COL783: Digital Image Processing

Assignment 1

Artistic Image Enhancement, and Style Transfer

Tushar Verma (2022AIY7514) & **Naimish Pintukumar Machchha** (2022AIY7512)

Part 1: Simple Image Enhancement:

Assumption: Our program expects input as 3 channel RGB images

Methodology: This part was divided into three parts.

The first part **shadow map generation** was straight forward implementing the equations was easy, we tried to use numpy as best as be could so that our code remain efficiency, also we were able to optimize the threshold calculation part in equation 6 by calculate mean (u1, u2) and weights (W1,W2) with simply adding the new index and removing the old index inside one loop instead of calculating these sums again and again,
The hyperparameter λ for the shadow map was calculated based on trial and error method.

The second part **line draft generation** was also simple direct equations were given so implementing them was not that difficult, only the bilateral filtering we implemented is based on brute force approach and was very slow for larger kernel size and sigma values , due to time constraints we were not able to optimize it further.
Threshold for the final line draft was calculated based on trial and error method.

The final part **color adjustment step,** very clear instructions for creating a chromatic map and updating the shadow map based on its corresponding values with an equation was given, implementing it was simple.
Hyperparameter ρ was calculated based on trial and error method within the given range [0.005, 0.2].

It was mentioned that performing the above may cause undesired changes to the overall brightness of the input image, but in our case this doesn't happen so we didn't perform and saturation correction.
Hyperparameter β was calculated based on trial and error method.

Hyperparameters: table below

Hyperparameters	λ	Shadow map T	Line Draft T	ρ	β	Bilateral (k,σ,σ)
Best value	0.3	0.27	11	0.021	0.1	(7,50,50)

Results:

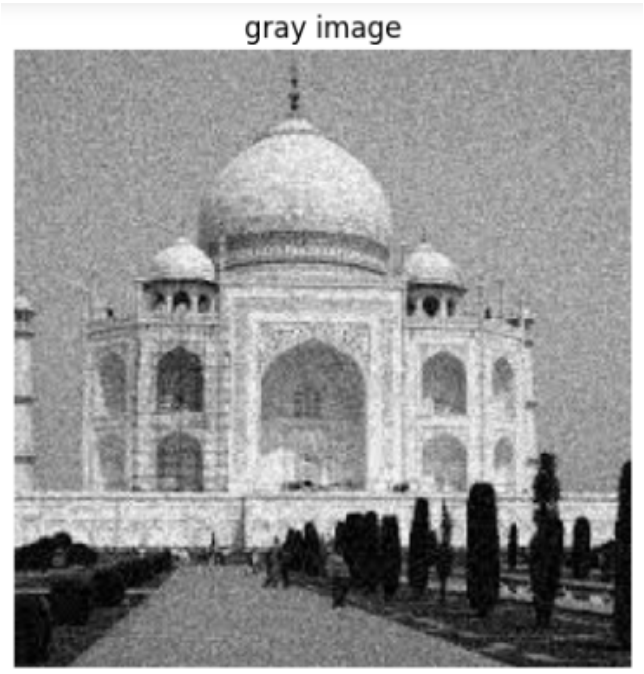
1.1. Artistic Enhancement Step:

Merging shadow map with original image over different lambda values

Observing best result as **lambda = 0.2 to 0.3**

1.1.2. Line Draft generation:

Using a noisy image to demonstrate it’s working with the best parameters



Observing best result at

Kernel Size = 7

Intensity Sigma = 45

Spatial Sigma = 45

Result of SobelX and SobelY filters

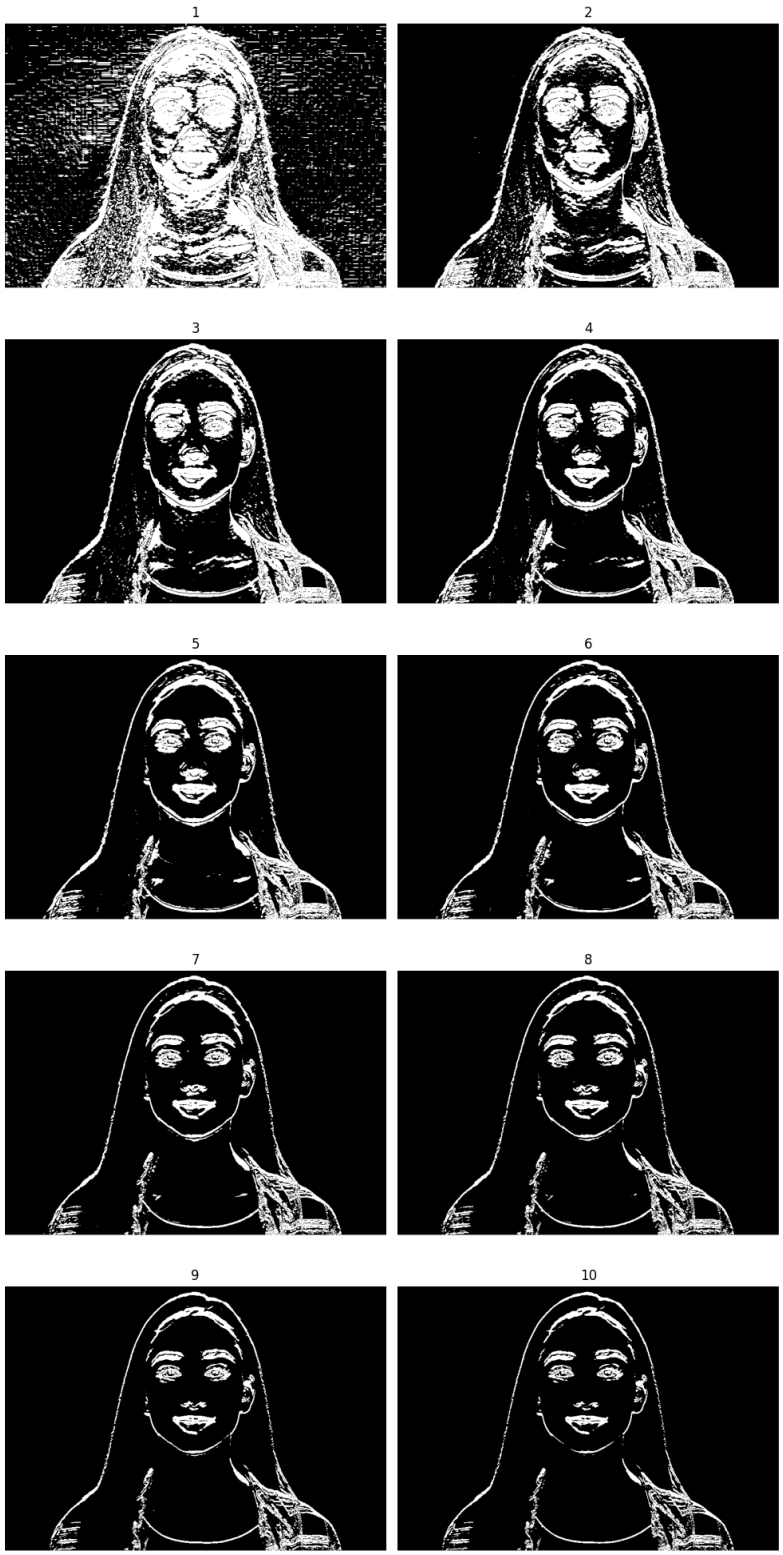
horizontal edge map

vertical edge map

Final edge map

edge map

Line Draft over different threshold



Observing best result as **lambda = 8-12**

final line draft

1.2. Color Adjustment Step:

Chromatic Map

Merging shadow map and CMap over differed lambdas

Observing best result at **lambda = 0.02-0.04**

Merging enhanced image and line draft over different beta

Observing best value at **0.1-0.2**

Final Artistic Output

Conclusion:

The hyperparameters are tuned to give the best appearance to the given input image, we found out that for different images hyperparameters may need to readjusted for best appearance

Part 2: Quantized Rendering:

Assumption: We assumed that image quantization can be done using any number of colors not just in 2^n colors so we implemented both the method one which is quicker and handles 2^n cases and other which is bit slower and handles rest of the cases

Methodology: This part was divided into two parts.

