Image and Video Processing Project

Group 17 Set 5

Submitted to

Suvendu Rup Sir
Sarthak Padhi Sir

Submitted by

Ritvik Nimmagadda (B118048)
Rohit Kumar Behera (B118049)
Saanidhya Vats (B118050)

PROJECT DESCRIPTION:

This image processing project is based on digitizing an input image based on its spatial and amplitude properties. For digitizing the spatial properties the concept of sampling is used and for digitizing the amplitude properties the concept of quantization is used.

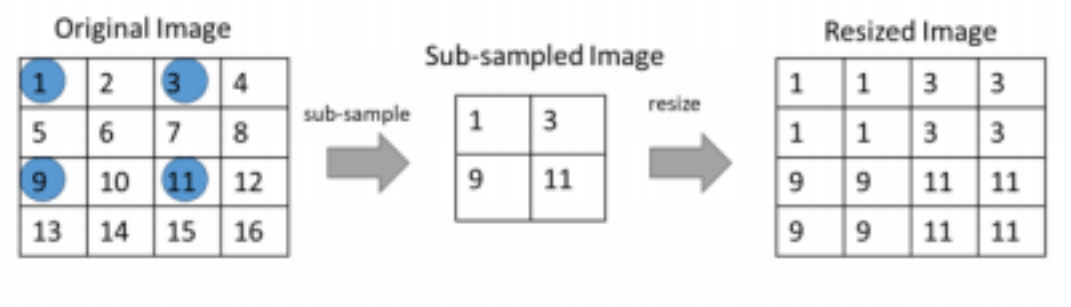
Both the codes are written in python 3.7.
Python Modules used:

- ➔ **Opencv:**
  to read and show the image
- ➔ **numpy:**
  to perform various mathematical operations on the Input image matrix
- ➔ **matplotlib:**
  to render the final results

The programs are performed on standard lena and peppers image and results are shown.

PROBLEM_1: Sampling

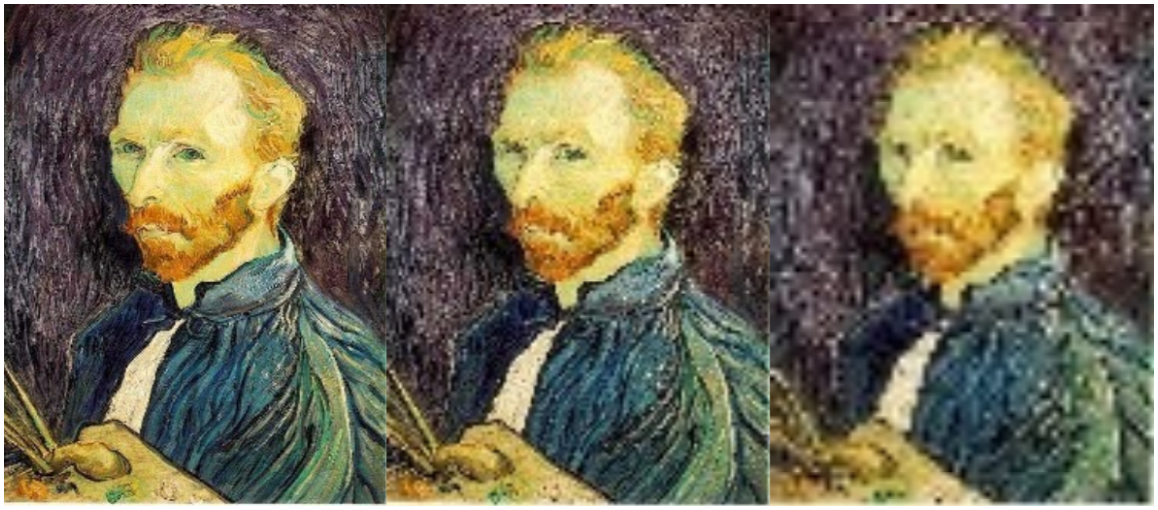
Write a program to change the spatial resolution from 256 x 256 to 128 x 128, 64 x 64, and 32 x 32 pixels using sub-sampling by a factor of 2, 4, and 8 correspondingly. For comparison purposes, resize the sub-sampled images back to the original size 256 x 256 (as shown in the lecture). The following example shows how to sub-sample and resize an image assuming a factor of 2. Use the same idea for factors 4 and 8.



Subsampling is a method that reduces data size by selecting a subset of the original data. The subset is specified by choosing a parameter n (factor), specifying that every nth data point is to be extracted.

