**IMAGE PROCESSING**

**Assignment 1 (28 points)**

**Goutham Selvakumar (2092286)**

1. **Getting familiar with image manipulation in Matlab (or your favorite language) (*10*/*10*)**

**Write a program/function that will:**

1. **Read and display an image**

import matplotlib.pyplot as plt

import numpy as np

import cv2

import os

import math

import PIL

from PIL import Image

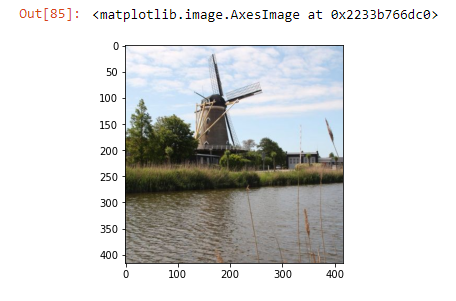
**#Saving the image and displaying it**

img\_location = r'C:\Users\admin\Desktop\Windmill.jpeg'

img\_es = cv2.imread(img\_location)

img\_2 = cv2.cvtColor(img\_es, cv2.COLOR\_BGR2RGB)

plt.imshow(img\_2)



1. **Calculate the size (total number of pixels) of the image**

**#Pixel Counts**

[width, height] = img\_es.shape[:2]

pixel\_counts = width \* height

print(pixel\_counts)



1. **Calculate the maximum pixel value**

**#Maximum number of values**

img\_2\_max = img\_2.max (axis = 0).max(axis = 0)

print('Red Value [Max]: ' + str(img\_2\_max[0]))

print('Blue Value [Max]: ' + str(img\_2\_max[1]))

print('Green Value [Max]: ' + str(img\_2\_max[2]))



1. **Calculate the mean pixel value**

**#Finding the mean pixel value**

img\_2\_mean = img\_2.mean(axis = 0).mean(axis = 0)

print('Red Value [Mean]: ' + str(img\_2\_mean[0]))

print('Blue Value [Mean]: ' + str(img\_2\_mean[1]))

print('Green Value [Mean]: ' + str(img\_2\_mean[2]))



1. **Change the pixel values of the image in the following way: all pixels’ values less than the average calculated at (d) will be equal to 0 and all the others will be equal to 1. What type of image is the new generated image?**

**#Duplication of Original Image**

img\_2\_dup = np.copy(img\_2)

**#Include standards for changing pixel**

black = np.where((img\_2\_dup[:,:,0] < img\_2\_mean[0]) & (img\_2\_dup[:,:,1] < img\_2\_mean[1]) & (img\_2\_dup[:,:,2] < img\_2\_mean[2]))

white = np.where((img\_2\_dup[:,:,0] > img\_2\_mean[0]) & (img\_2\_dup[:,:,1] > img\_2\_mean[1]) & (img\_2\_dup[:,:,2] > img\_2\_mean[2]))

**#Altering the pixels**

img\_2\_dup[black] = (0, 0, 0) ; img\_2\_dup[white] = (255, 255, 255)

**#Plotting the images**

pic = plt.figure()

points1 = pic.add\_subplot(1,2,1)

points1.imshow(img\_2)

plt.xticks([]), plt.yticks([])

points2 = pic.add\_subplot(1,2,2)

points2.imshow(img\_2\_dup)

plt.xticks([]), plt.yticks([])

plt.show()



* The image generated converted pixels that were less than the average intensity to black, while keeping the intensity of the brighter pixels. This can be seen mostly in the right side buildings of windmill, which tells us that his particular shade of red/orange is below the average pixel intensity in the image.

