Homework 1

ECE 253
Digital Image Processing

October 7, 2020

Problem 1. Basics (3 points)

$$\text{Input } A = \begin{bmatrix} 3 & 9 & 5 & 1 \\ 4 & 25 & 4 & 3 \\ 63 & 13 & 23 & 9 \\ 6 & 32 & 77 & 0 \\ 12 & 8 & 6 & 1 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

(i) Point-wise multiply A with B and set it to C.

(ii) Calculate the inner product of the 2nd and 3rd row of C.

(iii) Find the minimum and maximum values and their corresponding row and column indices in matrix C. If there are multiple min/max values, you must list all their indices.

In your report include all the outputs generated by your code.

Example solution:

(i)

Code input:

```
A = np.matrix(
    ((3, 9, 5, 1),
    (4, 25, 4, 3),
    (63, 13, 23, 9),
    (6, 32, 77, 0),
    (12, 8, 6, 1))
)

B = np.matrix(
    ((0, 1, 0, 1),
    (0, 1, 1, 0),
    (0, 0, 0, 1),
    (1, 1, 0, 1),
    (0, 1, 0, 0))
)

C = np.multiply(A,B)

print("(i)")
print("C =")
print(C)
```

Output:

```
C =
[[ 0  9  0  1]
 [ 0 25  4  0]
 [ 0  0  0  9]
 [ 6 32  0  0]
 [ 0  8  0  0]]
```

(ii)

Code input:

```
C_row2 = C[2,:]
C_row3 = C[3,:]

C_row3_T = np.matrix.transpose(C_row3)

inner_product = C_row2*C_row3_T
```

Output:

```
Inner product =
[[0]]
```

(iii)

Code input:

```
print("(iii)")
print("Maximum value and all corresponding indices:")
max_C_index = np.argwhere(C == np.amax(C))
print(np.max(C))
print(max_C_index)

print("Minimum value and all corresponding indices:")
min_C_index = np.argwhere(C == np.amin(C))
print(np.min(C))
print(min_C_index)
```

Output:

```
Maximum value and all corresponding indices:
32
[[3 1]]
Minimum value and all corresponding indices:
0
[[0 0]
 [0 2]
 [1 0]
 [1 3]
 [2 0]
 [2 1]
 [2 2]
 [3 2]
 [3 3]
 [4 0]
 [4 2]
 [4 3]]
```

Problem 2. Simple image manipulation (5 points)

(i) Download any color image from the Internet or use one of the given images. Read this image and call it **A**.

(ii) Transform the color image to gray-scale. Verify the values are between 0 and 255. If not, please normalize your image from 0 to 255. Call this image **B**.

(iii) Add 15 to each value of image B. Set all pixel values greater than 255 to 255. Call this image **C**.

(iv) Flip image B along both the horizontal and vertical axis. Call this image **D**.

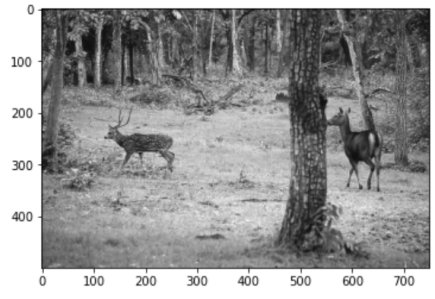
(v) Calculate the median of all values in image B. Next, threshold image B by the median value you just calculated i.e. set all values greater than median to 0 and set all values less than or equal to the median to 1. Name this binary image **E**.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

#Read an image and store it to A
A = plt.imread('forest.jpg')
plt.imshow(A);
```

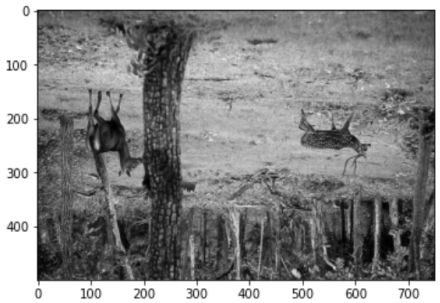

```
#Convert image A to grayscale and store it in B
B = cv2.cvtColor(A, cv2.COLOR_RGB2GRAY)
print("Min:", np.min(B), "Max:", np.max(B))
plt.imshow(B, cmap='gray');
```

