EE610: Assignment 1 - Basic Image Editor

Advait Kumar - 18D070003
Electrical Engineering
IIT Bombay
Mumbai, India
advaitkumar@iitb.ac.in

***Abstract*—Image processing is an important sub-field of computer vision. It has wide applications in a plethora of subjects. As part of my first assignment, I have made a GUI using Python and its libraries, which consists of various tools and filters so as to study the effects of these on different types of images more conveniently. This document consists of the explanation of the code and algorithms (coded from scratch), used in making the image editor.**

***Index Terms*—Image Processing, Computer Vision, GUI, Python**

I. Introduction

Digital Image Processing [1], is one of the most important fields of Computer Vision. It basically refers to use the use of a computer to perform a sequence of operations on the input image. It is widely used because it has some advantages over its analog counterpart. Since the values are discretized, it helps in removing any noise or distortion during processing, which is often encountered in analog image processing. A larger range of algorithms can also be formulated and used for digital image processing.

Through this image editor we can do the following:

1) Study how different images behave with different algorithms.
2) Check out the role of the hyperparameters of an algorithm and an image's dependance on it.
3) Perform a cascade of various algorithms to try out a variety of effects such as edge detection, cleaning, sharpening, etc.

This report describes the various components of the GUI as well as its widgets. It also deals with the implementation and the logic behind the image processing algorithms which are used and the vectorized convolution operation defined for it. The main parts of the code are:

- GUI widgets
- Image Processing Algorithms
- Vectorized Convolution Operation

II. GUI Design

For the GUI design, I have used the Tkinter [2] library which supports Python. The GUI consists of a window which has two parts, namely, (a) A toolbar where the buttons are present for the user. (b) A canvas, where the images are displayed after the user applies the transformations.

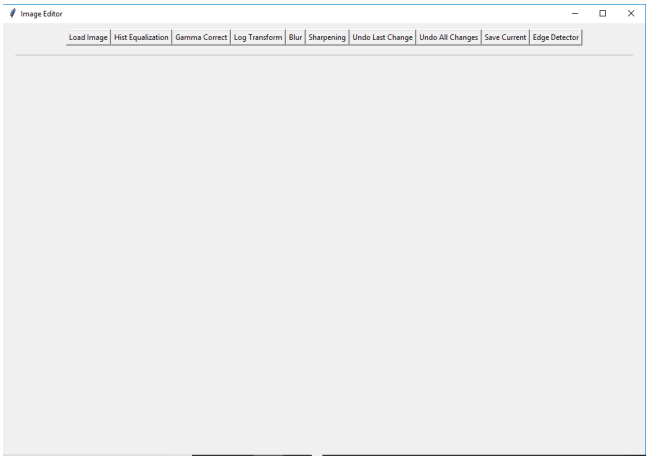

Fig. 1. The GUI appearance

The main class which contains the GUI and its widgets also contains a few more variables. It contains a list of images. Every time the user either loads an image, or applies some transformation on the image, that transformed/loaded image is appended to the list. This helps in the undo last button, where the most recent element is popped out from this list. In case of undo all button, the entire list is replaced by a list having just the first element of the original list. The class also contains variables for the vertical and horizontal sliders that are used by the users to give in their input for some hyperparameters.

The GUI has the following buttons for the user:

- **Load Image**: A dialog box is opened for the user, after which the user enters the image's absolute path. The image is then loaded onto the canvas from this location.

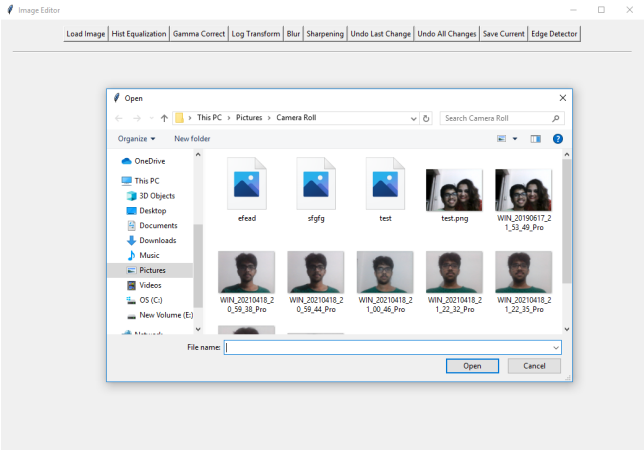

Fig. 2. The dialog box loaded for the user to give their absolute path

- **Save Current**: A dialog is opened which takes in the absolute path from the user. It then saves the current displayed image into that path on the local computer of the user. In case there is no image displayed, an error message is displayed.