This basically means to reduce the spatial resolutions so that we can store the image with lower file size.

Example of downsampling a coloured image:



Algorithm:-

The downsampling is done using the frequency factor. If the frequency factor is f then the input image matrix is divided into as many fxf matrices. Then the first element of each submatrix is selected for the resultant downsampled image.

Let f(x,y,z) be the input image matrix and F(x,y,z) be the downsampled image matrix.

Where x----is the width pixel
y----is the height pixel
Z----is the no of bands(for rgb image it is 3)

Let k be the frequency factor

Pseudocode:

```
down_sampling(f(x,y,z),k):
    a=[x/k]
    b=[y/k]
    For i in 0 to a-1:
        For j in 0 to b-1:
```

```
            For k in 0 to z-1:
                F(i,j,k)=f(x*k,y*k,z)
            end of loop
        end of loop
    end of loop
    return F
```

The upsampling is done from the downsampled image by the means of nearest neighbor replication. We create a fxf submatrix of each element in the downsampled image and subsequently adding these submatrices we get the upsampled image.

Let F(x,y,z) be the downsampled image matrix and f(x,y,z) be the reconstructed upsampled image matrix.

Where x----is the width pixel
y----is the height pixel
Z----is the no of bands(for rgb image it is 3)

Let k be the frequency factor
[ ] is the floor division operator

Pseudocode:

```
up_sampling(F(x,y,z),k):
    a=x*k
    b=y*k
    For i in 0 to a-1:
        For j in 0 to b-1:
            For k in 0 to z-1:
```

f(i,j,k)=F([x/k],[y/k],z)

end of loop

end of loop

end of loop

return f

Python code:-

sampling.py

```
import cv2
import numpy as num
from matplotlib import pyplot as plt

def down_sampling(pic,factor):
    a = pic.shape[0]//factor
    b = pic.shape[1]//factor
    change = num.zeros((a,b,3)).astype('uint8')
    for i in range(0,a):
        for j in range(0,b):
            for k in range(0,3):
                change[i,j,k] = pic[i*factor][j*factor][k]
    return change

def up_sampling(pic,reduced_image):
    factor = pic.shape[0]//reduced_image.shape[0]
    change = num.zeros((pic.shape[0],pic.shape[1],3)).astype('uint8')
    for i in range(0,pic.shape[0]):
        for j in range(0,pic.shape[1]):
            for k in range(0,3):
                change[i,j,k] = reduced_image[i//factor][j//factor][k]
    return change

def show(objects):
    image_rgb = cv2.cvtColor(objects[0],cv2.COLOR_BGR2RGB)
```

