CENG466
Fundamentals of Image Processsing

Take Home Exam 1

Ali Alper Yüksel
Department of Computer Engineering
METU
2036390

Ulaşcan Deniz Genç
Department of Computer Engineering
METU
2099034

I. Question 1 - Image Interpolation

A. What is Interpolation?

Interpolation is used for resizing or remapping images from one pixel grid to another. We need to resize the image when we need to increase or decrease the total number of pixels, whereas we need to remap the image when we distort or rotate the image.

Interpolation works by using known pixels' values to estimate unknown pixels' values. To achieve this, we usually use nearest pixels or neighborhood of pixels.

B. Bilinear Interpolation and Bicubic Interpolation

Bilinear interpolation is the one of interpolation techniques that uses 4-neighborhood. Goal to achieve is to find average value of unknown pixel from 4-neighborhood of that pixel with known values.

Let (x, y) denotes the coordinates of unknown pixel and let $\vartheta(x, y)$ is the intensity value at that coordinate of pixel. We can use (1) to find value of that unknown pixel. Four coefficients can be found bu using values of 4-neighbors of unknown pixel.

$$\vartheta(x, y) = ax + by + cxy + d \tag{1}$$

Bicubic interpolation is the interpolation technique that uses 16-neighbors of unknown pixel. The intensity value of point (x, y) is obtained from the equation:

$$\vartheta(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j \tag{2}$$

Smoothness and reliability of bilinear and bicubic interpolations depends on the values of neighboors of unknown pixel. However, the computation is more complex in bicubic interpolation. If time is not important, bilinear interpolation can be the better way for continuous-tone images, but it reduces contrast (sharp edges). That means we can encounter with more blurred output image for input images which includes so many sharp edges.

C. Implementation of Algorithm

First, we take sizes of original input images A1, A2 and A3. Then, we resize shrinked images to sizes of original input images with bilinear and bicubic interpolation methods by using $imresize$ function. Finally, we compute the euclidean distances between bilinear interpolated output and original image, and between bicuibic interpolated output and original image.

D. Usage of $imresize$ function

When we were writing the code of solution, we used $imresize(param1, param2, param3)$ to resize the shrinked image. Function takes 3 parameters. First parameter (denoted as $param1$) is input image that will be resized. Second parameter (denoted as $param2$) is the scale time size of input image. Third parameter (denoted as $param3$) specifies the interpolation method that will be used.

E. Computation of Euclidean Distance

We computed Euclidean distance by taking square root of sum of squared distances of two images' same positioned pixels. By the way, our computation needs two same sized image. If the explanation is not clear enough, this formula can clarify the Euclidean Distance computation of two MxN sized image:

$$EUD(f_1(x, y), f_2(x, y)) = \sqrt{\sum_{x=1}^{M} \sum_{y=1}^{N} [(f_1(x, y) - f_2(x, y))^2]} \tag{3}$$

Closeness of Euclidean distance to zero means the better similarity between two images. It is useful to determine which interpolation method is better for enlarging shrinked image to original size.

F. Results and Comments

The comparison between bicubic interpolated output and bilinear interpolated output is not obvious to see. Actually, we expected to see that bilinear interpolation is the better method

to use for all these images since we thougt that there is not too much sharp edges on original images.

When we computed Eucledian distances, the distance value of bilinear interpolation was smaller than the distance value of bicubic interpolation for A1 and A2, as we expected. However, for A3, there was a reverse situation.

Fig. 1. Bilinear interpolated and enlarged output of A3.

Fig. 2. Bicubic interpolated and enlarged output of A3.

In our opinion, the reason of this situation is that boy's hair in image makes so many sharp edges with the background (the wall) and face of boy. As we mentioned, bilinear interpolation is not useful for images with contrast. So, for image A3, we can find that Eucledian distance between the original image and bicubic interpolated enlarged output is smaller than the other.

II. Question 2 - Histogram Processing

A. Purpose of Histogram Matching

The purpose of histogram matching is to transform an image by matching it's histogram with the histogram of reference image.

