Image Processing

Homework 1

Names: Raneem Ibraheem (212920896), Selan Abu Saleh (212111439)

Question 1:

Section A:

There are a bunch of factors to take in mind, to mention some:

1. **Sensor Size: the bigger the sensor the more pixels it can fit without compromising the image quality.**
2. **Processing power: higher resolution requires more processing power and faster processing for image capturing and rendering.**
3. **Storage capacity: the higher the resolution the larger the file size is, therefore it requires more storage.**
4. **Editing and compression: high resolutions require efficient algorithms for compression and post-processing.**
5. **Application: the need for a specific case dictates the requirements, for example if you are a professional photographer then you might need a better resolution than a person with a normal use for the camera.**

Section B:

The strength of image quantization involves several considerations:

1. **Bit count: older hardware supported fewer bits per pixel, therefore limiting the color representation.**
2. **Memory constraints: lower quantization reduces storage and processing requirements.**
3. **Processing power: limited computational capacity required simpler quantization to reduce rendering time.**
4. **Data transfer rates: lower quantization minimized data transfer overheads for slower connections.**

Question 2:

Section A:

As we saw in the tutorial, the wave length is dependent on the frequency, and it is 1/f, where f is the frequency. Now in our case, we are given the function Sin(πkx). Recall tutorial 1 where we saw the k is the value of the frequency of the wave, therefore, in our case the frequency is k/2, and when we substitute in the equation we get: lambda = 1/ k /2 → lambda = 2/k.

Section B:

According to Nyquist, in order to reconstruct the image, the sampling frequency much be twice the frequency of the signal. We inferred from the last section that the sampling frequency is k/n (since lambda = 1/f = 2/k), and also we can infer from the image that there is an alternation between white and black bars and each bar has the width of A, therefore the sampling frequency is 1/2A.

As we mentioned before, we need the sampling frequency to be greater than twice the signal frequency, meaning that: 1/2A >= k.

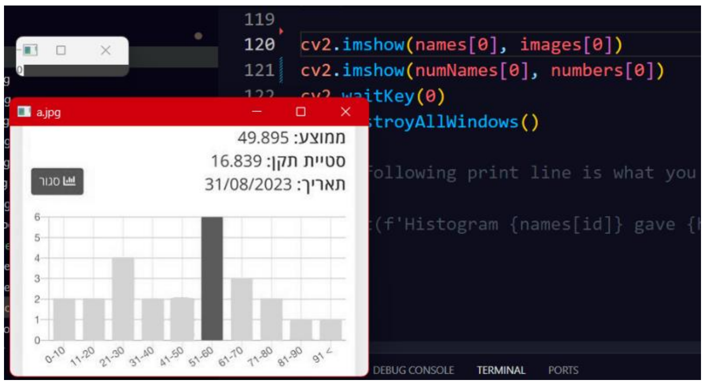
Now for case 1 where A = 0.25, we get: k<=2 and since k can't be less than 0 therefore k should be between 0 and 2 (including).

As for case 2 where A = 2, we get k<=0.25 and similar to before k can't be less than 0 therefore k should be between 0 and 0.25.

Question 3:

Section A + B:

Here is the output that we got, which also includes the code that we added:

