**Hithesh Shanmugam**

**CSC 481**

**Assignment 1**

Image I have used:



**Problem 1**

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

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

import matplotlib.pyplot as plt

import numpy as np

import math

import cv2

import os

import PIL

from PIL import Image

1. **Read and display an image**

#Reading the image

path\_of\_img = r'C:\Users\sures\OneDrive - DePaul University\Desktop\zoro.jpg' # Enter the path of the image

img = cv2.imread(path\_of\_img)

#Convert to rgb

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

plt.imshow(img\_process) #Displaying the image

**Output:**

<matplotlib.image.AxesImage at 0x23831606220>



# Removing the axes

path\_of\_img = r'C:\Users\sures\OneDrive - DePaul University\Desktop\zoro.jpg'

img = cv2.imread(path\_of\_img)

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

plt.imshow(img\_process)

#Now to hide axes

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

plt.show()

**Output:**

****

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

#Size of the image

breadth, length = img.shape[:2]

size = breadth \* length

print(size)

**Output:**

4247040

1. **Calculate the maximum pixel value**

#Max values of the RGB

maxvalue = img\_process.max(axis = 0).max(axis = 0)

print('Maximum value of red: ' + str(maxvalue[0]))

print('Maximum value of green: ' + str(maxvalue[1]))

print('Maximum value of blue: ' + str(maxvalue[2]))

**Output:**

Maximum value of red: 255

Maximum value of green: 255

Maximum value of blue: 255

1. **Calculate the mean pixel value**

#Mean pixel value

meanvalue = img\_process.mean(axis = 0).mean(axis = 0)

print('Mean value of red: ' + str(meanvalue[0]))

print('Mean value of green: ' + str(meanvalue[1]))

print('Mean value of blue: ' + str(meanvalue[2]))

**Output:**

Mean value of red: 98.00337340830316

Mean value of green: 78.80501572860143

Mean value of blue: 46.57957895381253

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?**

#Copy of original image

imgcpy = np.copy(img\_process)

#Set criteria for changing pixel

black = np.where((imgcpy[:,:,0] < meanvalue[0]) & (imgcpy[:,:,1] < meanvalue[1]) & (imgcpy[:,:,2] < meanvalue[2]))

white = np.where((imgcpy[:,:,0] > meanvalue[0]) & (imgcpy[:,:,1] > meanvalue[1]) & (imgcpy[:,:,2] > meanvalue[2]))

#Change pixels

imgcpy[black] = (0, 0, 0)

imgcpy[white] = (255, 255, 255)

# Plot images

fig = plt.figure()

axis1 = fig.add\_subplot(1,2,1)

axis1.imshow(img\_process)

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

axis2 = fig.add\_subplot(1,2,2)

axis2.imshow(imgcpy)

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

plt.show()

**Explanation:**

By converting the pixel values less than the average intensity values in black and keeping the intensity of the brighter pixels the generated image has more brighter pixels which can be seen in the face of zoro and some of the area like in the left side of the image in his shoulder area.

**Output:**



## Problem 2

## Image Interpolation

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**

#Original size

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

**#Eighty percent resolution**

img\_80 = cv2.resize(img\_process, (round(width \* 0.8), round(height \* 0.8)), interpolation = cv2.INTER\_NEAREST)

plt.clf()

plt.imshow(img\_80)

plt.title('Eighty percent Resolution')

plt.show()

**#Fifty percent resolution**

img\_50 = cv2.resize(img\_process, (round(width \* 0.5), round(height \* 0.5)), interpolation = cv2.INTER\_NEAREST)

plt.clf()

plt.imshow(img\_50)

plt.title('Fifty percent Resolution')

plt.show()

**#Twenty percent resolution**

img\_20 = cv2.resize(img\_process, (round(width \* 0.2), round(height \* 0.2)), interpolation = cv2.INTER\_NEAREST)

plt.clf()

plt.imshow(img\_20)

plt.title('Twenty percent Resolution')

plt.show()

**#Five percent resolution**

img\_5 = cv2.resize(img\_process, (round(width \* 0.05), round(height \* 0.05)), interpolation = cv2.INTER\_NEAREST)

plt.clf()

plt.imshow(img\_5)

plt.title('Five percent Resolution')

plt.show()

**#Original image**

plt.clf()

plt.imshow(img\_process)

plt.title('Original Image')

plt.show()

**Output:**







### **Bilinear**

#Original size

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

**#Eighty percent resolution**

img\_80 = cv2.resize(img\_process, (round(width \* 0.8), round(height \* 0.8)), interpolation = cv2.INTER\_LINEAR)

plt.clf()

plt.imshow(img\_80)

plt.title('Eighty percent Resolution')

plt.show()

**#Fifty percent resolution**

img\_50 = cv2.resize(img\_process, (round(width \* 0.5), round(height \* 0.5)), interpolation = cv2.INTER\_LINEAR)

plt.clf()

plt.imshow(img\_50)

plt.title('Fifty percent Resolution')

plt.show()

**#Twenty percent resolution**

img\_20 = cv2.resize(img\_process, (round(width \* 0.2), round(height \* 0.2)), interpolation = cv2.INTER\_LINEAR)

plt.clf()

plt.imshow(img\_20)

plt.title('Twenty percent Resolution')

plt.show()

**#Five percent resolution**

img\_5 = cv2.resize(img\_process, (round(width \* 0.05), round(height \* 0.05)), interpolation =cv2.INTER\_LINEAR)

plt.clf()

plt.imshow(img\_5)

plt.title('Five percent Resolution')

plt.show()

**#Original image**

plt.clf()

plt.imshow(img\_process)

plt.title('Original Image')

plt.show()