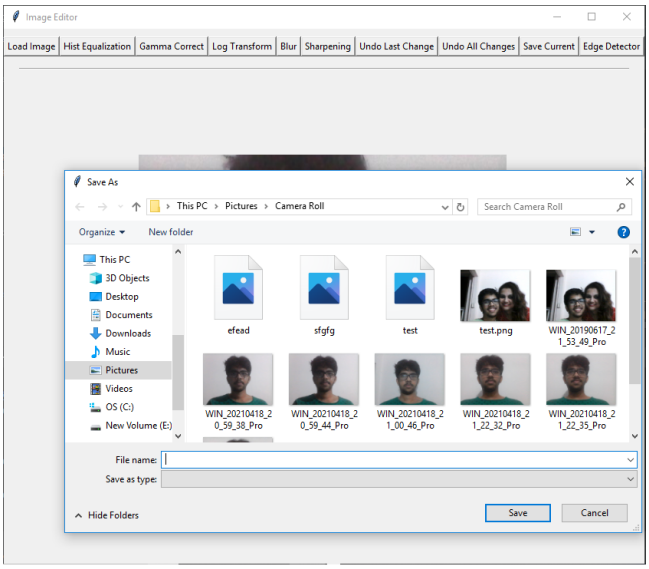

Fig. 3. The dialog box loaded for the user to give their absolute path for saving image

- **Undo Last Change**: This displays the previously displayed image (before the transformation was applied). In case there has been no transformation applied, it clears the canvas. In case the canvas is already cleared, it displays an error message.
- **Undo All Changes**: It reverts back to the very first image that was loaded onto the GUI screen by the user. In case the screen is already blank, it displays an error message.
- **Hist Equalization**: It applies histogram equalization on the image by first converting it into HSV, then applying the transformation on the 'V' channel. In case there is no image loaded onto the screen it gives an error message.
- **Log Transform**: It applies log transformation on the image by first converting it into HSV, then applying the transformation on the 'V' channel. In case there is no image loaded onto the screen it gives an error message.
- **Gamma Correct**: This first invokes a slider which is then used by the user to input their value of gamma. On clicking the 'set gamma' button, the gamma value is set and the gamma correcting algorithm is applied on the input image. The image is first converted to HSV type and then the correction is applied. In case there is no image loaded onto the screen it gives an error message.

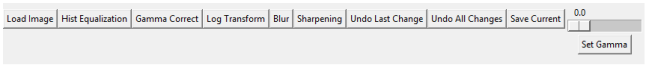

Fig. 4. The slider that is invoked for the gamma correct button.

- **Blur**: This first invokes a slider which is then used by the user to input their value of standard deviation for the gaussian kernel. On clicking the 'set STD of gaussian window' button, the std value is set and the gaussian blurring algorithm is applied on the input image. The image is first converted to HSV type and then the blurring is applied. In case there is no image loaded onto the screen it gives an error message.

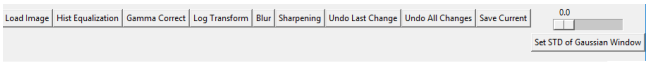

Fig. 5. The slider that is invoked for the blur button.

- **Sharpening**: This invokes two sliders, one for std of the gaussian kernel and the other for the value of the constant used for sharpening. It then performs unsharp masking on the input image as per the user input. In case there is no image loaded onto the screen it gives an error message.

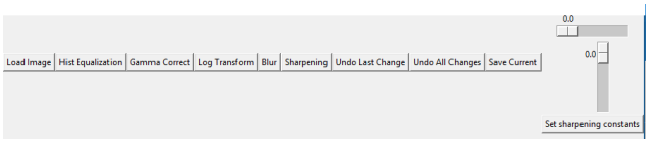

Fig. 6. The slider that is invoked for the sharpen button. Horizontal slider is for std and vertical is for constant value.

- **Edge Detector (Extra Feature)**: This invokes a slider which the user can use to input their threshold value. It then performs Sobel edge detection followed by hard thresholding on the input image. In case there is no image loaded onto the screen it gives an error message.

