_image=(np.array([int(i[0])for i in list1],dtype=np.uint8)\*128).reshape(img.shape[0],img.shape[1])  
seven\_bit\_image=(np.array([int(i[1])for i in list1],dtype=np.uint8)\*64).reshape(img.shape[0],img.shape[1])  
six\_bit\_image=(np.array([int(i[2])for i in list1],dtype=np.uint8)\*32).reshape(img.shape[0],img.shape[1])  
five\_bit\_image=(np.array([int(i[3])for i in list1],dtype=np.uint8)\*16).reshape(img.shape[0],img.shape[1])  
four\_bit\_image=(np.array([int(i[4])for i in list1],dtype=np.uint8)\*8).reshape(img.shape[0],img.shape[1])  
three\_bit\_image=(np.array([int(i[5])for i in list1],dtype=np.uint8)\*4).reshape(img.shape[0],img.shape[1])  
two\_bit\_image=(np.array([int(i[6])for i in list1],dtype=np.uint8)\*2).reshape(img.shape[0],img.shape[1])  
one\_bit\_image=(np.array([int(i[7])for i in list1],dtype=np.uint8)\*1).reshape(img.shape[0],img.shape[1])  
###now we will display  
#we will concatenate so that we don't have to display each image individually  
finalr=cv2.hconcat([eight\_bit\_image,seven\_bit\_image,six\_bit\_image,five\_bit\_image])  
finalv=cv2.hconcat([four\_bit\_image,three\_bit\_image,two\_bit\_image,one\_bit\_image])  
#vertically concatenate  
final=cv2.vconcat([finalr,finalv])  
#display  
cv2.imshow('a',final)  
cv2.waitKey(0)  
###agar vconcat na chaly to display each image separately



**4-bit image slicing:**

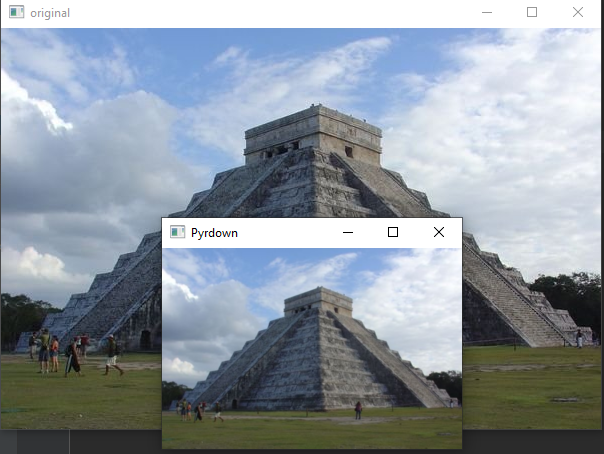
import numpy as np  
import cv2  
img=cv2.imread("im.jpg",0)  
list1=[]  
#iterate over each pixel and change pixel value to binary using np.binary\_repr() and store it in a list  
for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 list1.append(np.binary\_repr(img[i][j],width=4))#width=no of bits  
  
  
four\_bit\_image=(np.array([int(i[0])for i in list1],dtype=np.uint8)\*128).reshape(img.shape[0],img.shape[1])  
three\_bit\_image=(np.array([int(i[1])for i in list1],dtype=np.uint8)\*64).reshape(img.shape[0],img.shape[1])  
two\_bit\_image=(np.array([int(i[2])for i in list1],dtype=np.uint8)\*32).reshape(img.shape[0],img.shape[1])  
one\_bit\_image=(np.array([int(i[3])for i in list1],dtype=np.uint8)\*16).reshape(img.shape[0],img.shape[1])  
###now we will display  
#we will concatenate so that we don't have to display each image individually  
finalr=cv2.hconcat([four\_bit\_image,three\_bit\_image])  
finalv=cv2.hconcat([two\_bit\_image,one\_bit\_image])  
#vertically concatenate  
final=cv2.vconcat([finalv,finalr])  
#display  
cv2.imshow('a',final)  
cv2.waitKey(0)  
###agar vconcat na chaly to display each image separately



**Pyramid (image processing):TASK#4**

Pyramid, or pyramid representation, is a type of multi-scale signal representation developed by the computer vision, image processing and signal processing communities, in which a signal or an image is subject to repeated smoothing and subsampling. Pyramid representation is a predecessor to scale-space representation and multiresolution analysis.

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
img=cv2.imread('img.jpg')  
r=cv2.pyrDown(img)  
cv2.imshow('original',img)  
cv2.imshow('Pyrdown',r)  
cv2.waitKey(0)  
cv2.destroyAllWindows()



* **Conclusion:**

In this lab we learned about Digital Image Processing and some of the functions that we can perform on an image file such as sampling, quantization, bit-plane slicing, pyramid. This was a very interesting lab and we learned a lot in this lab.