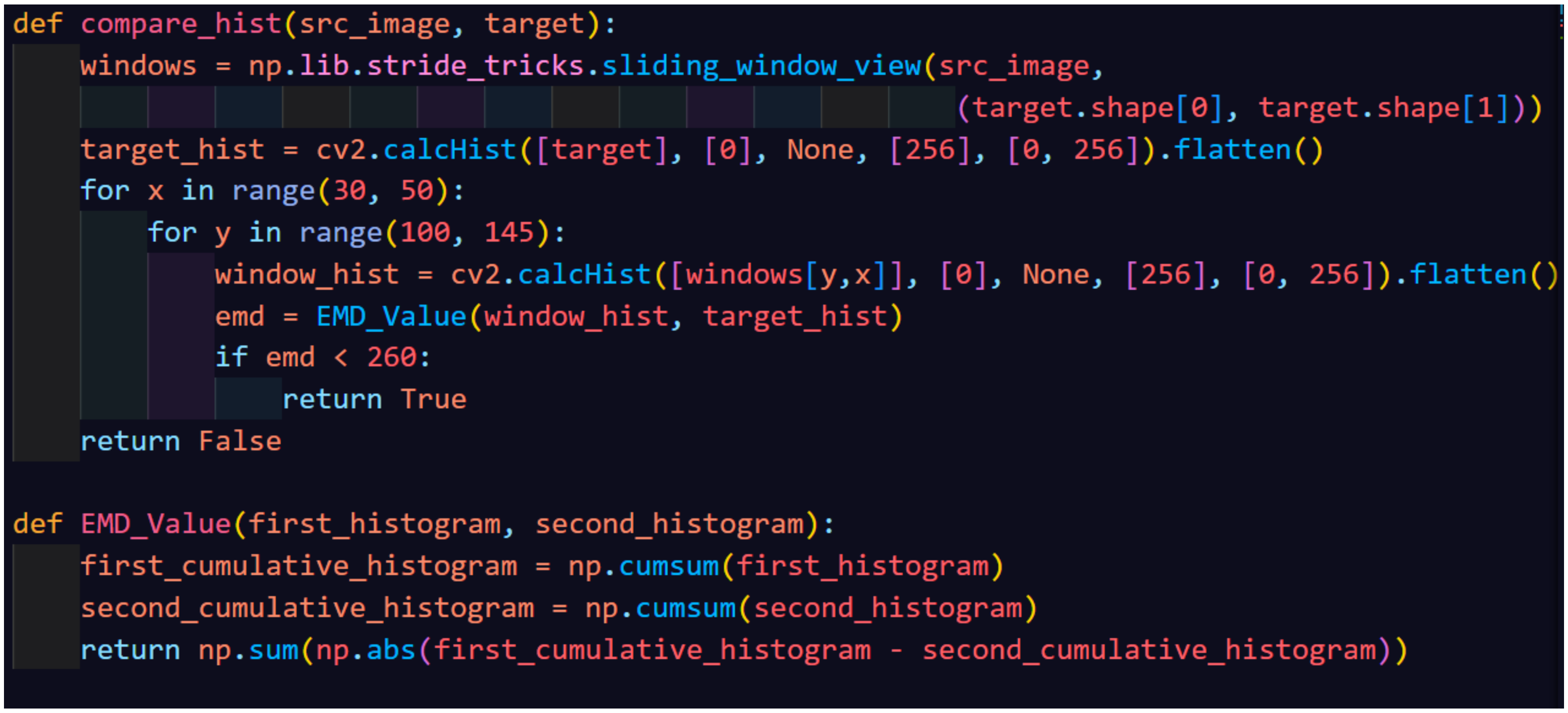


```
cv2.imshow(names[0], images[0])
cv2.imshow(numNames[0], numbers[0])
cv2.waitKey(0)
    stroyAllWindows()

    following print line is what you

    t(f'Histogram {names[id]} gave {
```

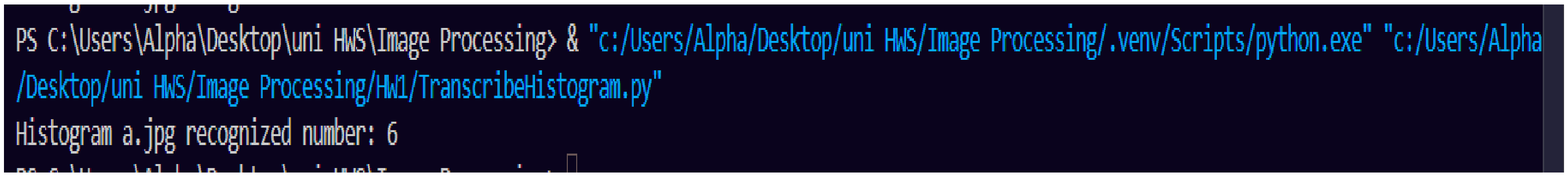
```
a.jpg
ממוצע: 49.895
סטיית תקן: 16.839
תאריך: 31/08/2023
```

Section C:

We start by implementing the function compare histogram by creating a sliding window that slides across the specified coordination in the code and we calculate the histogram for the source image then we start sliding the window and calculating each histogram for the current window, and then we use the EMD metric to see if the histograms match in order to find the matching number.