Fig. 7. The slider that is invoked for the edge detection button.

The error message which is displayed whenever a non logical button is clicked by the user is given below:

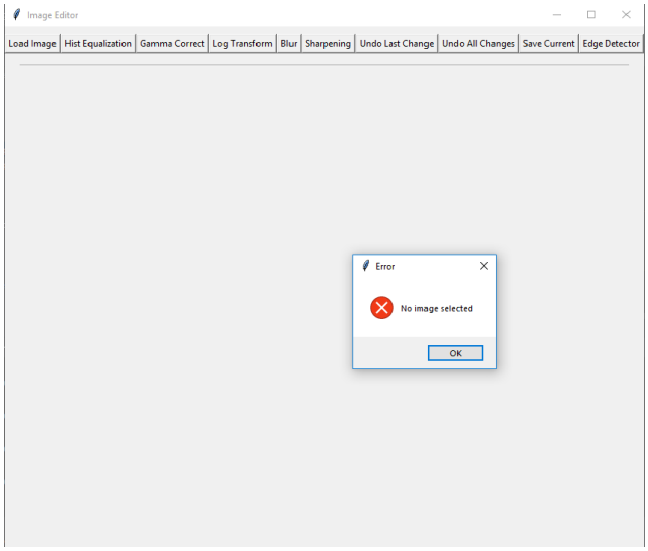

Fig. 8. Error message displayed

The entire code for these buttons and the error message is given at the end in the Appendix.

III. Image Processing Algorithms

This section describes the various image processing algorithms that were used in the GUI:

A. Vectorized Convolution

For most of the image processing transformations, we require a convolution betwwe the image and a kernel. Hence I created my own vectorized convolution function which first extracts windows equal to the size of the kernel from the padded image. Then lays down the windows as columns of a matrix, and then performs a matrix multiplication between the flattened kernel and each column finally reshaping it back to get the transformed image. Code:

```
h, w = kernel.shape  # Get the height and
    width of kernel
padded_img = np.pad(img, (((h-1)//2, (h
    -1)//2), ((h-1)//2, (h-1)//2)), mode='
    reflect')  # pad the image
# in such a way that dimension of output
    image equals dimension of input
sub_matrices = view_as_windows(padded_img
    , (h,w), 1)  # break the padded image
    into windows that are of the same
# size as the kernel with stride = 1
convoluted = np.einsum('ij,klij->kl',
    kernel,sub_matrices)  # This
    multiplies the windows with the kernel
    elementwise
# and adds the resultant elements of the
    matrix to give the output convolved
    image
return convoluted  # Return the
    convolved image
```

B. Log Transformation

Log transformation refers to the algorithm where we replace all the pixel intensity values by a value proportional to its logarithm to the base 2. Hence the mathematical formula is given by:

$$T(r) = c * \log_2(1 + s(r))$$

Where T(r) is the output intensity of pixel r, s(r) is its input intensity and 'c' is an arbitrary scaling constant. The 1 is added inside the log so that if the input intensity is 0, the output doesnt become unbounded.

The log transformation finds its applications in the field of image enhancement. This is due to the fact that it expands the range of pixels that have lower amplitudes and compresses the range of pixels having higher amplitudes. As a result it compresses the dynamic display of the image.

