CENG466 Digital Image Processing Take Home Exam 1 Solutions

Zumrud Shukurlu
Computer Engineering Department
Middle East Technical University
Ankara, Turkey
e2174761@ceng.metu.edu.tr

Yekta Demirci
Electrical and Electronics Department
Middle East Technical University
Ankara, Turkey
e209360@metu.edu.tr

***Abstract*—In this document, the solutions proposed for Take Home Exam 1 are explained and the results are discussed.**

***Index Terms*—affine transform, histogram equalization, histogram specification, convolution, edge detection**

I. INTRODUCTION

The general idea behind the MATLAB codes and the results are discussed in the following sections. In the first part by using brute force, some changed images are reconstructed. In the second part, by using histogram processing methods quality of the pictures are enhanced. And finally in the last part, edge detection is implemented by using two different kernel sizes. Also blurring filter is applied to the row images to improve edge detection results.

II. PART A - GEOMETRIC TRANSFORMATIONS

A. *A1.png*

The image is rotated counter-clockwise 90 degrees. Therefore basic matrix transpose is enough to re-construct the original image back.

B. *A2.png*

The main idea in implementation is to translate the center of the image to the coordinate plane origin so that it stays unchanged while being rotated, and translating it back to its original coordinates after the rotation. The overall transformation was written as the multiplication of inverse translation, rotation and translation matrices.
This image was rotated 45 degrees clockwise to match the reference image. The degree of rotation was determined by simple observation. Otherwise, a brute force algorithm would be used to test for each degree until reaching the desired result. After observing that, by using the help of cosine and sine functions the image is re-rotated. Some distortion occurs due to information loss during the 45 degree rotation. Interpolation could have been used to decrease such losses. However the information loss is not much therefore interpolation is not really required.

C. *A3.png*

The original image is added to the end of a pure black image. By finding the meaningful part of the matrix and getting rid of the pure black parts the original image is reconstructed easily.

D. *A4.png*

This image was horizontally sheared to the right by a factor of 0.5 to match the reference image. Multiplying the y coordinate value with this factor and adding it to the x coordinate of the image creates the shearing effect. Intuitively, we displace each point in a fixed position by an amount proportional to its distance from the line that is perpendicular to that fixed direction.
Again, the factor of shearing was observed by analyzing the input image with bare eyes. An alternative solution specific for this input image would be to find the index of the first nonzero element in the bottom row and calculate its ratio to the number of rows in the image. Another approach which is sent in the matlab script is: pure black pixels are found for each row and the remaining parts are shifted to left according to the pure black pixels. At the end original image is constructed with a small information loss. However it is hard to see the difference by bare eyes.
The main idea of shearing is to

E. *A5.png*

The given image is a reflected version of the original image. Changing order of the row elements in the opposite way is enough to construct back the original image. There is no information loss at the end.

III. PART B - HISTOGRAM PROCESSING

A. *Histogram Equalization*

The aim of Histogram Equalization is to find the probability of occurrence of each color intensity in the image and equalize those probabilities to obtain an image in which the intensities are more uniformly distributed.

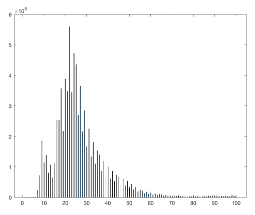
The algorithm for implementation was adapted from the textbook. In the MATLAB code, the image is scanned and the cumulative histogram of the image is calculated and kept in an $L-by-1$ vector named `cum` where L is the number of intensity values. Later, we divide histogram values by the size of the image to get cumulative distribution named as `cdf`. Finally, multiplying the values in `cdf` with $L-1$ gives us the transformation function. To obtain the final image, we write the intensity value from the transformation function to the (i,j)th pixel that was matched with the intensity of the (i,j)th pixel in the original image.

For the color image, the vectors was initialized as $L-by-3$ to keep separate values for three intensity levels and calculations were made separately for each level within an outer loop.

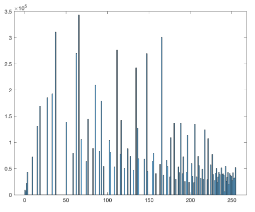
B. Discussion of histogram equalized images

1) B2.png: After applying histogram equalization, the contrast in the image notably increased. As the intensity values of the original image was accumulated in the center of the histogram, applying the equalization mapped some of those values to the edges equally. Consequently, a highly contrasted image was obtained.

