PROGRAM DOCUMENTATION

Author: Dogu Gerger (dgerger@uoguelph.ca)

CIS*4720 – Image Processing and Vision

Winter 2023

Table of Contents

Section 1: User's Manual

Section 1.1: What the program does.

This is program is a generic image processing and manipulation toolbox, that can apply various different operations on an image. The program will take one image as an input, give the user a list of operations in the toolbox, and according to user's selection, it will apply the given operation to the image and save the new version of the image as separate file on disk.

<u>Available operations in the toolbox:</u>

- Crop an image: The program can crop a part of the image on the given coordinates with the given dimensions.
- Flip an image: The program can flip and image horizontally and/or vertically.
- Scale an image: The program can scale an image according to the given dimension.
- Rotate an image: The program can rotate an image by a given angle.
- Gray-level mappings: The program can apply linear and/or power-level gray level mappings to an image.
- Histogram calculations: The program can calculate the histogram of an image and can apply histogram normalization.
- Convolution: The program can calculate and apply a convolution kernel of a given size to the image.
- Non-linear filtering: The program can apply min, max and median filtering on an image.

Section 1.2: How to run the program.

In order to run the program please navigate to the main directory where the files run.sh and main.py are located. On Linux or Mac go to the Terminal and type ./run.sh to start the program.

Section 1.3: How to use the program.

After running the program, the program will ask for the user input, asking to provide the path of the image you would like to apply the operations to. After providing a valid path and pressing enter the program will go to the menu which will have all 11 options available each corresponding to a different feature in the program detailed in section 1.1 and additionally have the option to display details on the image, if pressed it will display details such as image size, if the image is colored or grayscale, and the image file type. 10th option is to exit the program and the 11th option is to change the image (provide new image path). User can select any option as displayed on the menu, each feature then will ask the user to provide additional details if applicable, for example if the user has selected to flip the image option, the program will ask if the user wants to flip the image horizontally or vertically. After entering the required details, the program will take the time to process the image, then ask the user to provide a file name to save the processed image as. User will enter the name and click enter, which will save the new image that the operations has been applied to, on disk. The program will return to the menu, which the user can either continue with the same image selecting any feature again or provide a new file name to operate on a different image or exit the program. The program will continue to run until the user selects exit on the menu displayed.

Section 2: Technical Discussion

Section 2.1: How the toolbox operates.

The toolbox starts as taking the path to an image saved on disk. It will then read that image onto a 2D matrix. Rows representing the height of the image and columns representing the width of the image. Every element in the matrix will represent one pixel in the image. And the value of each element in the matrix represents the gray level value of the corresponding pixel.

Section 2.2: Technical Discussion on different functionalities.

Section 2.2.1: Common algorithms used in implementation.

We have a few common algorithms that we use in other different operations we apply on images explained in Sections 2.2.2 to 2.2.9. These algorithms include grayscale_conversion and bilinear interpolation.

In grayscale_conversion, we convert an RGB image to a grayscale image because some of our image operations strictly operate on grayscale images and not on RGB images. In this function we take an input image as a matrix explained at the beginning of this section, which 3 color channels Red, Green, and Blue. We use the coefficients 0.2989, 0.5870 and 0.1140 for each color channel respectively. These coefficients correspond to luminosity values of each color channel (source: https://en.wikipedia.org/wiki/HSL_and_HSV). We use indexing to extract the Red, Green and Blue channels on our matrix representing our image. We then use element wise multiplication and addition of the color channel arrays with the corresponding coefficients to calculate the weighted average of the colors. Mathematically our equation can be defined as follows: $A(x, y) = 0.2989 * R(x, y) + 0.5870 * G(x, y) + 0.1140 * B(x, y)$, where $A(x, y)$ is the grayscale pixel value at location (x, y), and $R(x, y)$, $G(x, y)$, and $B(x, y)$ are the red, green, and blue pixel values at location (x, y) in the input RGB image.