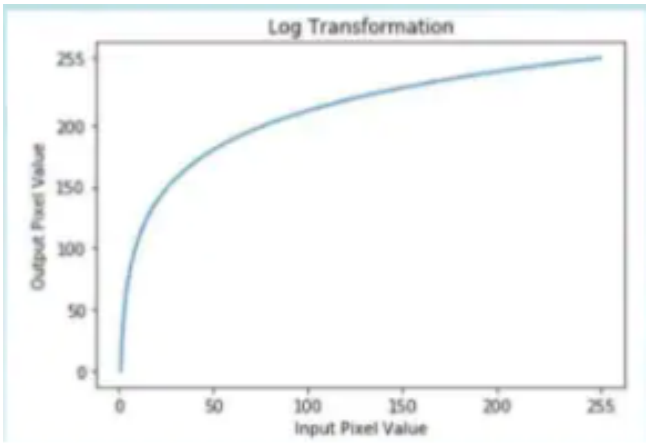

Fig. 9. The graph of output vs input intensities

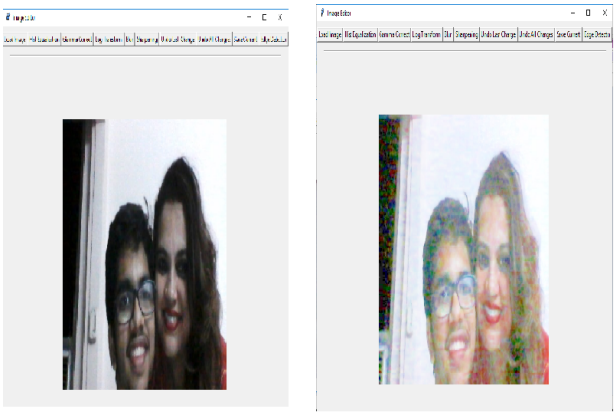

Fig. 10. (a) My selfie (b) The log transformed image

We can see from the above example that my original selfie had low contrast, however in the log transformed image, even the dark pixels have become a bit lighter. Hence it has stretched the contrast of my image.

C. Histogram Equalization

Histogram equalization [3] consists of an algorithm which adjusts the contrast of the image by using its histogram. Let the total number of pixels of an image be 'n' and let L be the number of different intensities that the pixel can have, then,

$$p(i) = \frac{n_i}{n}, 0 < i < L$$

is the probability of occurence of pixel of level 'i'. The cumulative distribution is defined as:

$$cdf(i) = \sum_{j=0}^{i} p(j)$$

Hence we define the transformation as:

$$y = T(x) = cdf(x)$$

This leads to a flattened histogram for the input image. It helps in increasing the global contrast of the image which can lead to discovery of more features for example in X-Rays.

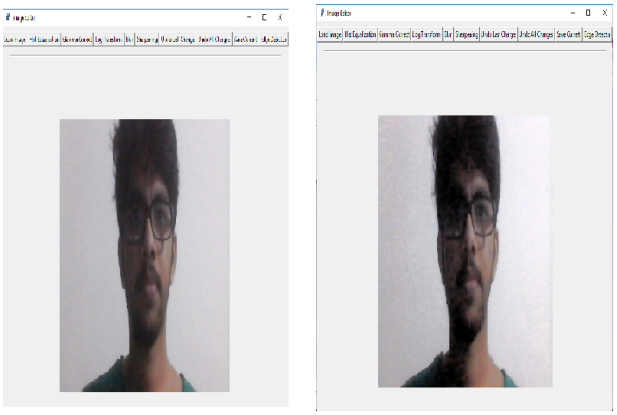

Fig. 11. Example of histogram equalization. (a) My Selfie (b) Histogram equalized image

Hence we can see that again my low contrast image was improved upon by performing histogram equalization.

D. Gamma Correction

This is an intensity transformation. Let s(r) be the input pixel intensity and T(r) be the output pixel intensity:

$$T(r) = c * s(r)^{1/\gamma}$$

Where c is used for normalization. This algorithm is especially useful when the input image needs to be encoded so that it becomes more pleasant for the human eye to observe (since it is noted that the human eye approxiamtely follows the power law function). When $\gamma < 1$ it is referred to as gamma compression. When $\gamma > 1$ it is called gamma expansion.

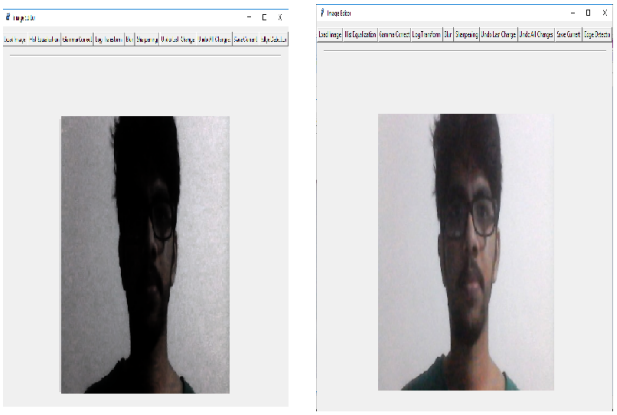

Fig. 12. (a) My original noisy selfie (b) Gamma corrected image

Hence we can see in this case that when my image was a bit unclear, on using gamma correction by adjusting the value of gamma, it became clear again.