```
Min: 0 Max: 255
```

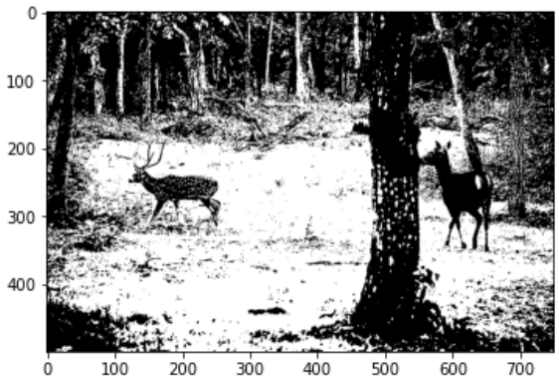

```
#Add 15 to the intensity value of every pixel in B and store it in C
C = B + 15;
plt.imshow(C, cmap='gray');
```




```
#Flip image B along both axes and store it in D
D = np.fliplr(np.flipud(B));
plt.imshow(D, cmap='gray');
```



```
#Threshold image B with its median and store the binary image in E
E = cv2.threshold(B, np.median(B), 1, cv2.THRESH_BINARY)[1]
plt.imshow(E, cmap='gray');
```

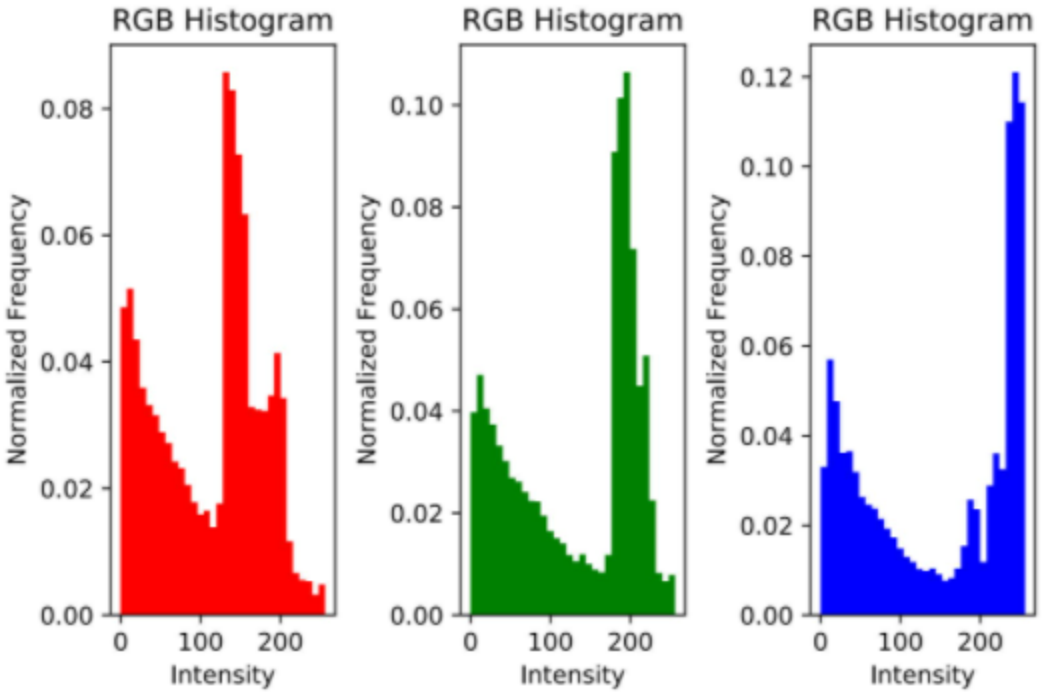


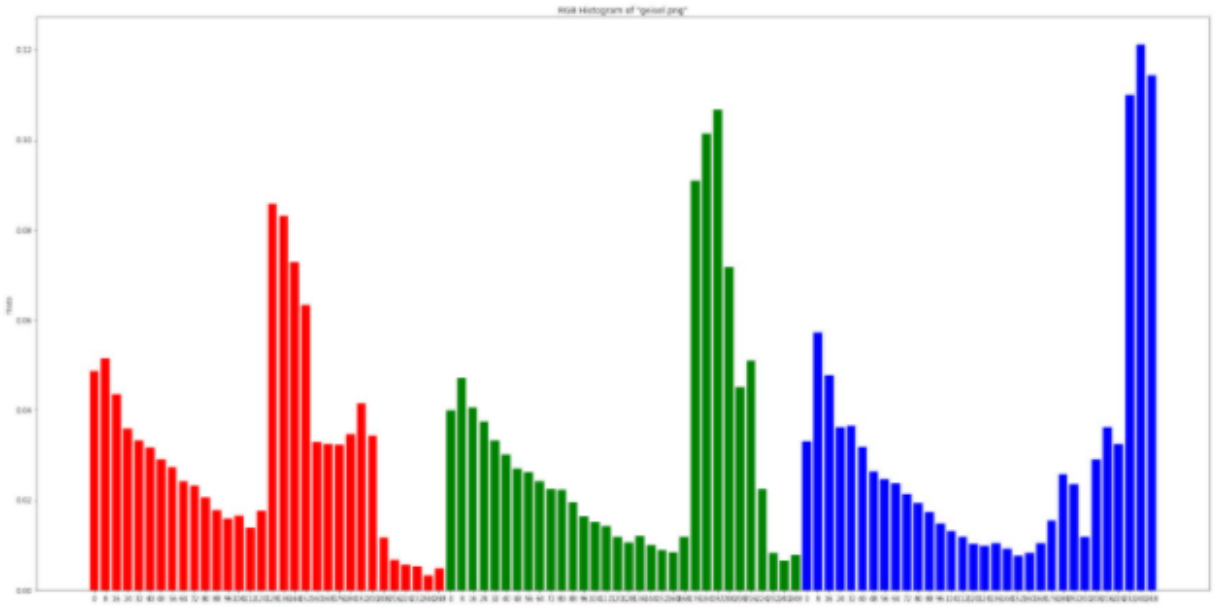
Problem 3. Histograms (6 points)

Histograms[1] are a great statistical tool to analyze the distribution of intensity values in an image. In this problem, you have to write a MATLAB/Python function with the following specifications:

- Write a function named *compute_norm_rgb_histogram* that computes the RGB color histogram.
- Use 32 bins for each color channel (i.e. Red, Green and Blue), spaced equally between 0 and 255. This should result in a 32-length vector for each channel.
- One input (RGB/color image) and one output (1 x 96 vector).
- Normalize the histogram of each color channel (so that it sums to 1), then concatenate the three histograms together (in the order R, G, B) to make a combined histogram of length 3x32 = 96.
- Do not use MATLAB/Python inbuilt histogram function. You may use loops if necessary.
- Call the function and plot the final combined, normalized histogram for the image *geisel.jpg*. Make sure the plot is labeled correctly. Show your plot in the report.

We received a variety of solutions for the histogram question. These two examples below best illustrate the main ideas; ideally, the first solution would have been displayed on a single graph, but the labeling is excellent.


RGB Histogram
RGB Histogram
RGB Histogram
Normalized Frequency
Intensity
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
100
200


RGB Histogram of "geisel.png"