Bilinear interpolation is a technique we use to estimate the value of a function at point (i,j) based on the values of the function at neighboring points. For this function we take 3 arguments, the first being the i coordinate, the second being the j coordinate, representing coordinates on our x and y axis, and our third argument is our input image which is stored in a matrix. The first thing we do in the function is to use the minimum function and division to find the index of the nearest four pixel to our point (i,j). Our function then computes the estimated value of the pixel using the weighted average of the values of our four nearest pixels. Mathematically we can define this as: r1 = (x2 - x) * A[y1, x1] + (x - x1) * A[y1, x2], r2 = (x2 - x) * A[y2, x1] + (x - x1) * A[y2, x2], pixel_value = (y2 - y) * r1 + (y - y1) * r2, where (x1, y1) and (x2, y2) are the indices of the nearest four pixels, A is the input image, and r1 and r2 are the weighted averages of the pixel values at the top and bottom rows of the four nearest pixels, respectively.

Section 2.2.2: Technical Discussion: Crop an image.

In the program as explained earlier we store the image as a 2D matrix. If we define this matrix as A with size mxn, and the cropped image as another matrix representing another image, let's call it B with size pxq. The process of how our program creates the output image as B = A(i:i+p-1,j:j+q-1) where row i and column j is the coordinates we are cropping from, which extends p rows and q columns.

Section 2.2.3: Technical Discussion: Flip an image.

In this function we are flipping an image, horizontally or vertically according to user input, again representing both the original image and the flipped image as a 2D matrix, in order to achieve this for horizontal flipping we use a function that returns the columns of the input matrix reversed. Mathematically, the flipping process can be represent as A[i,j] = B[I, n-1-j] where A is the matrix of our original image and the B is the matrix of our processed and flipped image. If the user input is vertical instead of horizontal, this time we reverse the rows of the matrix, which mathematically can be expressed A[i,j]=B[m-1-I,j]. In both cases assuming A is of size mxn.

Section 2.2.4: Technical Discussion: Scale an image.

In this function, we scale the image to a new size represented by output_width and output_height variables which define the size of the new image. The scaling process involves mapping the pixels in the input image to new coordinates in the output image and using the process of bilinear interpolation explained in Section 2.2.1 to determine the color of the pixels in the output image. These ratios represent the relationship between the size of the input and output image. We then map each coordinate looping over each pixel on our original image, multiplying the scaling ratio with the coordinate. We then send each pixel to our bilinear interpolation function to assign each of them colors finishing the scaling process.

Section 2.2.5: Technical Discussion: Rotate an image.

In this part of the program, we rotate an image by a specified angle in degrees. This process involves computing the new location of each pixel in the rotated image and copying the color of the original pixel to the corresponding location in the rotated image.

We first start with converting the rotation angles to radians with a defined function. And then calculate the sine and cosine of the angle and finds the new coordinates with the equation.

new_x = (x - center_x) * cos_a - (y - center_y) * sin_a + center_x

new_y = (x - center_x) * sin_a + (y - center_y) * cos_a + center_y

where center_x and center_y are the coordinates of the center point of the image. Then we copy over the gray level for each old coordinate with each new coordinate. We do this for every red, green, blue channel if the user wants to perform this for colored images.

Section 2.2.6: Technical Discussion: Gray-level mappings.

For this part we have 2 types of gray-level mappings that can be applied on an image, the first one is the linear gray-level mapping and the second one is the power-level gray level mapping.

If the user input is an RGB image, we convert the image to grayscale format using the conversion function defined in 2.2.1. We then calculate the new gray level mappings for each pixel with the formula a * A + b. Where A is the 2D matrix holding our original image and a and b are our scaling factors. We use a clipping function to clip to 0 if it falls below 0 and clip to 255 if it falls above 255, since that is our valid defined range for gray-level values.

For power-level mapping, we first normalize the gray-level value of each pixel by diving every value of our matrix with 255, if we name the normalized matrix B. We then define our output matrix C, where calculate each value by c * B^g, where c and gamma are our scaling factors. We then multiply by 255 again to get it to original form from normalized for and clip everything to stay in the range of 0 and 255.