1. **Image Interpolation (*5/7*)**

**Write a computer program that will, given an input image, reduce its spatial resolution, and then return it to its original resolution. Use all of nearest neighbor, bilinear and bicubic interpolation to do this. 481 Students (*2/0*): Design your program such that the desired change in spatial resolution (e.g. 0.5, which will halve the image in each dimension, or 2.0, which will double the image in each dimension) is a variable input to your program. Show an example run of your code.**

* **Nearest Neighbor Example:**

**#Original Size**

fx, fy = fx, fy = img\_es.shape[: 2]

**#Closest neightbor**

seventy\_Percent = cv2.resize(img\_2,(round(fx \* 0.75), round(fy \* 0.75)), interpolation = cv2.INTER\_NEAREST); fifty\_Percent = cv2.resize(img\_2,( round(fx \* 0.5), round(fy \* 0.5)), interpolation = cv2.INTER\_NEAREST); twentyfive\_Percent = cv2.resize(img\_2,(round(fx \* 0.25), round(fy \* 0.25)), interpolation = cv2.INTER\_NEAREST)

**#seventyfive\_Percent resolution size**

plt.imshow(seventy\_Percent)

plt.title('Resolution provided in the seventyfive\_Percent size')

plt.show()

**#fifty\_Percent resolution size**

plt.clf()

plt.imshow(fifty\_Percent)

plt.title('Resolution provided in the fifty\_Percent')

plt.show()

**#twentyfive\_Percent resolution size**

plt.clf()

plt.imshow(twentyfive\_Percent)

plt.title('Resolution provided in the twentyfive\_Percent')

plt.show()

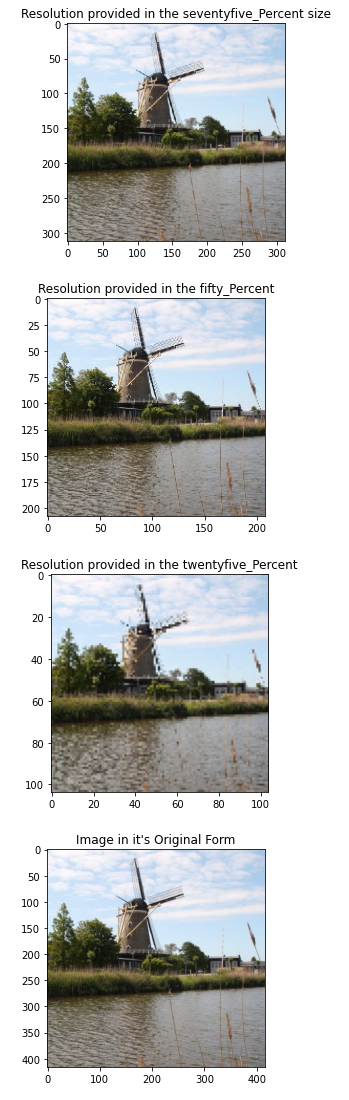
**#Image represented in it's original form**

plt.clf()

plt.imshow(img\_2)

plt.title("Image in it's Original Form")

plt.show()



* **Bilinear Example:**

**#Original size**

fx, fy = fx, fy = img\_es.shape[: 2]

**#Closest neightbor**

seventyfive\_Percent = cv2.resize(img\_2, (round(fx \* 0.75), round(fy \* 0.75)), interpolation = cv2.INTER\_LINEAR); fifty\_Percent = cv2.resize(img\_2, (round(fx \* 0.5), round(fy \* 0.5)), interpolation = cv2.INTER\_LINEAR); twentyfive\_Percent = cv2.resize(img\_2, (round(fx \* 0.25), round(fy \* 0.25)), interpolation = cv2.INTER\_LINEAR)

**#seventyfive\_Percent resolution size**

plt.imshow(seventyfive\_Percent)

plt.title('Resolution provided in the seventyfive\_Percent size')

plt.show()

**#fifty\_Percent resolution size**

plt.clf()

plt.imshow(fifty\_Percent)

plt.title('Resolution provided in the fifty\_Percent')

plt.show()

**#twentyfive\_Percent resolution size**

plt.clf()

plt.imshow(twentyfive\_Percent)

plt.title('Resolution provided in the twenty\_Percent')

plt.show()