For instance, if the given image's white pixels' density is same with reference image's density of green pixels, white pixels of given image will be transformed to green color since their histograms are same.

B. Implementation of Algorithm

In mathematical way, first we need to compute input and reference images' histogram. Then we need to compute cumulative distribution functions of these images -$F_1()$ for input image and $F_2()$ for refence image- by dividing their cumulative histograms to input size. After then, for each gray level $G_1 \in$ [0,255], we need to find G_2 for which $F_1(G_1) = F_2(G_2)$. Finally, we need to map each pixel of input image by using mapping function: $M(G_1) = G_2$.

In algorithmic way, we computed histograms of target image and reference image. Following, we divided cumulative histograms of these images to sizes of them to find cumulative distribution functions. We used $cumsum$ to find cumulative histogram and $numel$ to find sizes.

Coding the mapping was the hardest part. We created 256x1 array to implement $M(G_1) = G_2$ function. In for loop, we found absolute minimums of differences of two cumulative distribution function arrays, then we took x-coordinate of reference image's c.d.f. (cumulative distribution function) array which has minimum c.d.f., then we applied this coordinate value into mapping array. Finally, we used this map array to transform target image to histogram matched image.

Other problem was that index of array in MATLAB starts from 1 and intensity values start from zero. It is very confusing when we do operation for intensity value 0 since we need to save result of the operation to an array, but index starts from 1. We solved this problem very easily -just add or dectease 1 in index operations-, but it can be very exhausting in coding.

C. Results and Comments

When we looked at B1 and B2, B1 has mostly white color pixels and B2 has mostly blue colored pixels. That means, when we performed histogram matching on B1 with reference image B2, we expected that B1 becomes more blueish. It happened as we expected.

Fig. 3. B1 image.

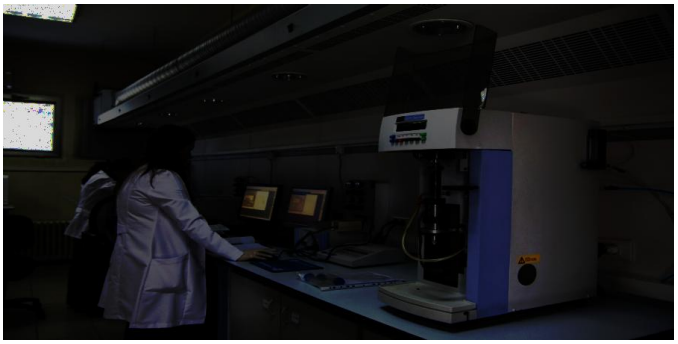

Fig. 4. Histogram matching on B1, when reference image is B2.

And for histogram matching on B2 with reference image B1, the expectation was more whitish B2 image. The results are as following:

Fig. 5. B2 image.

Fig. 6. Histogram matching on B2, when reference image is B1.

Also, when we looked at B3 and B4, we thought that B3 has mostly white color pixels and B4 has mostly blue colored pixels. That means, when we performed histogram matching on B3 with reference image B4, we expected that B3 becomes more blueish. It is not happened as much as we expected, that means B3 doesn't have mostly white color pixels.

Fig. 7. B3 image.

Fig. 8. Histogram matching on B3, when reference image is B4.

And for histogram matching on B4 with reference image B3, the expectation was more whitish B4 image. The results satisfied our expectation:

Fig. 9. B4 image.

Fig. 10. Histogram matching on B4, when reference image is B3.

III. Question 3 - Noise Elimination

A. Deciding the Type of Noise

When we analyzed images, we computed their histograms first. It was looked like probability density function of Gaussian Distribution. There were so many black and white pixels. On the other hand other colors are distributed in image according to histogram which cannot be obvious to see. As a result, we decided on Gaussian noise as the type of noise.