Python Example:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from skimage import color
from skimage import io
from PIL import Image
```

```
def space_value(channel):
    '''
    count the number of pixels distributed in 32 bins between 0 and 255
    and return the counts as 1*32 vector
    '''
    bins = [*range(0,255,8)]
    ans = np.zeros(32)
    for i in range (channel.size):
        value = channel[i]
        index = (int)(value/8)
        ans[index] += 1
```

```
    # normalize the value
    for j in range (32):
        ans[j] /= channel.size
    return ans
```

```
def compute_norm_rgb_histogram(img):
    '''
    compute the RGB color histogram
    return 1*96 vector
    normalize to 1
    '''
    redpixels = np.array(img.getdata(0))
    greenpixels = np.array(img.getdata(1))
    bluepixels = np.array(img.getdata(2))

    red = space_value(redpixels)
    green = space_value(greenpixels)
    blue = space_value(bluepixels)

    ans = np.concatenate((red, green), axis=None)
    ans = np.concatenate((ans, blue), axis=None)
    return ans
```

```
def plot_rgb_histogram(histo):
    '''
    plot the combined and normalized histogram,
    with plots for each channel
    '''
    # cut the 1*96 array to 3 channels
    red = histo[:32]
    green = histo[32:64]
    blue = histo[64:]
```

```
    ind = np.arange(32)
    indices = np.concatenate((ind, ind), axis=None)
    indices = np.concatenate((indices, ind), axis=None)
    width = 0.3
    fig, ax = plt.subplots(figsize=(30,15))
    plt.bar(np.arange(32), red, color='r')
    plt.bar(np.arange(32,64), green, color='g')
    plt.bar(np.arange(64,96), blue, color='b')
```

```
    ax.set_ylabel('Histo')
    ax.set_title('RGB Histogram of "geisel.png"')
    ax.set_xticks(range(96))
    ax.set_xticklabels(indices*8)
    plt.show()
```

```
# read in the image
img = Image.open('geisel.jpg').convert('RGB')
rgb_histo = compute_norm_rgb_histogram(img)
```

```
plot_rgb_histogram(rgb_histo)
```

