CENG466 - Take Home Exam I

Berk Güler
2310092

Onur Demir
2309870

I. INTRODUCTION

This document is prepared for CENG466 - Take Home Exam I. This report includes the solutions of the given problems, and also explains our methodology and includes the analysis off the results and our comments on them.

II. QUESTIONS

In this homework there are 4 questions given. First question is histogram processing. Matching the histogram of a given gray-scale image to mixture of Gaussians is asked in this question. Second one is Convolution. In this part we are asked to implement a convolution function. Third question is edge detection. It is asked to implement an edge detecting algorithm of our choosing. Fourth and the last question is noise reduction. This part is different than the previous ones, since in this question we worked on specific images. Given two images we detected the noise in images and denoised them by using filters.

A. Question 1 - Histogram Processing

Color histogram of of an image is defined as,
$h(r_k) = n_k$
Where n_k is the number o pixels with value r_k

An alternative technique to histogram equalization, is a technique, called histogram specification. This technique can handle some of the problems that histogram equalization can not solve, if a priori information about the histogram of the desired image is available. Then, we may find a transformation, T, which maps the color palette of the original image to that of the desired palette, such that the histogram of the original image is transformed to the histogram of the desired image

In this question we simply created an array size of 256 with 0's first, in order to hold the count of pixel values. Then we checked pixel values of one channel of source image, increment the value of that pixel by 3, since it is an gray-scale image, R-G-B values are the same we don't need to check every channel of the image. Then, as it is asked in the question, we plot the original image histogram and save it to output path.

Secondly, we created sample data by using rowSize * columnSize * 3 random samples as it is mentioned in the homework pdf. We created gaussian histogram by using sample data and the given mean and std values by input. Then, saved the created histogram to the output path.

Thirdly, we found the cdf values of these histogram and used them to find histogram matching. We implemented a look up table function which returns the index values from the gaussian histogram. Then, by using this look up table we created a new np.array with shape(rowSize, colSize, 3), changed the pixel values accordingly. After that, we saved this new image to output path. And finally, we created the histogram of the matched image by using the same function that we used in the original image histogram.

B. Question 2 - Convolution

In image processing, a filter is an algorithm, which is developed to remove the unwanted features in an image, according to a predefined goal. The filtered image consists of desired characteristics of the image. There is a wide range of methods for filtering an image for blurring, sharpening, detecting the edges and lines etc. One of the basic mathematical tool in the spatial domain is 2-dimensional convolution operation.

Given a 2-dimensional discrete function, $f(x, y)$ representing a gray-scale image and a two dimensional filter $w(x, y)$, convolution is defined as,
$g(x, y) = f(x, y) * w(x, y) = \sum_s \sum_t w(x - s, y - t) f(s, t)$
whereas correlation is defined as,
$g(x, y) = f(x, y) * w(x, y) = \sum_s \sum_t w(x + s, y + t) f(s, t)$
In most practical applications the filter, w(x,y) is symmetric, with respect to the origin pixel, called the seed pixel. In this case, flipping the window does not make any change, thus, convolution and correlation operations give the same result.

In practice, an image is represented by an NxM array and a filter has very small size compared to the image size, e.g., 3x3, 5x5 or 7x7 or slightly bigger. Therefore, in our implementation, we decided to leave the edge pixels as it is, instead of zero padding.

In our implementation, we first read the image by using *cv.imread(input_img_path)*. Then we get the row size and column size of the image by using *image.shape[0], image.shape[1]*. Then, in order to find seed pixel we calculated the filterSize // 2 and called it delta. We assumed that filter size is NxN in here. First seed of our image was shifted from the far left corner to the right and down by delta size. We applied this shifting operation according to edge of the image for seed selection in every border of image, since we decided to leave the edge pixels as it is. First, we created a new image with 0's with the same size of our image. Then assigned the convoluted pixel values to it after doing necessary convolution operations. Then we returned the convoluted new image.

When we tried our convolution function with [[1,1,1],[1,1,1],[1,1,1]] filter, and we saw that output image is blurred version of the input image as expected.

