

Image Processing HW_1

Student -1- : Mostufa Jbareen (212955587) מוסטפא גבארין

Student -2- : Mohammed Egbaria (318710761) מוחמד אגבאריה

Problem 1:

a) These are some considerations on how many megapixels the camera should have or what resolution a computer-generated image should be:

- Image Quality: Higher megapixels / resolution generally results in better quality images.
- Storage: Higher megapixel / resolution images result in larger file sizes. This impacts storage requirements
- Hardware Capability: The rendering capabilities of the hardware play a crucial role. More powerful GPUs can handle higher megapixels / resolutions and complex scenes without sacrificing performance.
- Processing Power: Processing higher megapixel / resolution images demands more computational power. Cameras need to balance between resolution and the camera's processing capabilities.

We can see that there is a trade-off between image quality, performance and storage.

b) These are some of the considerations that go into deciding how strong the quantization of an image is:

- Bit Depth and Processing Power: Bit depth determines the number of quantization levels available. Higher bit depths allow for more fine-grained quantization and, consequently, a wider range of colors and shades. Older hardware might be limited to lower bit depths due to memory and processing constraints.
- Color Accuracy and Image Quality vs Storage: Higher bit depth results in more color accuracy in the picture and more quality pictures, but on the other hand it also results in increase the size of the image. in some cases, artists or designers may intentionally choose stronger quantization for aesthetic reasons, even though it results in "Heavy" picture
- Display Capabilities: Older screens or devices had limited color reproduction capabilities, and exceeding these limits would be wasteful in terms of both processing power and storage.
- Hardware Capabilities: Older hardware had limitation in terms of memory and processing power which led into strong quantization.

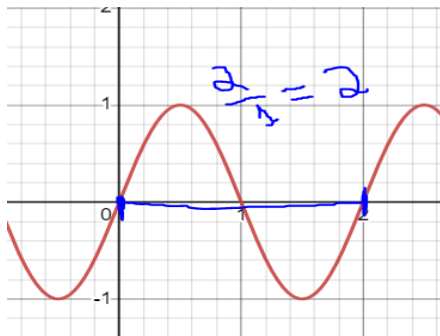
Problem 2:

a. $\lambda = \frac{2}{k}$

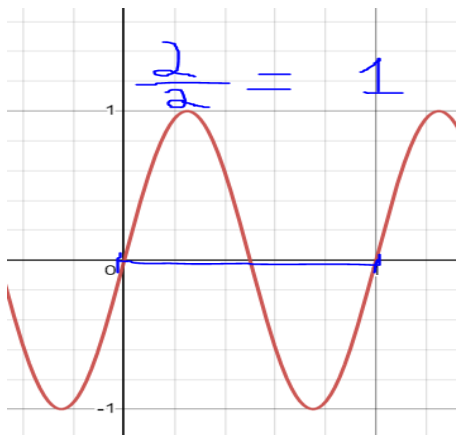
As we saw in tutorial 1 – we know that the function $\sin(k\pi x)$ has $\frac{2}{k}$ cycle, and therefore the wavelength for both A values $\{0.25, 2\}$ is $\lambda = \frac{2}{k}$, because the wavelength is independent from A values, and it is dependent only on k values in this situation as we saw in tutorial 1.

Examples:

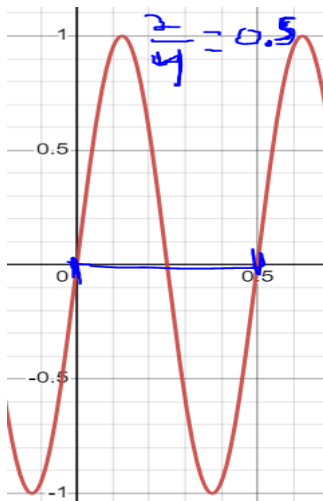
K = 1:



K = 2:



K = 4:



b. $A = 0.25 \rightarrow 2 \geq k$, $A = 2 \rightarrow 0.25 \geq k$

In the previous section we got that the wavelength is $\frac{2}{k}$, so from ($frequency = \frac{1}{wavelength}$), we get that the frequency for both A values is $\frac{k}{2}$ HZ.