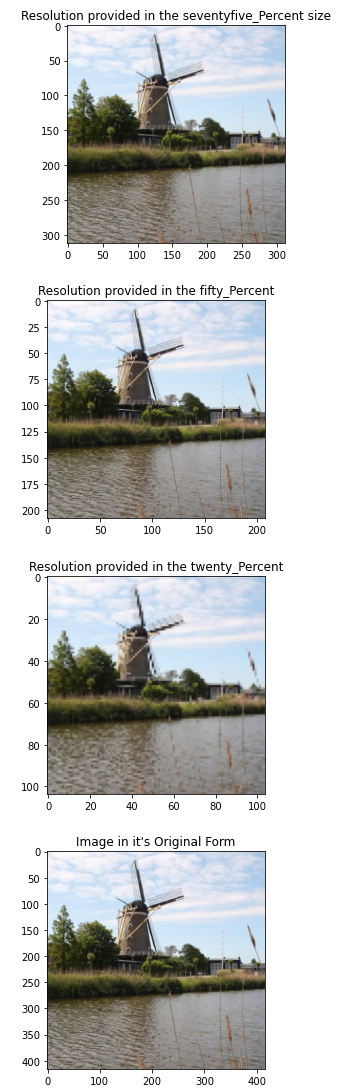
**#Image represented in it's original form**

plt.clf()

plt.imshow(img\_2)

plt.title("Image in it's Original Form")

plt.show()



* **Bicubic Example**

**#Original size**

fx, fy = fx, fy = img\_es.shape[: 2]

**#Closest neightbor**

seventyfive\_Percent = cv2.resize(img\_2, (round(fx \* 0.75), round(fy \* 0.75)), interpolation = cv2.INTER\_CUBIC); fifty\_Percent = cv2.resize(img\_2, (round(fx \* 0.5), round(fy \* 0.5)), interpolation = cv2.INTER\_CUBIC); twentyfive\_Percent = cv2.resize(img\_2, (round(fx \* 0.25), round(fy \* 0.25)), interpolation = cv2.INTER\_CUBIC)

**#seventyfive\_Percent resolution size**

plt.imshow(seventyfive\_Percent)

plt.title('Resolution provided in the seventyfive\_Percent size')

plt.show()

**#fifty\_Percent resolution size**

plt.clf()

plt.imshow(fifty\_Percent)

plt.title('Resolution provided in the fifty\_Percent')

plt.show()

**#twentyfive\_Percent resolution size**

plt.clf()

plt.imshow(twentyfive\_Percent)

plt.title('Resolution provided in the twentyfive\_Percent')

plt.show()

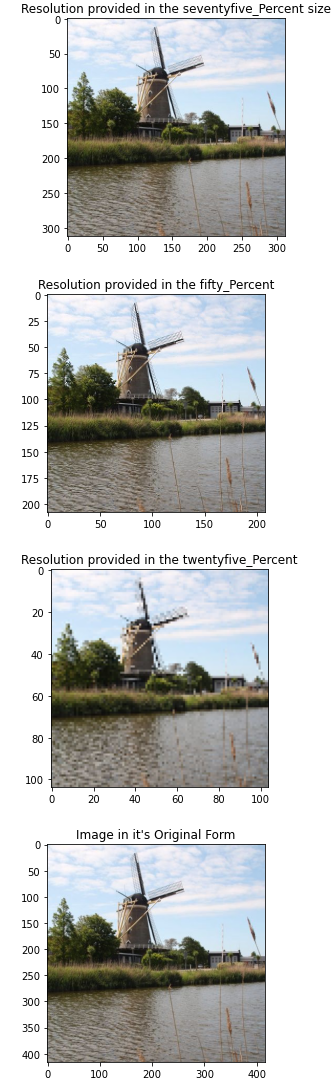
**#Image represented in it's original form**

plt.clf()

plt.imshow(img\_2)

plt.title("Image in it's Original Form")

plt.show()



def resize\_img(image, scaling, interpolation):

#Check interpolation value

inter\_lst = ['nearest neighbor', 'bilinear', 'bicubic']

if interpolation not in inter\_lst:

print('Ensure the interpolation is set to "nn" for the nearest neighbor, "bl" for the bilinear, or "bc" for the bicubic')

return None

**#Loading and displaying the image**

img = cv2.imread(image)