C. Question 3 - Edge Detection

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Common edge detection algorithms include Sobel, Canny, Prewitt, Roberts, and fuzzy logic methods.

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in image processing, computer vision, and machine vision. Common edge detection algorithms include Sobel, Canny, Prewitt, Roberts, and fuzzy logic methods. In our very first design, we first tried the Sobel edge detection algorithm. However, it did not match our expectations since the edges were not so precise. As a result, we decided to look at other options. We first taught that the Canny edge detection algorithm is the most appropriate of the algorithms. Then we decided to use the Prewitt edge detection algorithm instead of the Sobel edge detection algorithm. We smoothed the image by a Gaussian filter, which kernel size is five and standard deviation is1.4, in order to remove the noise embedded in the image. Then extracted the edge map of the image using Prewitt filters, namely:

$$W_h = [[1,1,1],[0,0,0],[-1,-1,-1]] \tag{1}$$

$$W_v = [[-1,0,1],[-1,0,1],[-1,0,1]] \tag{2}$$

$$W_ld = [[0,-1,-1],[1,0,-1],[1,1,0]] \tag{3}$$

$$W_rd = [[0,1,1],[-1,0,1],[-1,-1,0]] \tag{4}$$

These filters are resulted in a four different output image. We added them all and seen the result. Instead of using Canny algorithm design in further steps we simply choose higher and lower threshold values and assigned them to the black and white pixel values. In our solution we used cv2.filter2D function to convolve since out own convolution function has a pure optimization. In order to get the output image fast we decided to use cv.filter2D function when doing convolution operation. As a result we have seen that our result is closer to the built-in Canny function of the cv2. The difficult part in our implementation was choosing the upper and lower threshold values since they are an important part of our output.

D. Question 4 - Noise Reduction

In this part we tried some filters to denoise the given noisy images. We chose the filters from our lecture notes and from the internet. After trying different filters we decided to use Gaussian Blur for enhance_3 and Median filter for enhance_4.

- Enhance - 3
  In the first part of the determination of the denoise filter, we saw that the Gaussian filter is the most appropriate one between the filters we used. For the input 3.png from our lecture notes and the internet, we started with a median filter which kernel size is 5. However, we had seen that it is more appropriate for salt and pepper type noise. After this step, we decided to use a local kernel which kernel size is 5. From both attempts, we did not get enough blurriness and denoising. Finally, we decided to apply a Gaussian filter which kernel size is 5, and the standard deviation value is 1.4. Since our image had some Gaussian noise, this one was the appropriate filter in our case. OpenCV provides cv2.GaussianBlur() function to apply Gaussian Smoothing on the input source image. We used this function for the sake of simplicity.
- Enhance - 4
  For enhance_4 we get the most clear output when we use the median filter with 5x5 size. Median filter is the best solution to denoise the salt and pepper noise from the images. Since the noise looks like salt and pepper in the 4. image, we decided to use median filter. We tried median filter with 3x3 and 5x5 size. Although using 5x5 size gives more clear output, it also make the image more blurry. Since main purpose is denoising the image, we used median filter with size 5x5.
  We used medianBlur function from cv2 library. This function takes an image which will be denoised and filter size as input. It defines a window of size NxN with the given size input. It slides the windows over the image and at each move computes the median value of the window. Then, assigns the pixel brightness value at the center of the window(seed) as median. Let's assume size is given as 3 and the selected window is,

  W = [[101,69,0], [56,255,87], [123,96,157]].

  Function takes this window and sort it in ascending order.

  W1 = [0,56,69,87,96,101,123,157,255]

  Then assign the pixel brightness value at the center which is 96 in 255's place.

REFERENCES

[1] Rafael C. Gonzalez and Richard E. Woods. 2006. Digital Image Processing (3rd Edition). Prentice-Hall, Inc., USA.

[2] Sahir, S. (2019, January 27). Canny edge detection step by step in python computer vision. Medium. Retrieved November 29, 2021, from https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.

[3] Fard, A. P. (2021, November 10). Histogram matching. Medium. Retrieved November 29, 2021, from https://towardsdatascience.com/histogram-matching-ee3a67b4cbc1.