Calculating f_{sample} for both A values:

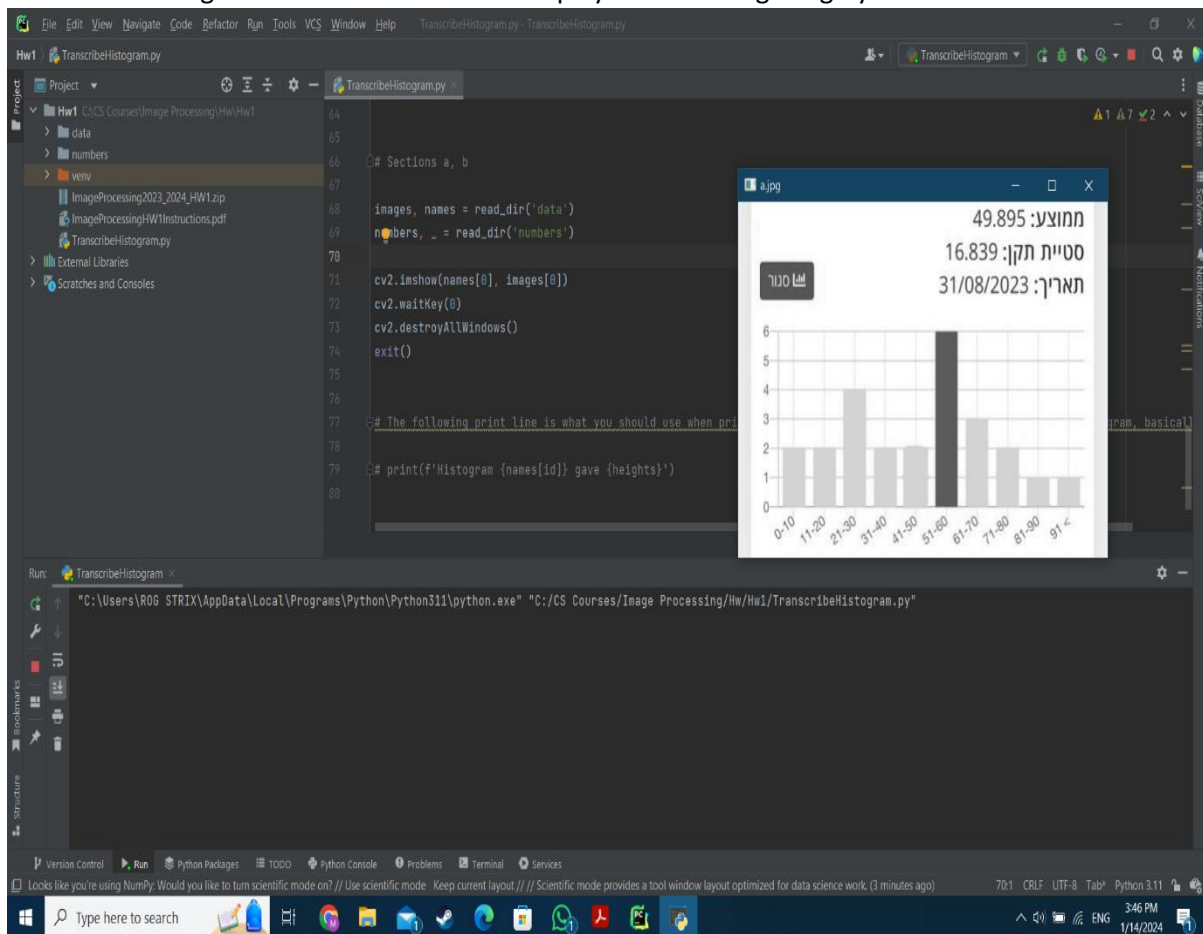
For $A = 0.25$: knowing that for every 1 cm in the picture we sample two times (1 cm divided into four columns $\rightarrow 0.25$ (sampling), 0.25 (space), 0.25 (sampling) and 0.25(space)), so the $f_{sample} = 2$. From Nyquist in order to restore the image we need to achieve $f_{sample} \geq 2 * frequency$, in our case that means $2 \geq 2 * \frac{k}{2} \rightarrow \rightarrow \rightarrow 2 \geq k$.