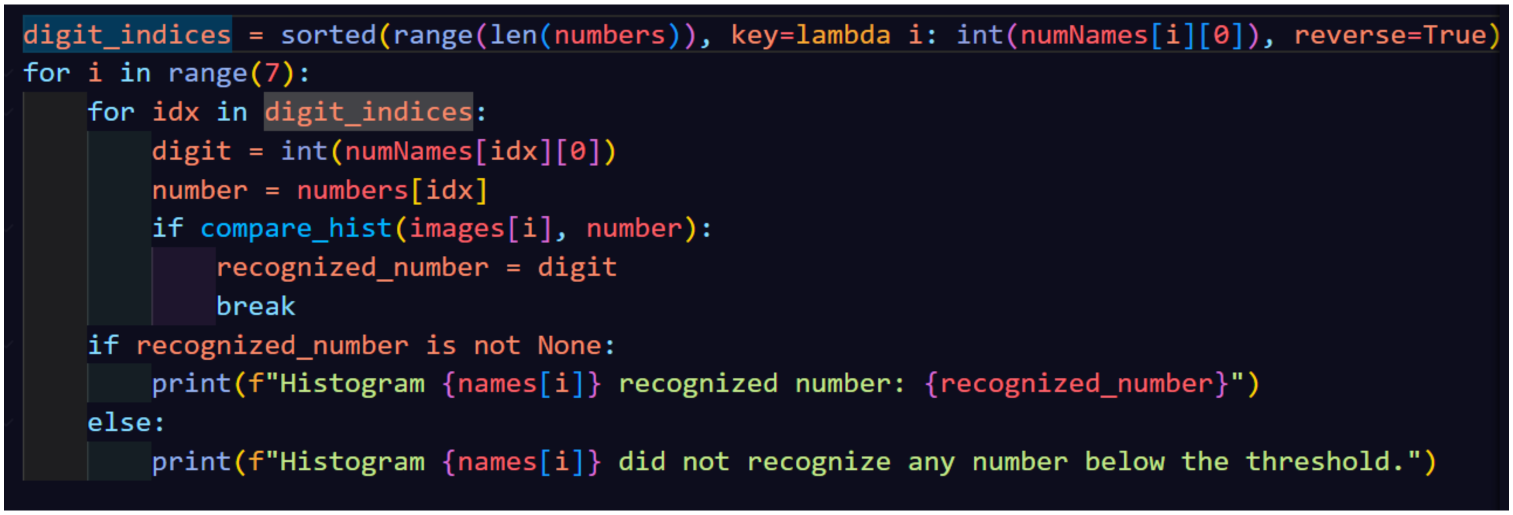
```python
def compare_hist(src_image, target):
    windows = np.lib.stride_tricks.sliding_window_view(src_image,
                                                       (target.shape[0], target.shape[1]))
    target_hist = cv2.calcHist([target], [0], None, [256], [0, 256]).flatten()
    for x in range(30, 50):
        for y in range(100, 145):
            window_hist = cv2.calcHist([windows[y,x]], [0], None, [256], [0, 256]).flatten()
            emd = EMD_Value(window_hist, target_hist)
            if emd < 260:
                return True
    return False

def EMD_Value(first_histogram, second_histogram):
    first_cumulative_histogram = np.cumsum(first_histogram)
    second_cumulative_histogram = np.cumsum(second_histogram)
    return np.sum(np.abs(first_cumulative_histogram - second_cumulative_histogram))
```



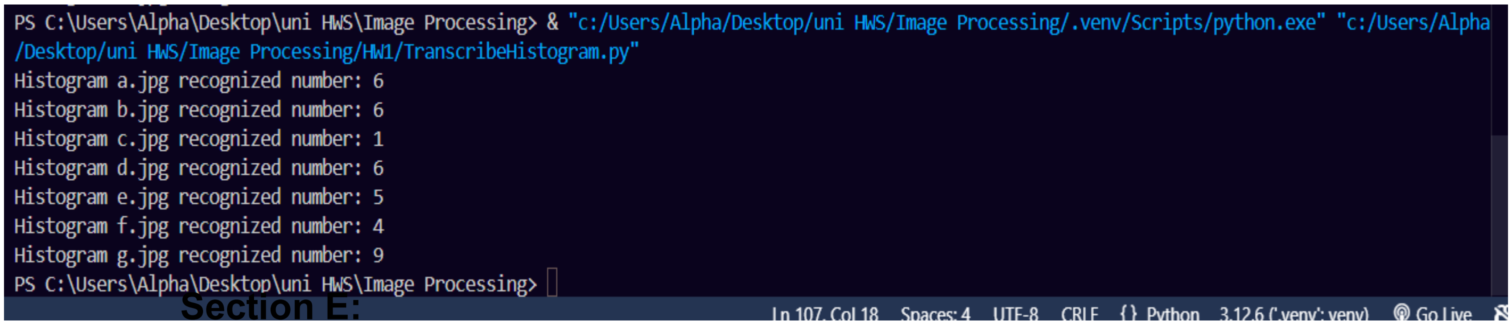
```
PS C:\Users\Alpha\Desktop\uni HWS\Image Processing> & "c:/Users/Alpha/Desktop/uni HWS/Image Processing/.venv/Scripts/python.exe" "c:/Users/Alpha
/Desktop/uni HWS/Image Processing/HW1/TranscribeHistogram.py"
Histogram a.jpg recognized number: 6
```