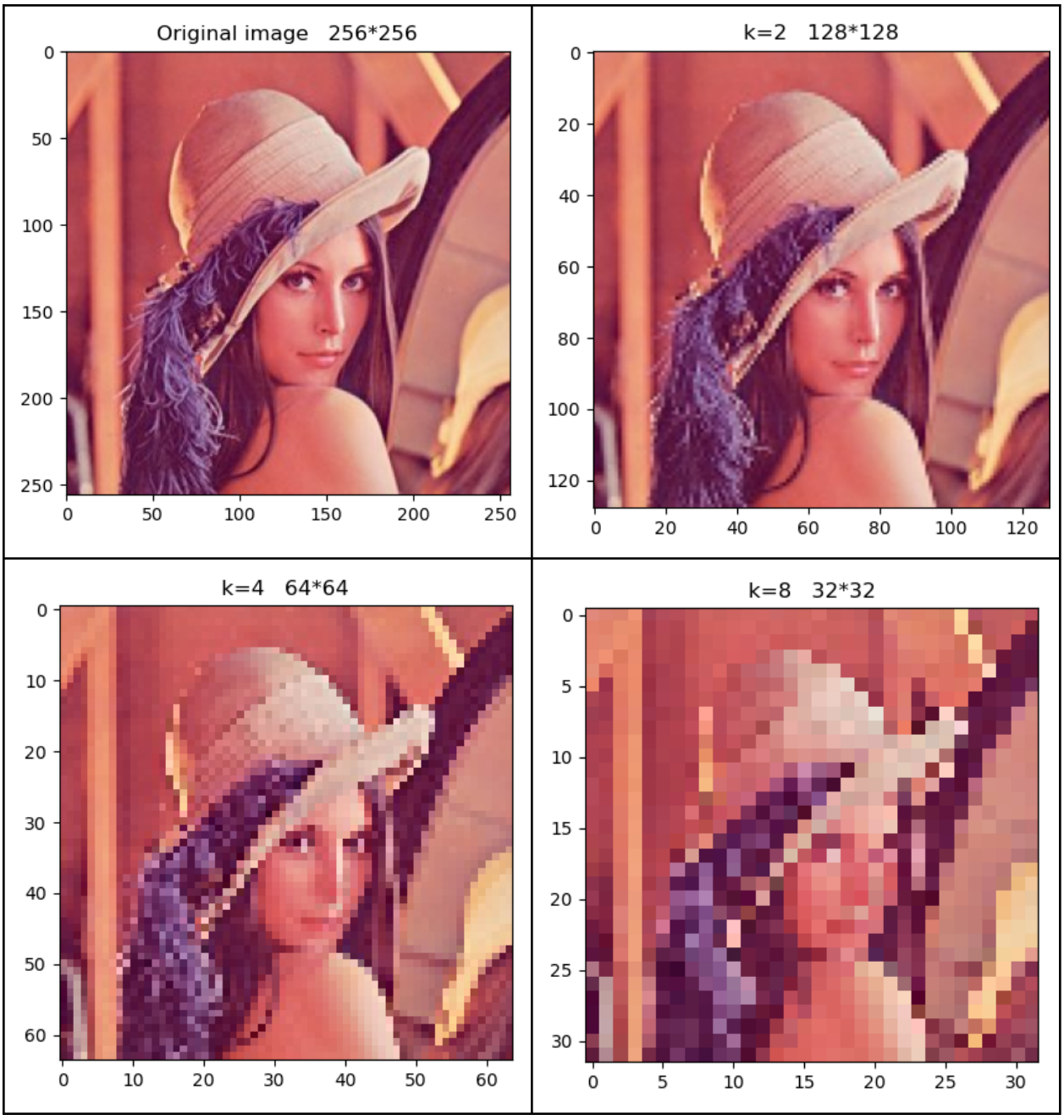
```
    a = objects[0].shape[0]
    b = objects[0].shape[1]
    plt.subplot(2,2,1)
    plt.title("Original image   "+ str(a)+"*"+str(b))
    plt.imshow(image_rgb)
    image_rgb = cv2.cvtColor(objects[1],cv2.COLOR_BGR2RGB)
    a = objects[1].shape[0]
    b = objects[1].shape[1]
    plt.subplot(2,2,2)
    plt.title("k=2   "+str(a)+"*"+str(b))
    plt.imshow(image_rgb)
    image_rgb = cv2.cvtColor(objects[2],cv2.COLOR_BGR2RGB)
    a = objects[2].shape[0]
    b = objects[2].shape[1]
    plt.subplot(2,2,3)
    plt.title("k=4   "+ str(a)+"*"+str(b))
    plt.imshow(image_rgb)
    image_rgb = cv2.cvtColor(objects[3],cv2.COLOR_BGR2RGB)
    a = objects[3].shape[0]
    b = objects[3].shape[1]
    plt.subplot(2,2,4)
    plt.title("k=8   "+ str(a)+"*"+str(b))
    plt.imshow(image_rgb)
    plt.show()

image1 = cv2.imread("lena.jpg")
lena_down_2,lena_down_4,lena_down_8=down_sampling(image1,2),down_sampling(image
1,4),down_sampling(image1,8)
lena_up_2,lena_up_4,lena_up_8=up_sampling(image1,lena_down_2),up_sampling(image
1,lena_down_4),up_sampling(image1,lena_down_8)
items_lena_down = [image1,lena_down_2,lena_down_4,lena_down_8]
items_lena_up = [image1,lena_up_2,lena_up_4,lena_up_8]
show(items_lena_down)
show(items_lena_up)
image2 = cv2.imread("peppers.png")
peppers_down_2,peppers_down_4,peppers_down_8=down_sampling(image2,2),down_sampl
ing(image2,4),down_sampling(image2,8)
peppers_up_2,peppers_up_4,peppers_up_8=up_sampling(image2,peppers_down_2),up_sa
mpling(image2,peppers_down_4),up_sampling(image2,peppers_down_8)
items_peppers_down = [image2,peppers_down_2,peppers_down_4,peppers_down_8]
items_peppers_up = [image2,peppers_up_2,peppers_up_4,peppers_up_8]
show(items_peppers_down)
show(items_peppers_up)
```