2) B1.png: Histogram equalization produced a very bright image in comparison with the original image as it was poorly illuminated. There are a lot more pixels with high intensity values than the original image. (Fig. 1).



(a) Histogram of B1.png



(b) Histogram of B1_histeq_output.png

Fig. 1.

C. Histogram Matching

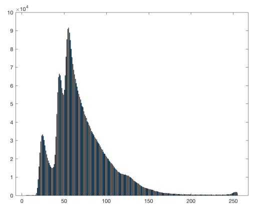
In this part, the images were modified to match the histogram of the reference image.

The histograms of both original and reference images were acquired with the same algorithm used to obtain histograms in the Histogram Equalization part. Afterwards, we find an output level j for each input level i, so that $H_z[j]$ best matches $H_x[i]$, where $H_z[j]$ is the cumulative histogram of the reference image and $H_x[i]$ is the cumulative histogram of the original image. In the code, we pair the i and j values in the `map` vector. Then we calculate the output image pixel values using this mapping.

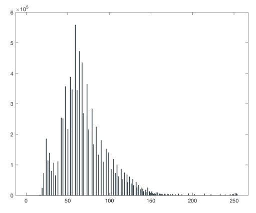
D. Discussion of histogram matched images

1) B2.png: The given reference image was a fragment of the original image, but the frequency of intensities with high values were more on average with respect to the original image. Matching the histogram of the original image with this image brought about the increase in the values of dark pixels of the original image.

2) B1.png: Matching the histogram of the input image with the reference image increased the illumination and produced an image with histogram similar to the histogram of the reference image. (Fig. 2)



(a) Histogram of B1_ref.png



(b) Histogram of B1_histmatch_output.png

Fig. 2.

IV. Part C - Edge Detection

In this part, two different sized kernels, in two different direction as a total of four kernels are applied to three grey scaled images to detect edges. After combining the results from both X and Y direction overall edge detection is observed.Then same procedures are applied to the same images, however this time the images are blurred before the processes. Therefore better edge detection is observed at the end.

A. the1_convolution.m

This is the implemented convolution taker function. The function takes two inputs named as 'input' and 'kernel' and convolves them. The important point here is, it is assumed only 2x2 and 3x3 sized kernels are fed to the function. Input size does not matter.

If the kernel is 2x2 sized matrix then the mirror of the last row and the column of the input is added to the input matrix. For instance if the input is 30x30 matrix it turns to be 31x31 matrix where 31st columns are exactly the same as 30th columns and rows. The reason is convolution results are applied to the input matrix according to the right top position of the kernel matrix. Therefore if mirrors are not taken last columns and rows are neglected through the process.

If the kernel is 3x3 sized matrix then the mirror of the 1st and last rows and columns are taken. For instance if the input is 30x30 matrix then it will be 32x32 where 1st rows and columns are the same as 2nd, 31st rows and columns are the same as the 32nd ones. Then convolution results are added to input pixels according to middle of the kernel locations. If the mirror would not be applied the pixels at the edges would have been neglected.

B. the1_part3_edgefilters.m

In this part, 3 input pictures are convolved with the 4 different kernels. The results are saved as png images.

C. the1_part3_edges.m

In this part results of the vertical and horizontal edges are combined by using Euclidean distance formula. Per each pixel the results from the X and Y directions are combined. As a result overall edge detection is obtained per image.

D. the1_part3_blur.m

Before applying the edge detection horizontally and vertically, and combining them, 3x3 sized Gaussian blur kernel is applied to the original images by using the convolution function. As a result slightly blurred images are obtained.

E. the1_part3_blurrededges.m

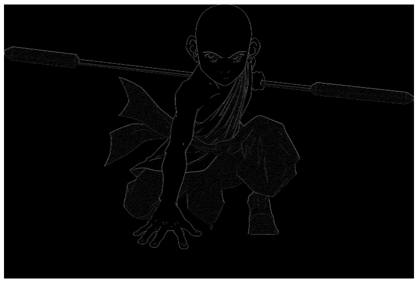
In this part, both 2x2 and 3x3 sized kernels are applied to the images in both X any Y directions, then results are combined to constract overall edges just like the previous cases. In the code there are 2 for loops doing almost the same thing. The only difference is the used kernels.