Section D:

In section d we simply just iterated over the images and printed the number they recognized



```
digit_indices = sorted(range(len(numbers)), key=lambda i: int(numNames[i][0]), reverse=True)
for i in range(7):
    for idx in digit_indices:
        digit = int(numNames[idx][0])
        number = numbers[idx]
        if compare_hist(images[i], number):
            recognized_number = digit
            break
    if recognized_number is not None:
        print(f"Histogram {names[i]} recognized number: {recognized_number}")
    else:
        print(f"Histogram {names[i]} did not recognize any number below the threshold.")
```



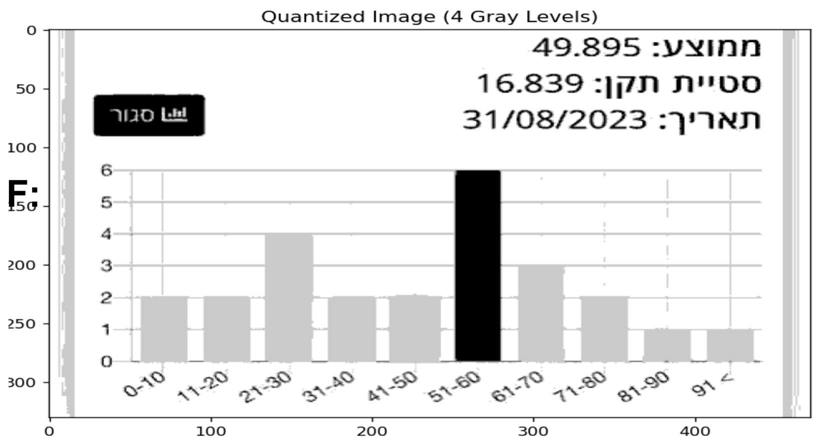
```
PS C:\Users\Alpha\Desktop\uni HWS\Image Processing> & "c:/Users/Alpha/Desktop/uni HWS/Image Processing/.venv/Scripts/python.exe" "c:/Users/Alpha
/Desktop/uni HWS/Image Processing/HW1/TranscribeHistogram.py"
Histogram a.jpg recognized number: 6
Histogram b.jpg recognized number: 6
Histogram c.jpg recognized number: 1
Histogram d.jpg recognized number: 6
Histogram e.jpg recognized number: 5
Histogram f.jpg recognized number: 4
Histogram g.jpg recognized number: 9
PS C:\Users\Alpha\Desktop\uni HWS\Image Processing>
```

Section E:

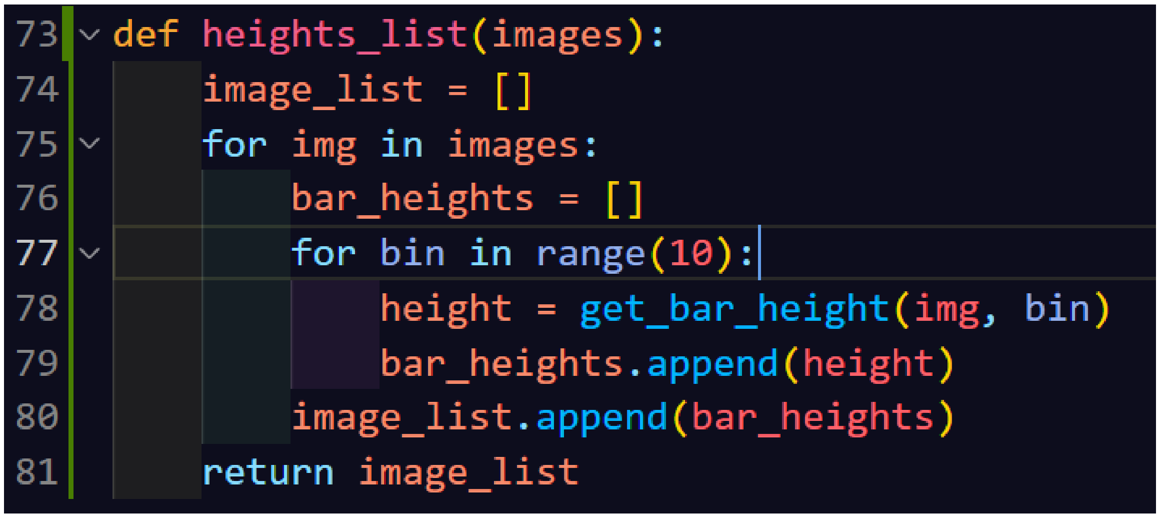
In this section we called the quantization function on the first image and chose to work with 4 gray levels due to the colorization of the images, and here is the quantized image with 4 levels of gray.