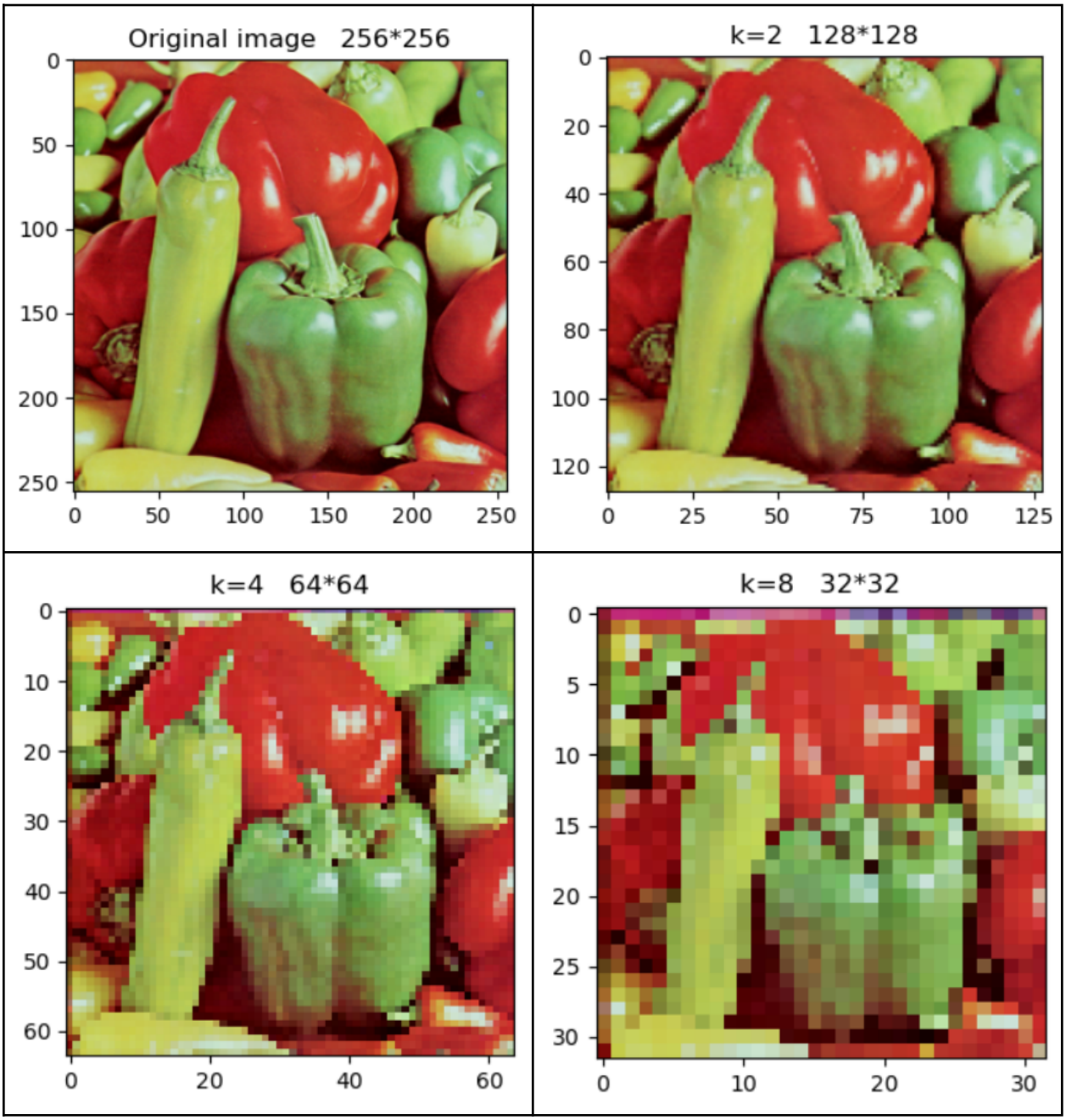
Results:-

Downsampled images:

1) Lena Image

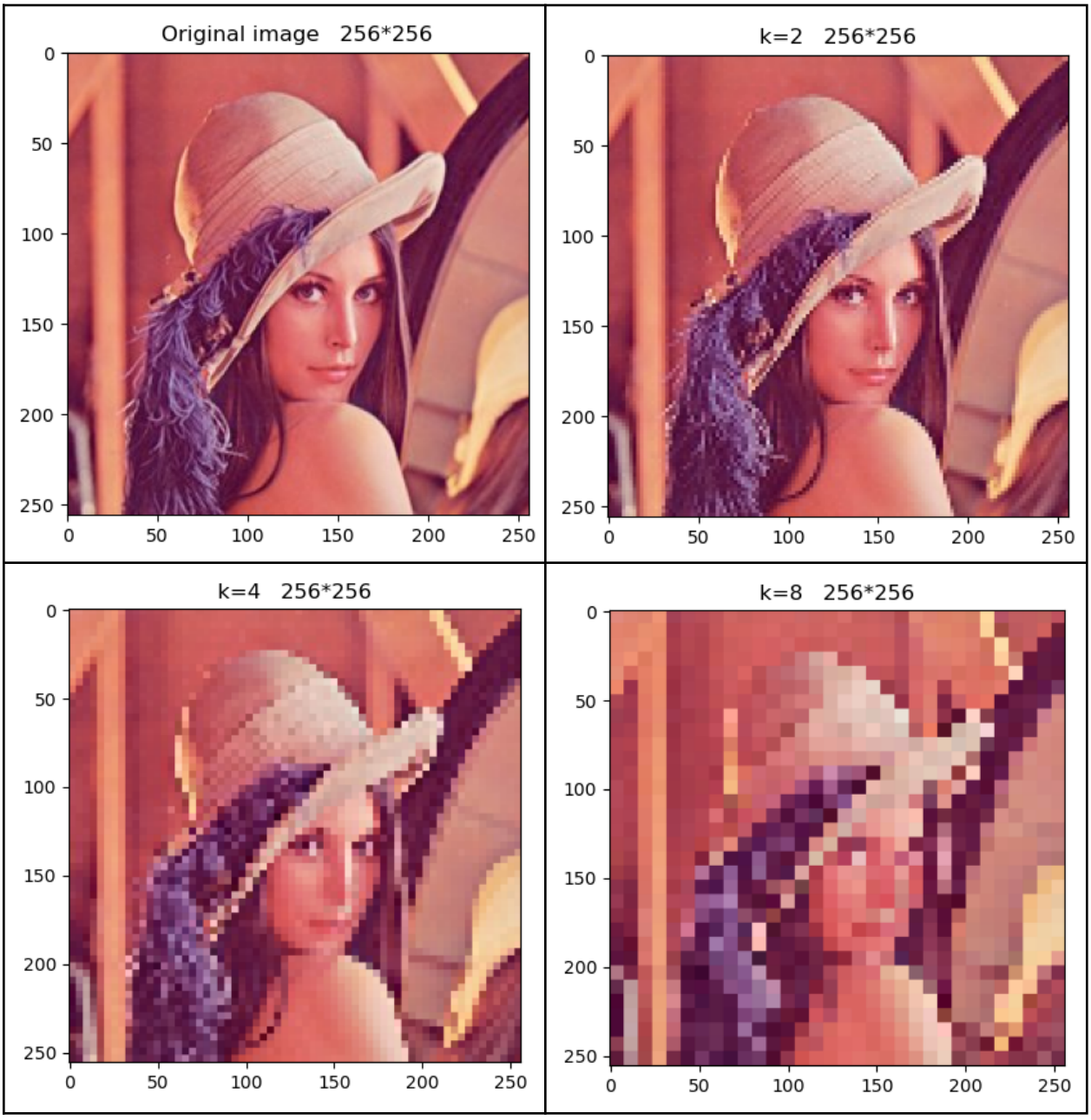


2) Peppers Image

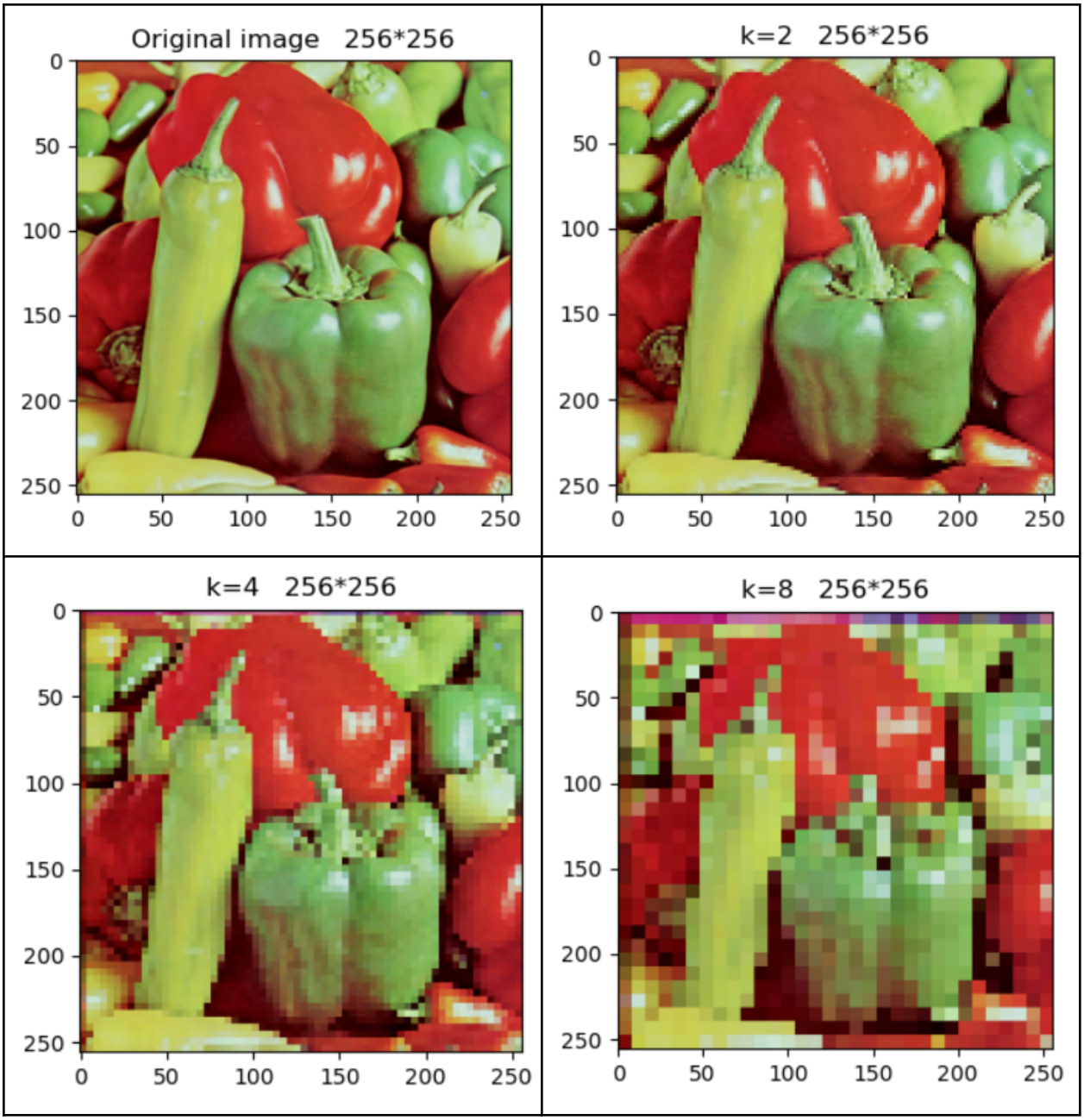


Reconstructed upsampled images:

1) Lena Image



2) Peppers Image



PROBLEM_2: Quantisation

Write a program to reduce the number of gray levels L in a PGM image from L=256 to: (i) L=128, (ii) L=32, (iii) L=8, and (iv) L=2. For visualization purposes, you can still use values in [0, 255] (i.e., Q=255 in the PGM file) although you will only be using fewer gray level values (e.g., when L=2, you can use 0 and 255).

Quantization is the process of mapping input values from a large set to output values in a smaller set, often with a finite number of elements (i.e, digitizing the amplitude or intensity values of pixels). When you are quantizing an image, you are actually dividing a signal into quanta(partitions).

It mainly helps us in reducing the set of pixel values from a larger range into a smaller range of discrete values. This way quantization is helpful towards image compression.

Example of quantizing a grayscale image:

256 levels

32 levels

16 levels

8 levels

4 levels

2 levels

Algorithm:

We have implemented this problem using two different algorithms;

1. Global Thresholding Method
2. K-means Clustering algorithm

to compare the results of these algorithms.

1) The first algorithm that has been used is the global thresholding method. In this method, we have used the floor division operation in order to create the subclasses. The division quotient or the factor of the modulo division is the ratio of total number of pixels to the no of reduced pixels.

Mathematical form for k-level thresholding

Let G(x,y,z) be the quantized image and f(x,y,z) be the input image.