Problem 4. Chroma Keying (6 points)

Chroma keying is used for extracting the foreground from images, with the background typically being a green screen. In this problem, you have been provided 2 images: *travolta.jpg* and *dog.jpg*. Write a matlab script to extract the foreground from either image and overlay the foreground on a different background of your choice. In your report you should include for each of the images:

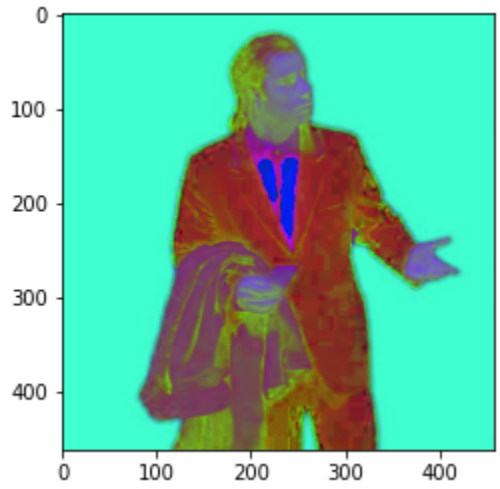
(i) A binary image showing the foreground mask, i.e., all foreground pixels set to 1 and all background pixels set to 0.

(ii) An image with the background pixels set to 0 and the foreground pixels set to their original values.

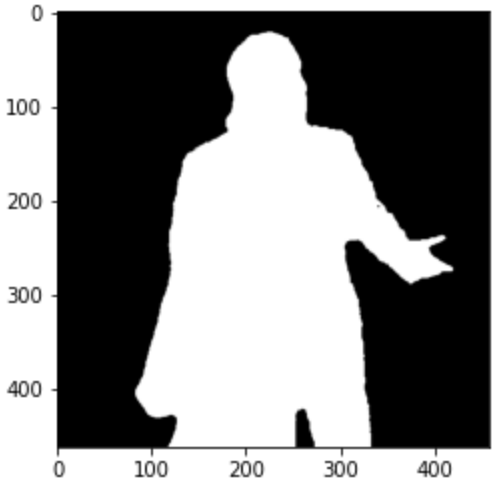
(iii) An image with the foreground overlayed on a background of your choice.

Sample Solution (Python):

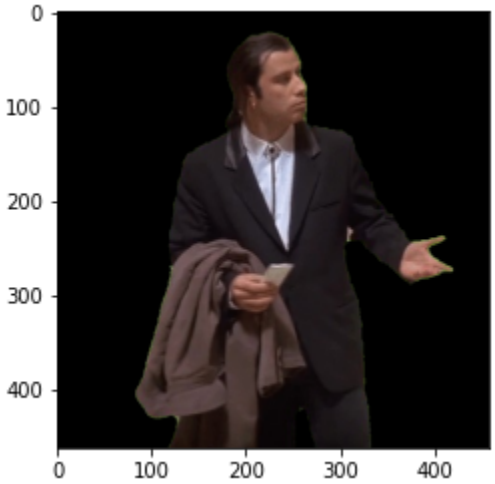
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

travolta = cv2.imread('travolta.jpg')
travolta = cv2.cvtColor(travolta, cv2.COLOR_BGR2RGB)
travolta_hsv = cv2.cvtColor(travolta, cv2.COLOR_BGR2HSV)
plt.figure()
plt.imshow(travolta_hsv,cmap='hsv')
plt.show()
```