For $A = 2$: similar to $A = 0.25$, we get that for $A = 2 \rightarrow f_{sample} = 0.25$, and from Nyquist in order to restore the image we need that $f_{sample} \geq 2 * frequency \rightarrow \rightarrow \rightarrow 0.25 \geq k$.

Problem 3:

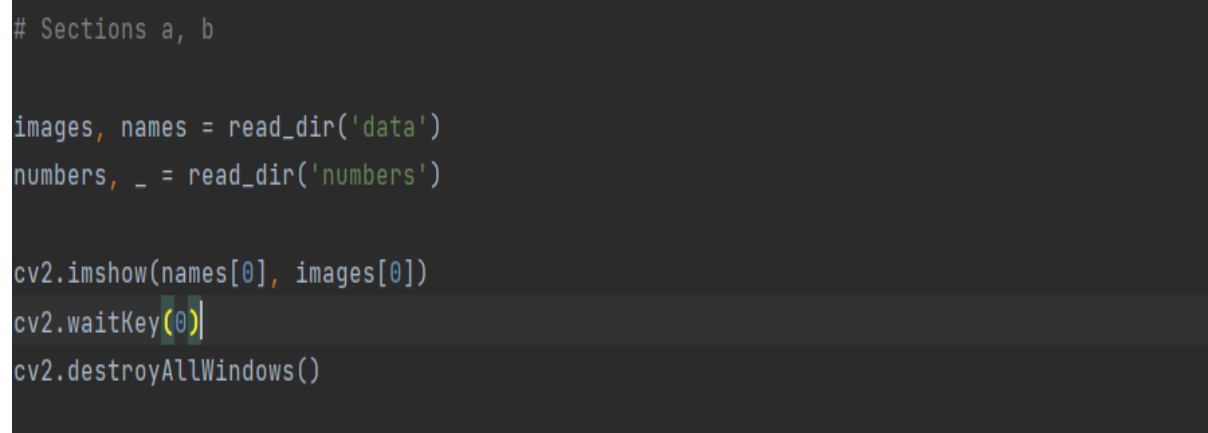
a:

we read the images data on line 68 and we display the first image in grayscale later in lines 71-73.