Section 2.2.7: Technical Discussion: Histogram calculations.

For histogram calculations we have two functions, one to calculate the histogram for any given image, and one to apply normalization histogram.

In this function we create a histogram array with 256 elements, the first element representing the value 0 and the last element representing the value 255. We then traverse over every pixel value in our input image and increase the value of the corresponding index in our array by one to count the number of total gray-levels of that value for each value. We then return the histogram array for further processing.

For the normalization part for our histogram. We adjust the intensity values of an image so that they are distributed more evenly across the entire range of possible values. In the first step we get the histogram of our image using the previously defined function in this subsection. Then we use a function to calculate the CDF (Cumulative Distribution Function) of our histogram. We then use this CDF to compute and equalization mapping that maps each intensity value in the input image to a new value that is more evenly distributed across the range of possible values.

If call equalization mapping C, the equation is as follows C = (cdf – cdf_min) * 255 / (image_size – cdf_min), where cdf_min is the minimum non-zero value in the CDF, image_size is the total number of pixels in our image.

And if we call our equalized image B we then map our original image using the C mapping to image B.

Section 2.2.9: Technical Discussion: Convolution

In this part we apply a given convolution kernel to our image. This is the process of applying a sliding window operation over our image, where the values in our window are multiplied by a kernel and then summed to produce the new gray-level or RGB value.

We first start by determining the dimensions of our input image, then apply zero-padding required to apply the given kernel. We divide kernel width and height by 2 and get the nearest integer to find out how many zeroes we need to pad the image with before applying to kernel. We then add the zeroes to surround our original 2D matrix representing our input image. We then create an output image B who is also a 2D array same size as our original input image. We initialize this matrix with all zeros first. Then we traverse the rows and columns of our matrix we define for each pixel in our output image sum (padded_image[i:i+kernel_height, j:j+kernel_width] * kernel). The resulting value is assigned to the corresponding pixel in the output image array.

Section 2.2.10: Technical Discussion: Non-linear Filtering

In this part of our program, we apply non-linear filtering to our image. Each min, max and median filtering uses a similar method to apply the filter.

This function will ask for user input on if they would like to do minimum, or maximum or median filtering. Non-linear filtering refers to image filtering techniques that modify pixel values based on the values of neighboring pixels but do not involve a linear combination of pixel values. We first determine the dimensions of the input image, we then calculate the amount of padding required to apply the filter, in a similar way we have done in convolution explained in Section 2.2.0. We then initialize an output image represented by a 2D matrix, where each element is a zero initially. We then calculate the maximum, minimum or median value of the pixels within the sub-region that we are applying the filter in, according to user input. After we apply the filter in a similar way a compute the value, we assign it to the corresponding pixel in our output image.

Section 3: Discussion of Results and Future Work

Section 3.1: Assumptions.

One of our assumptions in our program is the type of the image file that is being passed. Our program supports the JPEG images who are images with the extension .jpg, .jpeg, PNG images with .png, BMP images with .bmp, TIFF images with .tif, .tiff. Program doesn't not support they image types of GIFS with .gif extension, SVG images with .svg extension, or RAW, WebP or .psd and .cdr images.

One other assumption that is made with our program is that the user who is running the program has "write" and "execute" access permissions on the operating system, as our program generates a new output file to be written on disk. And also there is enough space on the disk to hold the output image.

Section 3.2: Limitations.

Some of the limitations of our program is the failure to be able to operate on .gif and .svg images which are common image formats. One other limitation is that of speed, since some of the operations that is in the toolbox works slower that ideal. These are detailed on Section 3.3 of the documentation. One other limitation would be the color assignment accuracy on operations that use interpolation. As explain in Section 2, we use Bilinear Interpolation for assigning color/grayscale values to newly mapped pixels, even though it generally works well, for some large and detailed images with a high color variety an interpolation technique such as bicubic interpolation could have worked better, unfortunately our program does not support bicubic interpolation.

Section 3.3: Testing of the Toolbox.

Section 3.3.1: Testing methods and data used.

