CENG 466 Fundamentals of Image Processing Take-Home Exam 1

Mustafa Sezgin
Computer Engineering
e2380863@ceng.metu.edu.tr

I. Affine Transformation

The first question is about affine transformation, where we are supposed to implement a rotation function, named `rotate_image`, that rotates given images by given degree and uses the specified interpolation type. The function takes three arguments, which are the image to rotate, degrees of rotation, and the interpolation type.

For this purpose, I decided to use `skimage` library because it provides the required image transformation functions. `skimage.transform.rotate()` function rotates an image by a given degree and interpolates the rotated image using spline interpolation. It has a parameter, `order`, which specifies the order of spline interpolation. Depending on the value of `order`, the function performs different types of interpolation.

- `order = 0` : nearest neighbor interpolation
- `order = 1` : bi-linear interpolation
- `order = 3` : bi-cubic interpolation

In order to rotate the image by θ degrees, we need to use a rotation matrix:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

A pixel p' on the rotated image will get the value of the pixel p on the original image. Therefore, the relation between the coordinates of two pixels is as follows:

$$p' = R_\theta p \tag{2}$$

This is equivalent to the following:

$$p = R_{-\theta} p' \tag{3}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

We need to use (3) to find the coordinates of the pixel p for a given p'. By this formula will calculate the coordinates with respect to the origin, which is at the top left corner.

In order to rotate the image with respect to the center, we need to translate the origin to the center. Or, equivalently, we can translate the image to the left by half its width and to the up by half its height such that its origin is at its center. After that, we will rotate it and translate it back. The translation matrix we will use is as follows:

$$T_{\frac{n}{2},\frac{m}{2}} = \begin{bmatrix} 1 & 0 & \frac{n}{2} \\ 0 & 1 & \frac{m}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The relation between p and p' is now

$$p = T_{\frac{n}{2},\frac{m}{2}} R_{-\theta} T_{-\frac{n}{2},-\frac{m}{2}} p' \tag{5}$$

After getting the coordinates of p, we need to interpolate it with the neighbor pixels to calculate the color of p'.

Bi-linear interpolation calculates the value of a pixel by the given formula:

$$f(x,y) = a_{00} + a_{10}x + a_{01}y + a_{11}xy \tag{6}$$

The coefficients are as follows:

$$\begin{aligned} a_{00} &= f(0,0) \\ a_{10} &= f(1,0) - f(0,0) \\ a_{01} &= f(0,1) - f(0,0) \\ a_{11} &= f(0,0) + f(1,1) - f(0,1) - f(1,0) \end{aligned} \tag{7}$$

Bi-cubic interpolation calculates the value of a pixel by the given formula:

$$\sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} x^i y^j \tag{8}$$

The coefficients can be found by using the 16 equations $f(x,y)$, $f_x(x,y)$, $f_y(x,y)$, and $f_{xy}(x,y)$ when x and y are 0 and 1.

II. Histogram Equalization

The second question is about histogram equalization, where we are supposed to implement two functions, named `histogram_equalization` and `extract_save_histogram`. The first function does equalization and produces the enhanced image, and the second one extracts the histogram. For this purpose, I decided to use the `skimage` and `matplotlib` libraries.

`skimage.exposure.equalize_hist()` function takes only the image as argument, applies histogram equalization algorithm on that image, and returns the enhanced image. `plt.hist()` function takes a one-dimensional array of the pixel values of the image as an argument and displays the histogram of it. The number of bins can also be defined, and it is 20 in my case. The

histogram of the original image is shown in Fig. 1, which shows us that the pixels are not equally distributed to all gray values.

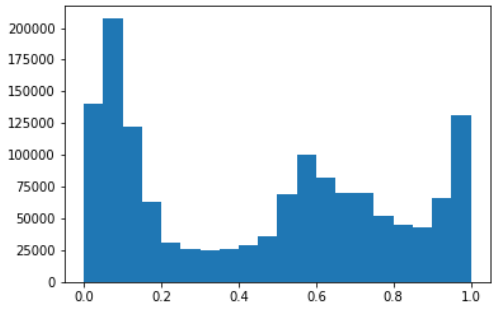

Fig. 1. Histogram of the original image "b1.png"

First, we need to extract the cumulative distribution of this histogram. For all gray values r_k we sum the number of all pixel with values less than or equal to that value.

$$h_c(r_k) = \sum_{r_i=0}^{r_k} h(r_i) \tag{9}$$

The cumulative histogram of the original image is shown in Fig. 2.

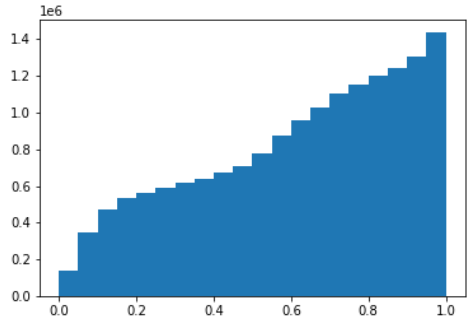

Fig. 2. Cumulative histogram of the original image "b1.png"

Second, we define a function T that maps r_k values to s_k values such that the number of pixels with each gray value s_k is as equal as possible:

$$s_k = T(r_k) = \frac{h_c(r_k)}{NM}(L-1) \tag{10}$$

N and M are the width and height of the image respectively, and L is the number of gray value levels, typically 256 in an 8-bit image. All pixels are then replaced by the new gray values s_k corresponding to their current gray value r_k according to the mapping T. The histogram of the enhanced image is shown in Fig. 3.

`adaptive_histogram_equalization` function uses `skimage.exposure.equalize_adapthist()` function, which takes only the image as argument. Adaptive

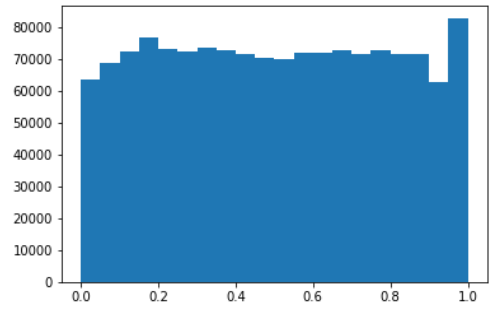

Fig. 3. Histogram of the enhanced image

histogram equalization algorithm applies the same calculations as the conventional histogram equalization; however, while calculating the new value of a pixel, it considers only the pixels in the neighborhood of the subject pixel instead of the whole image. The histogram of the adaptively enhanced image is shown in Fig. 4.

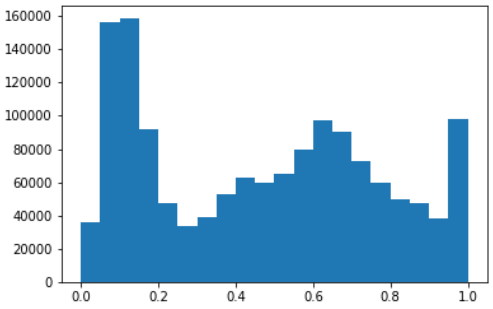

Fig. 4. Histogram of the adaptively enhanced image