**Output:**

****

### **Bicubic**

#Original size

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

**#Eighty percent resolution**

img\_80 = cv2.resize(img\_process, (round(width \* 0.8), round(height \* 0.8)), interpolation = cv2.INTER\_CUBIC)

plt.clf()

plt.imshow(img\_80)

plt.title('Eighty percent Resolution')

plt.show()

**#Fifty percent resolution**

img\_50 = cv2.resize(img\_process, (round(width \* 0.5), round(height \* 0.5)), interpolation = cv2.INTER\_CUBIC)

plt.clf()

plt.imshow(img\_50)

plt.title('Fifty percent Resolution')

plt.show()

**#Twenty percent resolution**

img\_20 = cv2.resize(img\_process, (round(width \* 0.2), round(height \* 0.2)), interpolation = cv2.INTER\_CUBIC)

plt.clf()

plt.imshow(img\_20)

plt.title('Twenty percent Resolution')

plt.show()

**#Five percent resolution**

img\_5 = cv2.resize(img\_process, (round(width \* 0.05), round(height \* 0.05)), interpolation = cv2.INTER\_CUBIC)

plt.clf()

plt.imshow(img\_5)

plt.title('Five percent Resolution')

plt.show()

**#Original image**

plt.clf()

plt.imshow(img\_process)

plt.title('Original Image')

plt.show()

**Output:**

****

**Spatial Solution:**

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

#Check interpolation value

types = ['nearest', 'bilinear', 'bicubic']

if interpolation not in types:

print('nearest neighbor, bilinear, bicubic')

return None

#Load and display image

img = cv2.imread(picture)

#Convert to rgb

img\_process= 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':

#Scaled image

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

#Plots

plt.clf()

fig = plt.figure()

axis1 = fig.add\_subplot(1,2,1)

axis1.imshow(img\_scaled)

axis1.set\_title('Nearest Neighbor')

axis2 = fig.add\_subplot(1,2,2)

axis2.imshow(img\_process)

axis2.set\_title('Original')

plt.show()

if interpolation == 'bilinear':

#Scaled image

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

#Plots

plt.clf()

fig = plt.figure()

axis1 = fig.add\_subplot(1,2,1)

axis1.imshow(img\_scaled)

axis1.set\_title('Bilinear after scaling')

axis2 = fig.add\_subplot(1,2,2)

axis2.imshow(img\_process)

axis2.set\_title('Original')

plt.show()

if interpolation == 'bicubic':

#Scaled image

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

#Plots

plt.clf()

fig = plt.figure()

axis1 = fig.add\_subplot(1,2,1)

axis1.imshow(img\_scaled)

axis1.set\_title('Bicubic after scaling')

axis2 = fig.add\_subplot(1,2,2)

axis2.imshow(img\_process)

axis2.set\_title('Original')

plt.show()

### **0.5 scaling**

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

for interpolation in interpolations:

resize\_img(path\_of\_img, 0.5, interpolation)

**Output:**

Original Scaling: [2212, 1920]

New Scaling: [1106.0, 960.0]

<Figure size 432x288 with 0 Axes>



Original Scaling: [2212, 1920]

New Scaling: [1106.0, 960.0]

<Figure size 432x288 with 0 Axes>



Original Scaling: [2212, 1920]

New Scaling: [1106.0, 960.0]

<Figure size 432x288 with 0 Axes>



### **2.0 scaling**

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

for interpolation in interpolations:

resize\_img(path\_of\_img, 2, interpolation)

print(' ')

**Output:**

Original Scaling: [2212, 1920]

New Scaling: [4424.0, 3840.0]

<Figure size 432x288 with 0 Axes>



Original Scaling: [2212, 1920]

New Scaling: [4424.0, 3840.0]

<Figure size 432x288 with 0 Axes>



Original Scaling: [2212, 1920]

New Scaling: [4424.0, 3840.0]

<Figure size 432x288 with 0 Axes>

### **Problem 3**

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.

#Load image

image\_original = Image.open(path\_of\_img)

#Set quantinization level

image\_original = image\_original.quantize(3)

# to show the quantized image

plt.imshow(image\_original)

plt.show()

**Output:**

****

### **Quantizing in powers of 2**

#Quantize function for powers of 2

def quantize\_img\_p2(image, grey\_val): #grey\_exp needs to be between 1 and 8

#Check grey\_lvl

if int(grey\_val) not in range(1, 17):

print("Enter a valid grey level value between (1 - 17)")

return None

img = Image.open(path\_of\_img)

grey\_lvl\_img = img.quantize(2\*\*grey\_val)

plt.imshow(grey\_lvl\_img)

plt.show()

quantize\_img\_p2(path\_of\_img, 1)

**Output:**

****

quantize\_img\_p2(path\_of\_img, 4)

**Output:**

****

quantize\_img\_p2(path\_of\_img, 8)

**Output:**

****

quantize\_img\_p2(path\_of\_img, 17)

**Output:**

Enter a valid grey level value between (1 - 17)

### **Quantizing between 1-256**

#Quantize function for

def quantize\_img(image, grey\_val): #grey\_exp needs to be between 1 and 8

#Check grey\_lvl

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

print("Enter a valid grey level value between (1 - 256)")

return None

img = Image.open(path\_of\_img)

grey\_lvl\_img = img.quantize(grey\_val)

plt.imshow(grey\_lvl\_img)

plt.show()

quantize\_img(path\_of\_img, 10)

**Output:**

****

quantize\_img(path\_of\_img, 100)

**Output:**

****

quantize\_img(path\_of\_img, 400)

**Output:**

Enter a valid grey level value between (1 - 256)