For testing, I have used multiple different images for each of the operations that are detailed in section 3.3.2 to 3.3.9. One of the images that was used in testing is the following:

Source: https://www.arl.org/wp-content/uploads/2022/01/2022.01.03-hand-holding-lightbulb-at-sunset-by-diego-ph-unsplash-600x450-1.jpg

Section 3.3.2: Testing: Crop an image.

When testing the feature of cropping an image, I have fed the given image above to the program, first providing the left most pixel as the part we are cropping from. I have done this with multiple dimensions and checked the output image to check if the cropping was done accurately. I have then tested with different points in the image and different dimensions, and also tested with some invalid values outside of the range of the image. Overall, the results were accurate, and operation was done quite fast on the image satisfying expectations. This testing worked accurately and equally as fast on both colored and grayscale images.

Section 3.3.3: Testing: Flip and image.

For this feature I have tested with both a colored RGB image and a grayscale image. I ran the test first providing “horizontal” user input and then “vertical”, both of the test results were accurate, and the operation was performed very fast.

Section 3.3.4: Testing: Scale an image.

I have tested this feature again with a colored and grayscale image, I have provided different scales as input for both shrinking and enlarging of the image. The operation was applied not as fast as the previous features however it still satisfied my expectations. The color assignments were accurate however they could’ve been improved if I had used bicubic interpolation instead.

Section 3.3.5: Testing: Rotate an image.

I have tested this feature again with different types of images. I have provided as user input multiple degrees to rotate the image with all the test results were accurate and the rotation was done relatively fast.

Section 3.3.6: Testing: Gray-level mappings.

I have tested this feature with an RGB image and grayscale image. As explained in Section 2, this function converts the RGB image to a grayscale image first. Then applies the gray-level mapping. I have tested both the options for gray-level mappings, linear and power-level. I tested the extreme values of the coefficients for both mappings to observe how the image is being reassigned colors. The results seemed to be accurate, however the speed of power-level mapping could be improved.

Section 3.3.7: Testing: Histogram calculations.

I again tested this feature with both RGB and grayscale image. The RGB was supposed to return 3 histogram one for each color, grayscale to return one histogram. I tested with small images and some images that do not contain a certain color value at all, I observed that histograms seemed to be accurate. Then when I tested the normalization function, it was also working fast and working as expected.

Section 3.3.9: Testing: Convolution

To test this, I applied the standard convolution kernel for blurring to both colored and grayscale images, the image seemed blurred on the output, even though the speed in which the operation is applied could be improved the results have satisfied my expectations.

Section 3.3.10: Testing: Non-linear Filtering

I have also tested this feature with multiple types of different images, min median and max all seemed to have been working fine with some basic values of windows. However, the operations have taken a long time, for some images even exceeded 1 minute. I found that the speech could be improved for this feature.

Section 3.4: Analysis of Results

Section 3.4.1: Performances.

We have observed a very high performance and accuracy in the operations of flipping, rotating, and cropping an image. Since there was no interpolation required, and we worked directly with manipulating the location of the elements of the matrix, the speed of our operations was on ideal levels.

We have also observed very accurate interpolation results with grayscale images and high speed on our histogram calculations for both grayscale and RBG images.

Section 3.4.2: Weaknesses.

One weakness we had was the speed on non-linear filtering operations especially on our tests with larger RGB images. Some of the tests have taken over a minute to produce an output image which is not an ideal number. We have also found some relatively poor interpolation results with images that have a high color variety and high detail, such as the image that is given an example in the beginning of this section.

Section 3.5: Future Work.

Overall, the program has worked as intended and satisfied our expectation. However, there are still multiple things which I am planning on adding to this project to improve and develop it further. One of the future additions is adding bicubic interpolation, since bicubic interpolation will generate a more accurately interpolated image when we transform the image in different operations, I believe this will be a major improvement that will contribute a lot when added. One other future work that can be added to this project is the addition of a GUI. Where you can see the changes the different operations make to image right in front of your eyes and select to apply different operations more easily. We can also think of adding features and operations such as image segmentation and image registration in the future.