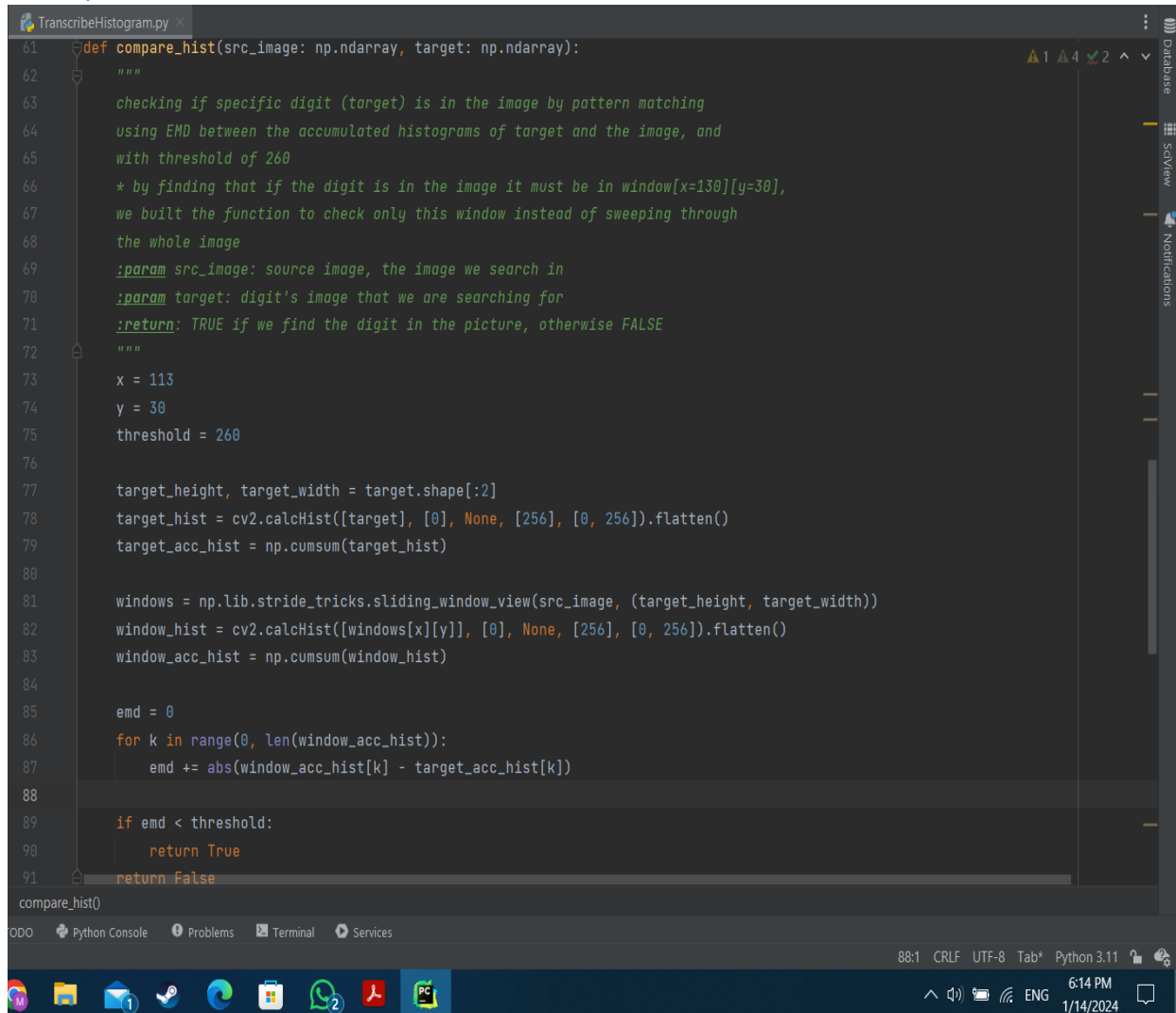
b:

we read the digits data in `(numbers, _ = read_dir('numbers'))` as needed using `read_dir` help function.



c:

implementing compare_hist function as required, check the documentation in green lines to know the way the function works.



```
61 def compare_hist(src_image: np.ndarray, target: np.ndarray):
62     """
63     checking if specific digit (target) is in the image by pattern matching
64     using EMD between the accumulated histograms of target and the image, and
65     with threshold of 260
66     * by finding that if the digit is in the image it must be in window[x=130][y=30],
67     we built the function to check only this window instead of sweeping through
68     the whole image
69     :param src_image: source image, the image we search in
70     :param target: digit's image that we are searching for
71     :return: TRUE if we find the digit in the picture, otherwise FALSE
72     """
73     x = 113
74     y = 30
75     threshold = 260
76
77     target_height, target_width = target.shape[:2]
78     target_hist = cv2.calcHist([target], [0], None, [256], [0, 256]).flatten()
79     target_acc_hist = np.cumsum(target_hist)
80
81     windows = np.lib.stride_tricks.sliding_window_view(src_image, (target_height, target_width))
82     window_hist = cv2.calcHist([windows[x][y]], [0], None, [256], [0, 256]).flatten()
83     window_acc_hist = np.cumsum(window_hist)
84
85     emd = 0
86     for k in range(0, len(window_acc_hist)):
87         emd += abs(window_acc_hist[k] - target_acc_hist[k])
88
89     if emd < threshold:
90         return True
91     return False
compare_hist()
```

d:

checking in decreasing order from 9 to 0 if the current digit is the highest digit in the histogram, if it is the highest, we break out the loop.