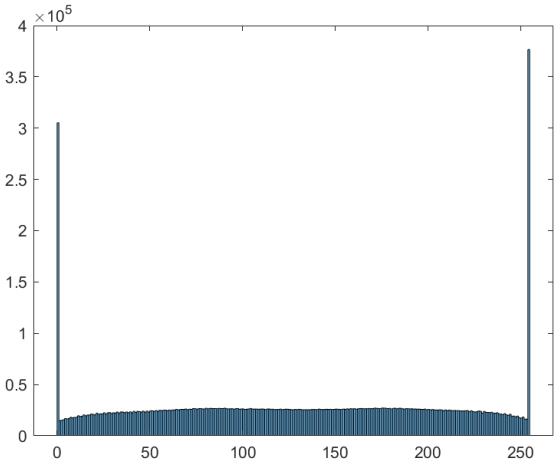

Fig. 11. Histogram of C1 image.

B. Solution of Noise Elimination

We extracted these images into three channels as red, blue and green channels, firstly we applied mean filter to these channels. However, we applied two separate mean filters since noise levels of this two images is not same. For C1 image, we convoluted channels with 9x9 matrix with ones which is multiplied by 1/81. On the other hand, for C2 image, we convoluted channels with 5x5 matrix with ones which is multiplied by 1/25. These values means that C1 image's noise level is higher than other's. Purpose of using mean filter is simple averaging pixel with surrounding pixels.

After we combined these channels to make single image, the result image showed that mean filter is not enough. It made image more blurred but didn't eliminate the noise entirely.

To solve this problem, we applied Gaussian filter after mean filter separately. For C1, we applied Gaussian filter with 12x12 matrix and sigma value $\sigma = 15$. For C2, we applied Gaussian filter with 9x9 matrix and sigma value $\sigma = 15$. Gaussian kernel values are like the values of Gaussian curve in two-dimension which makes the computation and filtering more accurately.

Outputs are shown in Figure 12 and 13.

Fig. 12. C1 image with mean and Gaussian filter.

C. Edge Detection

When we researched about edge detection techniques, we found that the most advantegous technique is Canny Filter technique. It gives more accurate and correct solution for edge detection.

To appliy Canny Filter edge detection algorithm, we created vertical and horizontal masks at first step as following:

$$MaskX = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, MaskY = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Then we convoluted these mask with image separately to find G_x and G_y, which are the vertical and horizontal gradients.

$$G_x = MaskX * InputImage \tag{4}$$

$$G_y = MaskY * InputImage \tag{5}$$

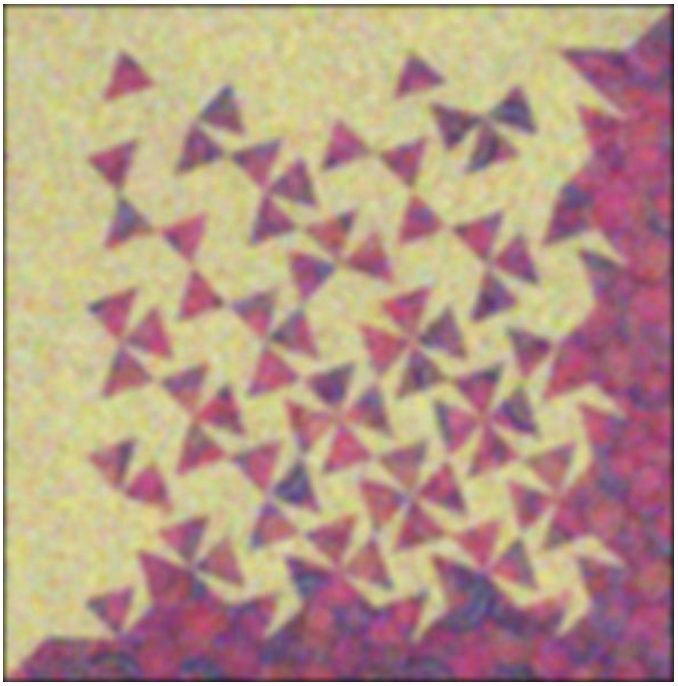

Fig. 13. C2 image with mean and Gaussian filter.

After this step, we calculated directions and orientations with $atan2$ formula of MATLAB. Then we adjusted negative directions to positive directions, nearing 0, 45, 90 and 180 degrees.