Where x---- is the width pixel

y---- is the height pixel

Let k be the no of levels

[ ] is the floor division operator

$$G(x,y)= \begin{cases} n & , f(x,y) > t_{n-1} \\ n-1 & , t_{n-2} < f(x,y) <= t_{n-1} \\ n-2 & , t_{n-3} < f(x,y) <= t_{n-2} \\ n-3 & , t_{n-4} < f(x,y) <= t_{n-3} \\ - & \\ - & \\ - & \\ 0 & ,f(x,y,z) > t_1 \end{cases}$$

Pseudocode:

```
thresholding(f(x,y),k):
    c=[256/k]
    For i in 0 to x-1:
        For j in 0 to y-1:
            G(i,j)=[F(x,y)/c]*c
        end of loop
    end of loop
    end of loop
    return G
```

2) This algorithm comes into play for unsupervised data (unlabelled data), which means that given some set of points, we would like to find the structure in the data. The algorithm that helps to form clusters of similar data points is known as the clustering algorithm. K-Means Clustering is a technique in which the data points are segregated into K different clusters. Each cluster is uniquely identified by the centroid of the cluster. This algorithm groups similar color values into K clusters and each pixel value (in the final output image)is replaced by the value of the centroid of the cluster to which it belongs.

After randomly initializing the K centroids, the iterative algorithm of K-Means has mainly 2 steps:

1. Cluster Assignment : In this we allot the centroid index to the pixel values such that the euclidean distance[2] between the pixel and the centroid is minimum. The distance between 2 points (a1,b1,c1) and (a2,b2,c2) is given by : $sqrt((a2-a1)^2 + (b2-b1)^2 + (c2-c1)^2)$.
2. Moving Centroids: In this, we calculate the new centroids, by taking the average of all the pixel values that are allocated to a given centroid.

Python code:-

1. quantisation_threshold.py

```
import numpy as num
import cv2
from matplotlib import pyplot as plt
def thresholding(pic,levels):
    a=pic.shape[0]
    b=pic.shape[1]
    factor=256//levels
    change=num.zeros((a,b)).astype('uint8')
    for i in range(0,a):
        for j in range(0,b):
            change[i,j]=(pic[i][j]//factor)*factor
    return change
def show(objects):
    image_rgb=cv2.cvtColor(objects[0],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,1)
    plt.title("Original image \n with 256 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[1],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,2)
    plt.title("128 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[2],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,3)
    plt.title("32 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[3],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,4)
    plt.title("8 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[4],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,5)
    plt.title("2 levels")
    plt.imshow(image_rgb)
    plt.show()

image1=cv2.imread("lena.pgm",cv2.IMREAD_UNCHANGED)
lena_128,lena_32,lena_8,lena_2=thresholding(image1,128),thresholding(image1,32)
,thresholding(image1,8),thresholding(image1,2)
items_lena=[image1,lena_128,lena_32,lena_8,lena_2]
show(items_lena)
image2=cv2.imread("peppers.pgm",cv2.IMREAD_UNCHANGED)
peppers_128,peppers_32,peppers_8,peppers_2=thresholding(image2,128),thresholdin
```

```
g(image2,32),thresholding(image2,8),thresholding(image2,2)
items_peppers=[image2,peppers_128,peppers_32,peppers_8,peppers_2]
show(items_peppers)
```