**#Converting the image to rgb**

img2 = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

**#Image size**

width, height = img.shape[: 2]

**#New scale**

new\_scaling = [(width \* float(scaling)), (height \* float(scaling))]

print('Original Scaling: ' + str([width, height]))

print('New Scaling: ' + str([new\_scaling[0], new\_scaling[1]]))

if interpolation == 'nearest neighbor':

**#With Sclaed image**

img\_scaled = cv2.resize(img2, (int(new\_scaling[0]), int(new\_scaling[1])), interpolation = cv2.INTER\_NEAREST)

**#Using the Plots**

plt.clf()

fig = plt.figure()

points1 = fig.add\_subplot(1,2,1)

points1.imshow(img\_scaled)

points1.set\_title('Scaled Nearest Neighbor')

points2 = fig.add\_subplot(1,2,2)

points2.imshow(img2)

points2.set\_title('Original')

plt.show()

if interpolation == 'bilinear':

**#With Scaled image**

img\_scaled = cv2.resize(img2, (int(new\_scaling[0]), int(new\_scaling[1])))

**#Using the Plots**

plt.clf()

fig = plt.figure()

points1 = fig.add\_subplot(1,2,1)

points1.imshow(img\_scaled)

points1.set\_title('Scaled Bilinear')

points2 = fig.add\_subplot(1,2,2)

points2.imshow(img2)

points2.set\_title('Original')

plt.show()

if interpolation == 'bicubic':

**#With Scaled image**

img\_scaled = cv2.resize(img2, (int(new\_scaling[0]), int(new\_scaling[1])), interpolation = cv2.INTER\_CUBIC)

**#Using the Plots**

plt.clf()

fig = plt.figure()

points1 = fig.add\_subplot(1,2,1)

points1.imshow(img\_scaled)

points1.set\_title('Scaled Bicubic')

points2 = fig.add\_subplot(1,2,2)

points2.imshow(img2)

points2.set\_title('Original')

plt.show()

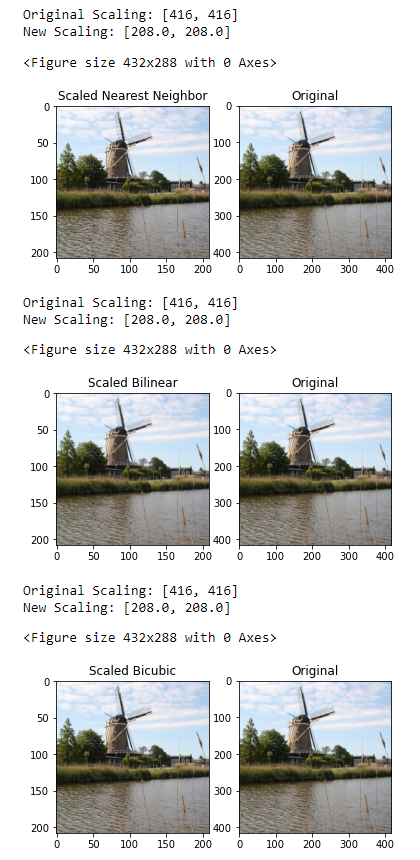
* **Running the scale with 0.5:**

**#Test run with 0.5 scaling**

interpolations = ['nearest neighbor', 'bilinear', 'bicubic']

for interpolation in interpolations:

resize\_img(img\_location, 0.5, interpolation)