```
lower_bound = np.array([45, 90, 0])
upper_bound = np.array([75, 255, 255])
mask = cv2.inRange(travolta_hsv, lower_bound,upper_bound)
mask = cv2.bitwise_not(mask)
plt.figure()
plt.imshow(mask, cmap='gray')
plt.show()
```




```
foreground = cv2.bitwise_and(travolta, travolta, mask = mask)
plt.figure()
plt.imshow(foreground)
plt.show()
```




```
geisel = cv2.imread("geisel.jpg")
geisel = cv2.cvtColor(geisel, cv2.COLOR_BGR2RGB)

geisel[1000:1000+462,1500:1500+458,0] =
geisel[1000:1000+462,1500:1500+458,0]*((255-mask)//255)
geisel[1000:1000+462,1500:1500+458,1] =
geisel[1000:1000+462,1500:1500+458,1]*((255-mask)//255)
geisel[1000:1000+462,1500:1500+458,2] =
geisel[1000:1000+462,1500:1500+458,2]*((255-mask)//255)

geisel[1000:1000+462,1500:1500+458,0] += foreground[:,:,0]
geisel[1000:1000+462,1500:1500+458,1] += foreground[:,:,1]
geisel[1000:1000+462,1500:1500+458,2] += foreground[:,:,2]
```

```
plt.figure(figsize=(16,10))
plt.imshow(geisel)
plt.show()
```

Problem 5. Upsampling and downsampling (10 points)

Sampling is a technique that enables you to resize the image to desired resolution. Different interpolation techniques can be used for sampling. In this question, you will perform experiments on different interpolation methods for upsampling and downsampling (Hint: you can use resize function for both upsampling and downsampling).

(i) List (and describe in a short paragraph) 3 interpolation methods.

(ii) Select 3 color images and downsample using the different methods with the 3 ratios below. What differences do you observe? Which interpolation method do you think works best?

- Dowsampling ratio: 0.3, 0.5, 0.7
- Please include the images in the report. Crop small regions to compare if necessary. You should have 27 images (3 images $*$ 3 methods $*$ 3 ratios).

(iii) Repeat the previous step, but this time use upsampling with the ratios below. What differences do you observe? What interpolation method works best?

- Upsampling ratio: 1.5, 1.7, 2
- Please include the images in the report. Crop small regions to compare if necessary.

(iv) Using 3 color images, downsample the images with scale 0.1 and upsample back to the original size, using all combinations of the three chosen interpolation methods. Which interpolation combination do you think works best to reconstruct the original image?

Results and discussion vary based on interpolation techniques and images selected. Here is an exemplary solution:

Size \ Method	Bicubic	Bilinear	Nearest
0.3			
0.5			
0.7			

Table 1: Airplane, downsampling, Dowsampling ratio: 0.3, 0.5, 0.7, methods, bicubic, bilinear, nearest

- **Nearest neighbour interpolation**: The interpolated value is the nearnest neighbour's value.
- **Linear interpolation**: The interpolated value is based on the linear interpolation of each dimension of two.
- **Cubic interpolation**: The interpolated value at a query point is based on a cubic interpolation of the values at neighboring grid points in each respective dimension. The interpolation is based on a cubic convolution.

Here are some of the code snippets in Python that could be used for resizing images for this problem:

Python

```python
import PIL
from PIL import Image
import os
modes=[
    ["PIL.Image.NEAREST",PIL.Image.NEAREST],
    ["PIL.Image.BILINEAR",PIL.Image.BILINEAR],
    ["PIL.Image.BICUBIC",PIL.Image.BICUBIC],
    ["PIL.Image.LANCZOS",PIL.Image.LANCZOS]
]
images = ['data/Lenna.png','data/geisel.jpg','data/luhan.jpg']
k=0
for imgname in images:
    img=Image.open(imgname)
    w,h=img.size
    for i in range(4):
        mode = modes[i]
        if os.path.exists('data/{}_downsample'.format(mode[0]))==False:
            os.mkdir('data/{}_downsample'.format(mode[0]))
        for j in [0.3,0.5,0.7]:
            img = img.resize((int(w*j),int(h*j)),mode[1])
            img.save("data/{}_downsample/img2_{}_{}.jpg".format(mode[0],j,k))
            k+=1
```