I made sure that all the images in the 'data' directory give the correct result when using compare_hist on them by doing this to all images and by checking and print the digit that enters the condition below, but I removed the code cause it is not required for the assignment.



```
for i in range(len(numbers) - 1, -1, -1):
    if compare_hist(images[0], numbers[i]):
        break
```

e:

we quantize the colors in the image in order to find a threshold value which help us to turn the image to black and white image where all the bars in black (1) and the remaining part of the image is white (255).

looking at the images before quantizing we can see that there is 3 domain colors in the picture: white (background), psu-black(the selected bar), and gray (other bars), so if we quantize the picture into 3 colors seems the 3 domain colors will still domain in the quantized picture and nothing else, so we quantized to 3 colors, and after that in order to threshold we chose the threshold value to be `max(np(im[i]))` which it is the domain white in the picture. This way we get the all the bars and the histogram in black and all the background in white, we achieved the goal.

```
# section e
# quantizing the images to 3 colors then thresholding the images,
# we found the threshold value after examining the quantized image,
# everything <= 213 -> 1, and <= 213 -> 255 in order to get a better image to do the search on.
im = quantization(images, 3)
black = 1
white = 255
for i in range(0, len(im)):
    threshold = np.max(im[i])
    im[i] = np.where(im[i] < threshold, black, white)
```

f:

as mentioned in section f section, we did section f for all images (also g in one go, check g section):

calculating the pixel height for each bin in the image (`pixel_heights_list[10]`), the maximum pixel height (`max_pixel_height`), and the maximum digit in the histogram (`max_height`).

```
# doing section f, g for all images in one go.
# section f
for i in range(len(im)):
    # creating list of 10 items, in each slot the height of the bar in pixels.
    pixel_heights_list = [get_bar_height(im[i], num) for num in range(len(numbers))]
    max_pixel_height = max(pixel_heights_list)

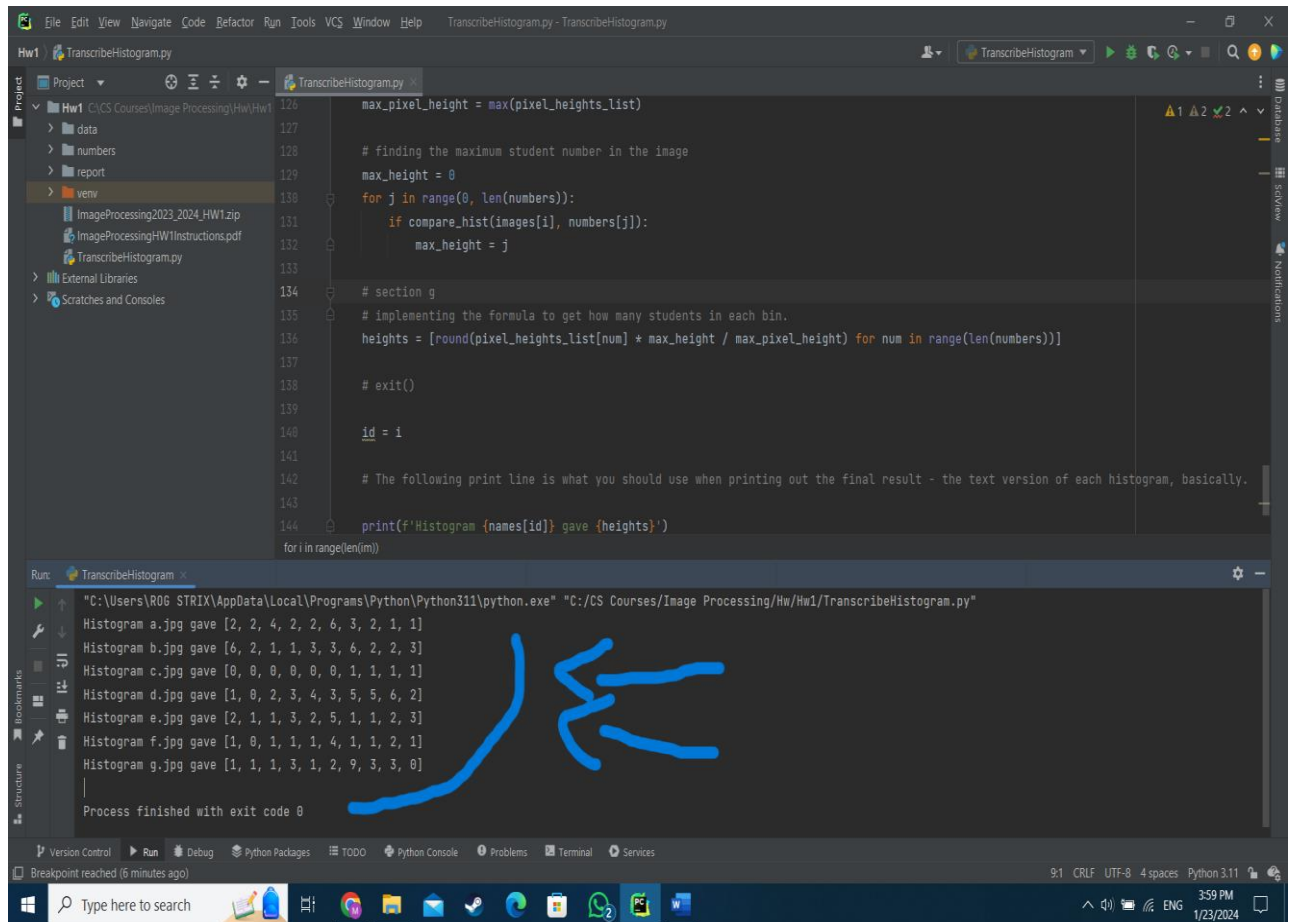
    # finding the maximum student number in the image
    max_height = 0
    for j in range(0, len(numbers)):
        if compare_hist(images[i], numbers[j]):
            max_height = j
```