After, to find overall gradient magnitude at each pixel, we used this formula:

$$G = \sqrt{(G_x)^2 + (G_y)^2} \tag{6}$$

Then we applied non-maximum supression to magnitudes.

Finally, we applied hysteresis threshold with low value 0.075 and high value 0.175. This value was selected by testing values.

D. Results and Comments

For noise elimination, we expected that image will be blurred. However, we didn't expect that edges lose their sharpness. We tried all possible combinations on filters and we selected the best solution for this problem.

For edge detection, after we researched all edge detection techniques, we selected Canny Filter technique since of its advantages.

When we operated edge detection on C1 and C2 images, there were edges and white pixels, not only edges. The reason of this result is noise. If there were no noise on images, we could detect edges more easily.

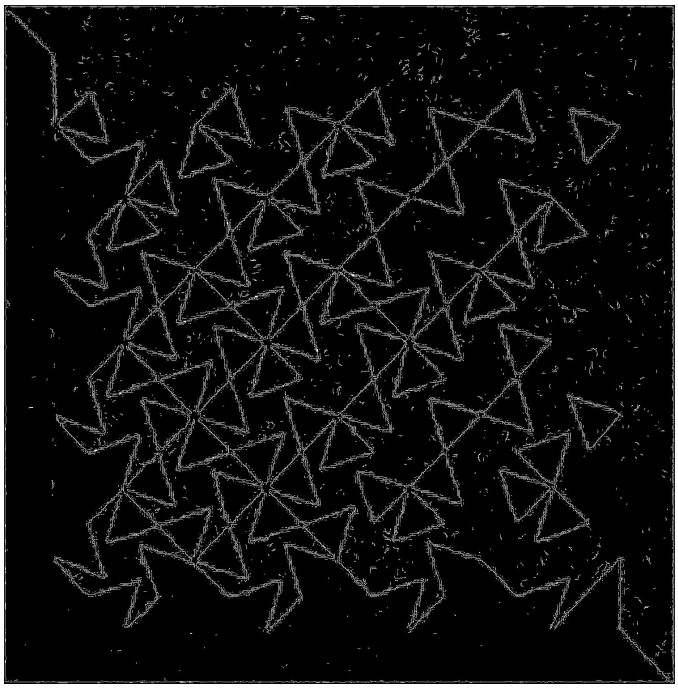

Fig. 14. Edge detection on C1 image.

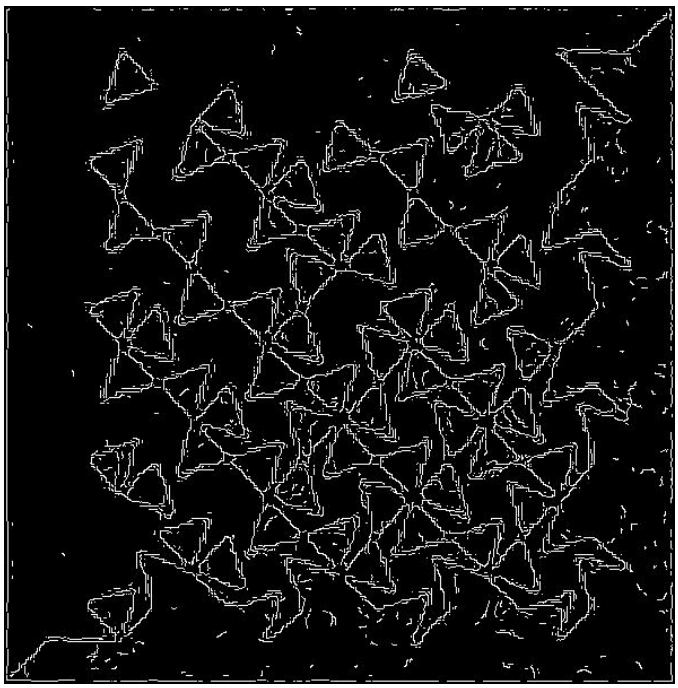

Fig. 15. Edge detection on C2 image.