Also we didn't lose any bars when we worked with 4 levels of gray.

Section F:



We created a function that goes over the images and calculates for each image the heights of the bars by calling the function get_bar_height.

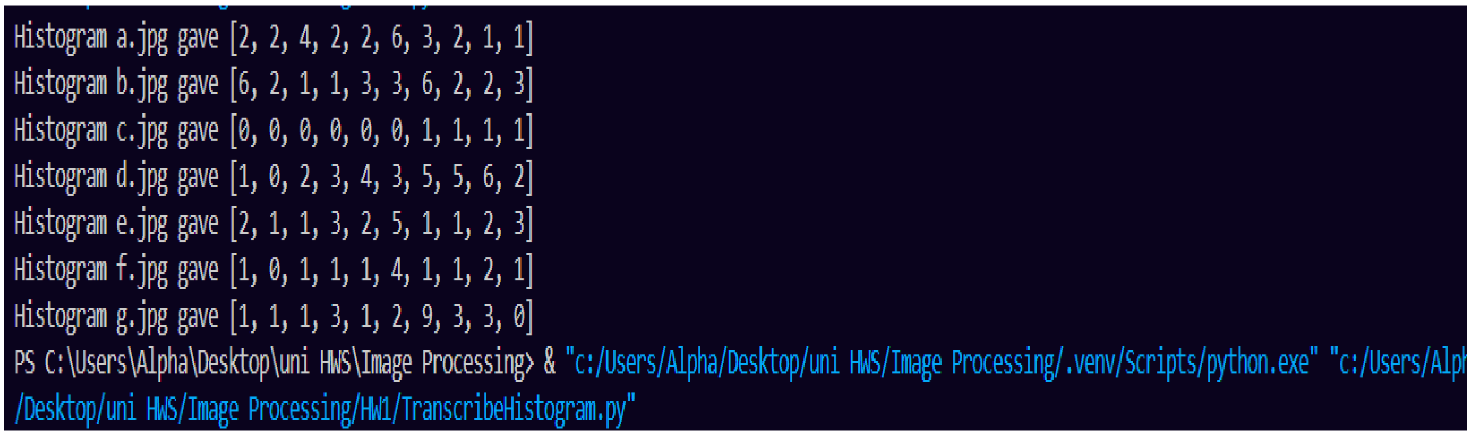


```python
def heights_list(images):
    image_list = []
    for img in images:
        bar_heights = []
        for bin in range(10):
            height = get_bar_height(img, bin)
            bar_heights.append(height)
        image_list.append(bar_heights)
    return image_list
```

Section G:

By using the topmost number we found earlier and the heights of the bars, we were able to find the number of the students for each bar and then we printed.

We implemented that by creating a function that finds the number of students per bar and a function that gives max amount of students.



```
Histogram a.jpg gave [2, 2, 4, 2, 2, 6, 3, 2, 1, 1]
Histogram b.jpg gave [6, 2, 1, 1, 3, 3, 6, 2, 2, 3]
Histogram c.jpg gave [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
Histogram d.jpg gave [1, 0, 2, 3, 4, 3, 5, 5, 6, 2]
Histogram e.jpg gave [2, 1, 1, 3, 2, 5, 1, 1, 2, 3]
Histogram f.jpg gave [1, 0, 1, 1, 1, 4, 1, 1, 2, 1]
Histogram g.jpg gave [1, 1, 1, 3, 1, 2, 9, 3, 3, 0]
PS C:\Users\Alpha\Desktop\uni HWS\Image Processing> & "c:/Users/Alpha/Desktop/uni HWS/Image Processing/.venv/Scripts/python.exe" "c:/Users/Alph
/Desktop/uni HWS/Image Processing/HW1/TranscribeHistogram.py"
```