g:

as mentioned below we implemented the formula to get how many students in each bin with giving attention if the histogram is empty (0 is the highest number in the histogram, not dividing by 0)

```
# section g
# implementing the formula to get how many students in each bin.
if max_height != 0:
    heights = [round(pixel_heights_list[num] * max_height / max_pixel_height) for num in range(len(numbers))]
else:
    heights = [0 for num in range(len(numbers))]
```

Final Result (Transcribe all the images):



The screenshot shows a VS Code editor window with a Python script named `TranscribeHistogram.py`. The script is located in a project folder `Hw1` under `C:\CS Courses\Image Processing\Hw1\Hw1`. The script's purpose is to transcribe images by finding the maximum student number in the image, comparing histograms, and implementing a formula to get the number of students in each bin. The script is written in Python 3.11 and uses the `exit()` function to terminate the program.

```
126 max_pixel_height = max(pixel_heights_list)
127
128 # finding the maximum student number in the image
129 max_height = 0
130 for j in range(0, len(numbers)):
131     if compare_hist(images[i], numbers[j]):
132         max_height = j
133
134 # section g
135 # implementing the formula to get how many students in each bin.
136 heights = [round(pixel_heights_list[num] * max_height / max_pixel_height) for num in range(len(numbers))]
137
138 # exit()
139
140 id = 1
141
142 # The following print line is what you should use when printing out the final result - the text version of each histogram, basically.
143
144 print(f'Histogram {names[id]} gave {heights}')
for i in range(len(im))
```

The output of the script is displayed in the Run console, showing the histogram results for images a.jpg through g.jpg. The output is as follows:

```
"C:\Users\ROG STRIX\AppData\Local\Programs\Python\Python311\python.exe" "C:\CS Courses\Image Processing\Hw1\TranscribeHistogram.py"
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]
Process finished with exit code 0
```

Handwritten blue annotations are present on the output, including a large bracket under the first two histograms and a large 'E' shape under the last two histograms.