In the first part we are asked to perform **color quantization** using the **median cut** method, as mentioned earlier we implemented it in two ways.
In the **first** way we sort the pixels according to the intensity values of the most dynamic channel then we divide it into two equal buckets and repeat the process until desired depth n is not reached.
In the **second** way we sort the pixels according to the intensity values of the most dynamic channel then using max heap we find the highest value bucket (which is nothing but multiplication of the range of all the three channels) and then again divide it using the same process until n buckets are created.
At last we assign the pixels of an image to the average RGB value of the bucket to which the pixel belongs.

In the second part we had to implement **Floyd Steinberg dithering** and apply it on the quantized image. It was easy to implement since equations were easily available on the wikipedia page, we also perform some numerical optimization using numpy for faster calculation.

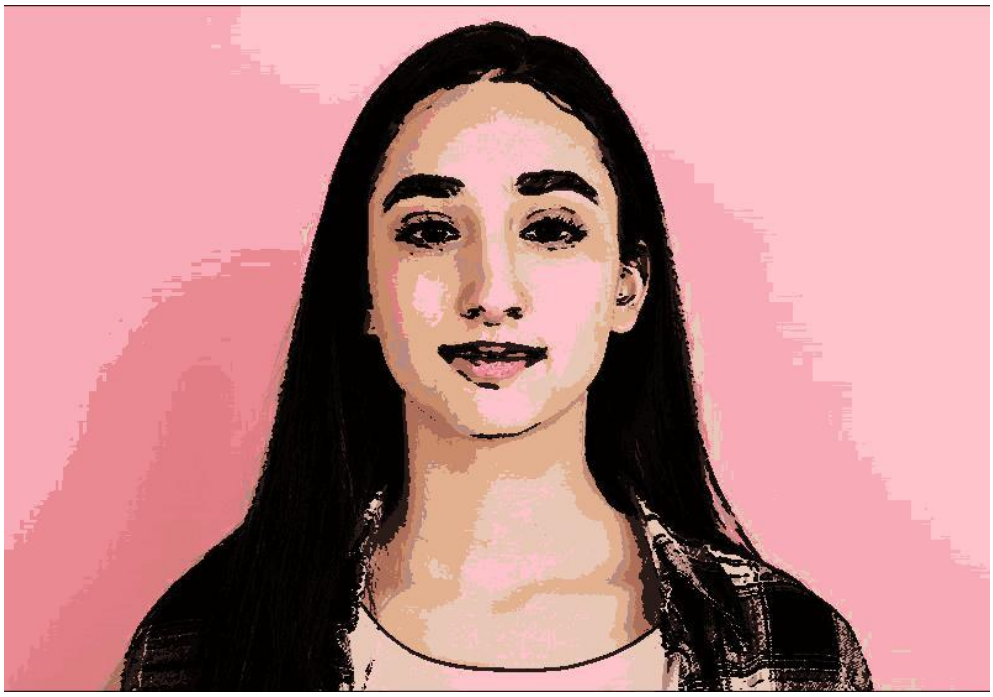
Hyperparameters: No hyperparameters in this part of the assignment

Results:

Median cut color quantization:

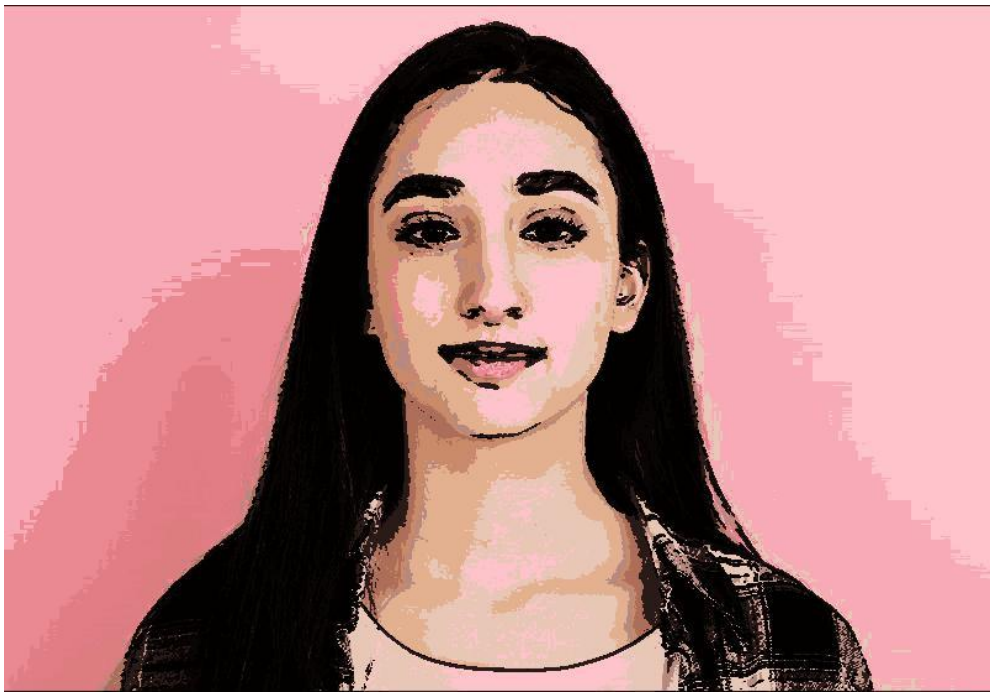
Input image:

Output image: **12 colors**



Floyd Steinberg dithering:

Input image: **12 colors**



Output image:

Display of quantization on artistic image with different color count

Conclusion: No further optimization is needed here.

Part 3: Artistic Style Transfer:

Assumption: We assumed that when comparing pixel neighborhood with swatches neighborhood, we don't need to consider every point in the swatch, so we picked random neighborhoods from each swatch.
Instead of filling just a single pixel of the target at a time we are filling the entire neighborhood of that point this is faster.
Users are not selecting very small swatches

Methodology: This part was divided into two parts.
The first part was standard **color transfer** based on **Luminance** value of the image
We followed the following paper
"Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images**".**

As mentioned we first converted the input image to LAB space and performed jittered sampling to extract 200 samples and sorted it based on luminance value, than performed luminance mapping by changing the mean and standard deviation of the source image as target image, than we calculated neighbor statistics of the target image for each pixel and calculated the final score map using weighted average of luminance (50%) and neighbor statistics (50%)
Finally for each pixel in the score map we used the binary search algorithm to find the nearest sample and assign it's a,b values (LAB space) to the corresponding pixel of the target map.

The second part was **color transfer using swatches.** Here first we accepted manual swatches from users to do this we created a simple GUI tool, after that we first performed color transfer between corresponding source and target swatches exactly as in the first part. Now using these colored target swatches we extracted a random 200 sample pixel using jittered sampling and n neighborhood samples from each swatch. Finally for each target pixel we compared its neighborhood with each neighborhood sample of each swatch and calculated error as given in the paper, swatch with the least error is selected and just as in first part we transferred color to the target image using binary search on best swatch samples for each target pixel.