F. Differences between the edge filters

In the homework two different sized kernels are used as 2x2 and 3x3. As it can be foreseen 2x2 kernels return more detailed edges since smaller neighbourhood are used through the convolution.Furthermore 2x2 kernels are less affected by the noise and the results are less errored. However when 3x3 kernels are used the borders are more clear to see, lines are thicker since more area is included through the filtering. So depending on the application one can select its kernel size and magnitudes in it.

Also in order to get more detailed edges in horizontal and in vertical axes, a single kernel is used for each way rather than combining these kernels into one. Afterwards, at the end the results of the vertical and horizontal edges are combined. It is a clearer method than trying to get edges by only one mixed kernel.

As it can be seen in the figure 3, 2x2 and 3x3 kernels give slightly different results. The smaller kernel result can be seen in figure 3-a. The 3x3 kernel result can be seen in figure 3-b. The bigger kernel gives thicker edges as it is mentioned above. However there is more noise in the resulted image.


(a) Result of the R edge filter on image C1

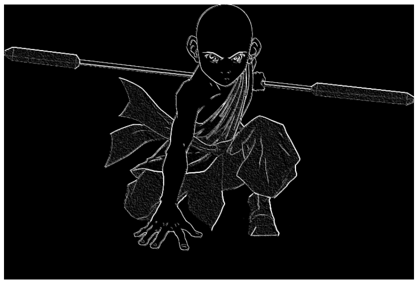

(b) Result of the S edge filter on image C1

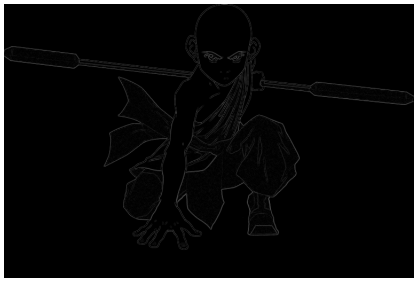
Fig. 3.

G. Differences between the blurred and non-blurred approach

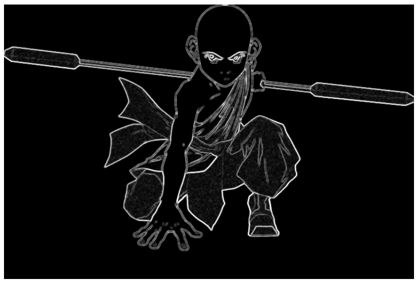
When the higher sized kernels are applied, noise appears in the final image. In order to avoid these noise and also smoother edges, pictures can be blurred before edge detection. For this purpose Gaussian blur kernel is applied. Which is 3x3 matrix as 1/16*(1 2 1; 2 4 2; 1 2 1). Basically, it slightly mixes neighbour pixels. Therefore edges are stressed and the non edge places are smoothed when the edge detection kernel is applied. As a result way better results are obtained at the end.

Depending on the application, if one needs a sensitive edge detection, s/he can blur the image before applying edge detection.

The resulted images after blurring the row images can be seen in figure 4. These results can be compared with figure 3 to see the effects of blurring. Noises are disappeared and the detected edges are way clearer. In 4-a smaller R kernel is used where as in 4-b 3x3 sized S kernel is used.



(a) Result of the R edge filter on blurred C1 image



(b) Result of the S edge filter on blurred C1 image

Fig. 4.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] Image Filtering. (n.d.). Retrieved November 06, 2018, from http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html

[3] Kernel (image processing). (2018, July 12). Retrieved November 6, 2018, from https://en.wikipedia.org/wiki/Kernel_(image_processing)

[4] R. Wang. Histogram Equalization. (2016, September 29). Retrieved November 7, 2018, from http://fourier.eng.hmc.edu/e161/lectures/contrast_transform/node2.html

[5] R. Wang. Histogram Specification. (2016, September 29). Retrieved November 7, 2018, from http://fourier.eng.hmc.edu/e161/lectures/contrast_transform/node3.html