Results for thresholding quantization:-

1) Lena Image

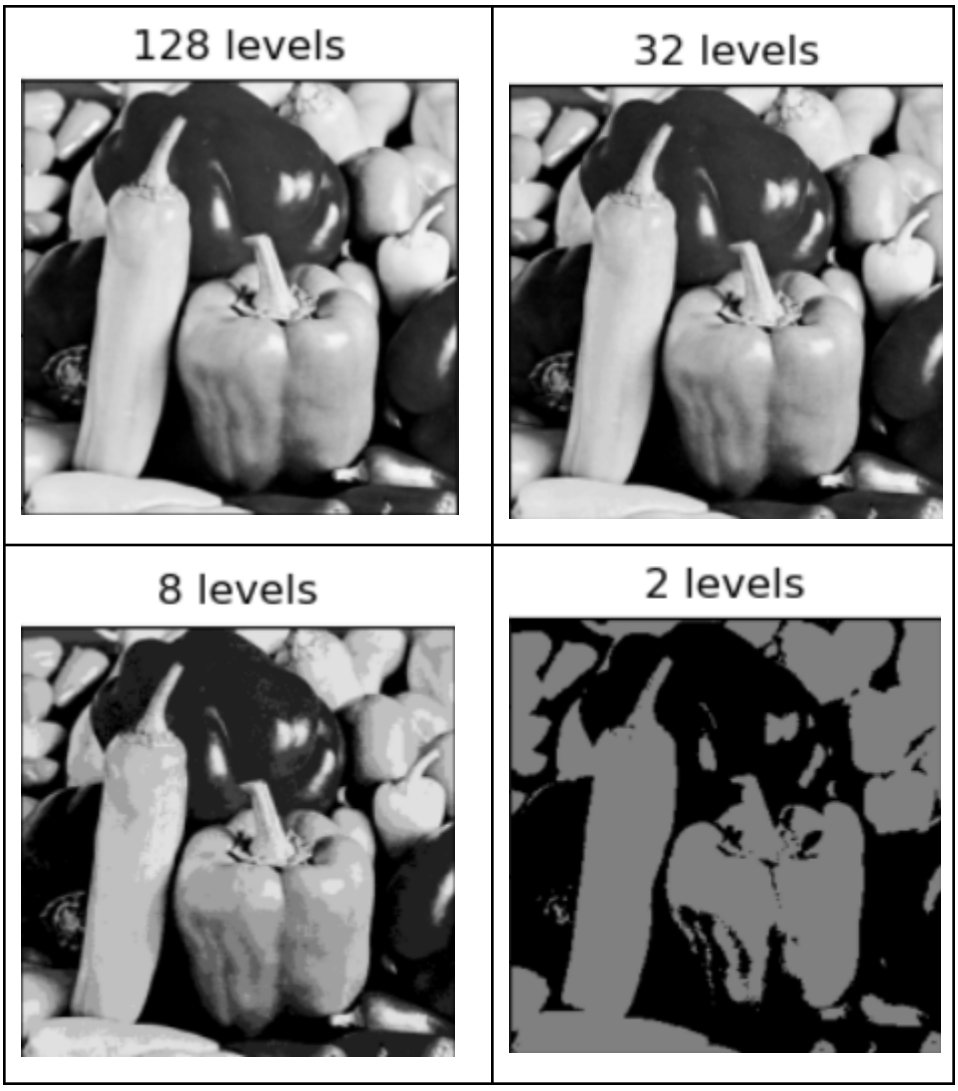
<table>
<tr><td>Original image
with 256 levels
  </td><td></td></tr>
<tr><td>128 levels
  </td><td>32 levels
  </td></tr>
</table>

8 levels

2 levels

2) Peppers Image

Original image
with 256 levels



2.quantisation_k_means.py

```
import numpy as num
import cv2
from matplotlib import pyplot as plt
def k_means(pic,levels):
    q=pic.reshape((-1,3))
    q=num.float32(q)
    act=(cv2.TERM_CRITERIA_EPS+cv2.TERM_CRITERIA_MAX_ITER,10,0.1)
    ret,label,center=cv2.kmeans(q,levels,None,act,10,cv2.KMEANS_RANDOM_CENTERS)
    center=num.uint8(center)
    change=center[label.flatten()]
```

```
    change=change.reshape((pic.shape))
    return change
def show(objects):
    image_rgb=cv2.cvtColor(objects[0],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,1)
    plt.title("Original image \n with 256 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[1],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,2)
    plt.title("128 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[2],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,3)
    plt.title("32 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[3],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,4)
    plt.title("8 levels")
    plt.imshow(image_rgb)
    image_rgb=cv2.cvtColor(objects[4],cv2.COLOR_BGR2RGB)
    plt.subplot(3,2,5)
    plt.title("2 levels")
    plt.imshow(image_rgb)
    plt.show()
image1=cv2.imread("lena.pgm")
lena_128,lena_32,lena_8,lena_2=k_means(image1,128),k_means(image1,32),k_means(i
mage1,8),k_means(image1,2)
items_lena=[image1,lena_128,lena_32,lena_8,lena_2]
show(items_lena)
image2=cv2.imread("peppers.pgm")
peppers_128,peppers_32,peppers_8,peppers_2=k_means(image2,128),k_means(image2,3
2),k_means(image2,8),k_means(image2,2)
items_peppers=[image2,peppers_128,peppers_32,peppers_8,peppers_2]
show(items_peppers)
```

Results for k-nearest quantisation:-

1) Lena Image

Original image
with 256 levels

128 levels

32 levels

8 levels

2 levels

2) Peppers Image

Original image
with 256 levels

128 levels

32 levels

Comparison between k means and thresholding quantisation :

Visually both methods seem to produce the same output but at binary level, the output obtained from thresholding shows better contrast than k-means.

Conclusion:

We have successfully implemented two programs to change the spatial resolution of an image through subsampling and reduced the number of gray levels in a PGM image through global thresholding and k-means.