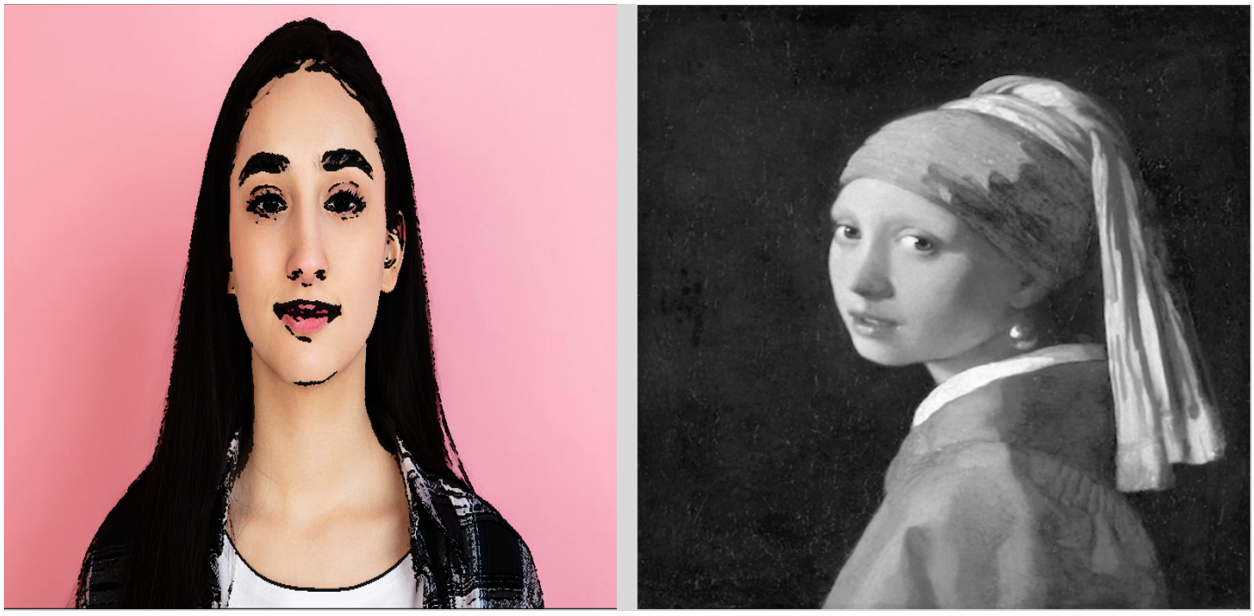
Hyperparameters: table below

Hyperparameters	Sampling Size	Neighbour Statistic kernel	Neighborhood size	Neighbor count	alpha
Best value	40-50	5	5-7	100	0.6

Results:

Standard color transfer:

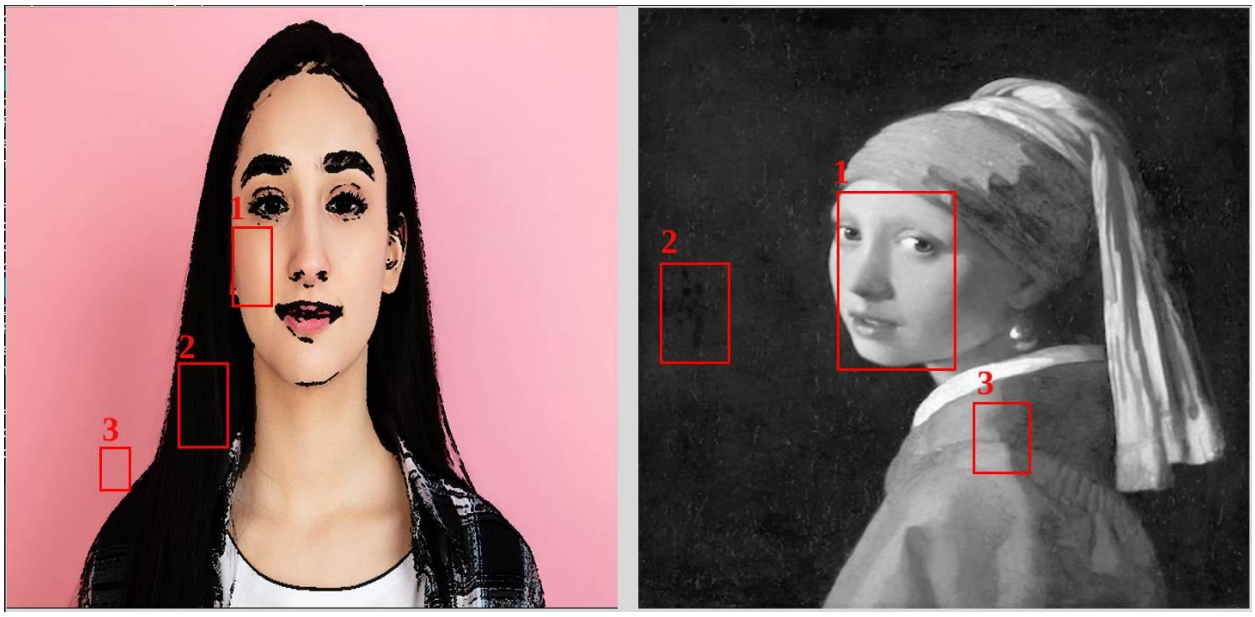
Source and Target images



Output image

Color transfer using swatch:

Swatches selected



Output image

Colored swatches

1

2

3

Conclusion: Conclusion: The performance of the algorithm visually speaking depends highly on the swatch selection, and the Luminance of the target and source scene should be similar. We were hoping to implement KL divergence to compare neighborhood texture but because of time constraint we were unable to implement it.