* **Running the scale with 2.0:**

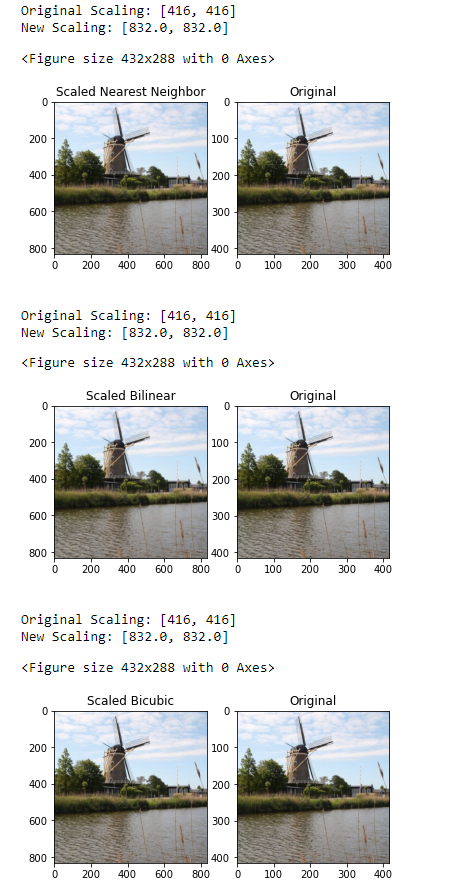
**#Test run with 2.0 scaling**

interpolations = ['nearest neighbor', 'bilinear', 'bicubic']

for interpolation in interpolations:

resize\_img(img\_location, 2, interpolation)

print('')



1. **Reducing the Number of Gray Levels in an Image (*8/11*)**

**Write a computer program capable of reducing the number of gray levels in an image from 256 to 2, in integer powers of 2.  481 Students (*3/0*): Design your program such that the desired number of gray levels does not have to be a power of 2. Show an example run of your code.**

**#Stacking the picture**

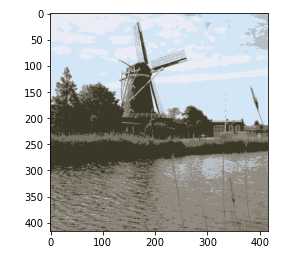
image\_1 = Image.open(img\_location)

image\_1 = image\_1.quantize(6)

#to show determined picture

plt.imshow(image\_1)

plt.show()



**#For the Power of 2**

def quantize\_image\_p2(image, grey\_lvl):

#Gray Level

if int(grey\_lvl) not in range(1, 9):

print("Kindly Enter the grey\_lvl between (1-9)")

return None

img\_es = Image.open(img\_location)

image\_grey = img\_es.quantize(2\*\*grey\_lvl)

plt.imshow(image\_grey)

plt.show()

**#Choosing any integer between 1-256 and applying the Quantization Function**

def quantize\_img(image, grey\_lvl):

**#Gray Level**

if int(grey\_lvl) not in range(1, 257):

print("Kindly Enter the grey\_lvl (1-256)")

return None

img\_es = Image.open(img\_location)

image\_grey = img\_es.quantize(grey\_lvl)

plt.imshow(image\_grey)

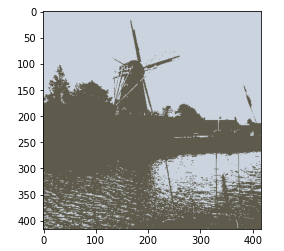
plt.show()

* **Running the image with the powers of 2:**

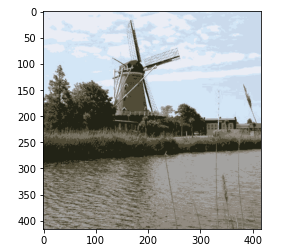
quantize\_image\_p2(img\_location, 9) #Please Enter a grey\_lvl (1-9)



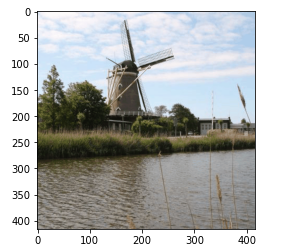
quantize\_image\_p2(img\_location, 1)



quantize\_image\_p2(img\_location, 3)



quantize\_image\_p2(img\_location, 8)

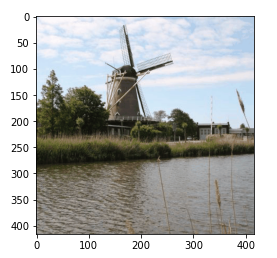


* **Running the image with any integer b/w 1- 256:**

quantize\_img(img\_location, 300)

‘

quantize\_img(img\_location, 200)



quantize\_img(img\_location, 10)