E. Gaussian Blurring

Gaussian blurring is performed by convolving the image with a gaussian kernel. The gaussian kernel is defined as:

$$G(x, y) = \frac{e^{-(x^2+y^2)/2\sigma^2}}{\sqrt{2\pi\sigma^2}}$$

Where 'x' is the horizontal distance of a pixel from the centre of the kernel and 'y' is the vertical distance. σ is the standard deviation of the kernel.

It finds its use in many areas especially for blurring and smoothing as a pre processing step. The discrete image areas can be smoothened out by applying this algorithm. This will help maintain continuity in the image. It is also a powerful pre processor for edge detection, where it helps in removal of unwanted edges.

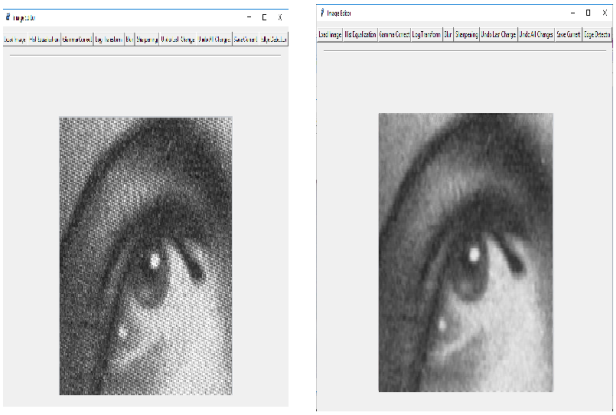

Fig. 13. (a) Image that has discontinuities (b) Gaussian Blurred image

Hence we can see from the above images that applying Gaussian blurring helped in smoothening out the artifacts which made for a much more pleasant representation. ((a) image taken from [4])

F. Unsharp Masking

This is just the inverse of blurring. Blurring is first used to create a 'mask' of the image. This is then subtracted from the original image, leading to a sharper image. If T(x) denotes blurring of image x, then the unsharp masking will do the following:

$$G(x) = x - c * T(x)$$

Where 'c' is an arbitrary constant for multiplication. I have used Gaussian blurring in this implementation as well as my blurring kernel. Unsharp masking is used as for sharpening an image. This could be potentially used in number plate recognition from CCTV footage where the number plates are usually blurred out.

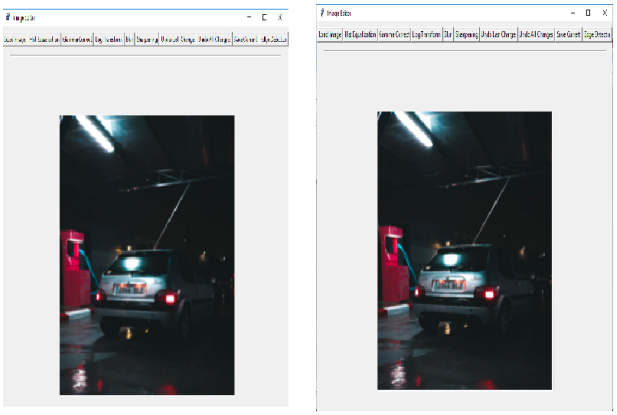

Fig. 14. (a) The blurred out number plate (b) Sharpened image

Hence we can see in the above example that unsharp masking was useful in removing much of the blurred content of the image. (Image (a) taken from [5])

G. Sobel Edge Detection

The extra feature that I implemented was the Sobel edge detector. Computer Scientists, Sobel and Friedman first proposed the Sobel filters in one of their talks. This is an edge detector which calculates the gradient of the image using the Sobel filters and returns the magnitudes which contain the edge information. This returned magnitude is then thresholded to give a clearer edge map. The sobel gradient kernels are defined as:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & 2 \end{bmatrix}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Where G_x is the derivative in the X direction and G_y gives the derivative in the Y direction. The magnitude matrix is then given by:

$$G = \sqrt{G_x^2 + G_y^2}$$

The magnitude matrix is then hard thresholded. Let the threshold be some T_o. Hence the thresholding function is given by:

$$G(x) = \begin{cases} 0 & x \leq T_o \\ 255 & x > T_o \end{cases}$$

Hence the Sobel edge detection is used for detecting edges in a wide range of applications.

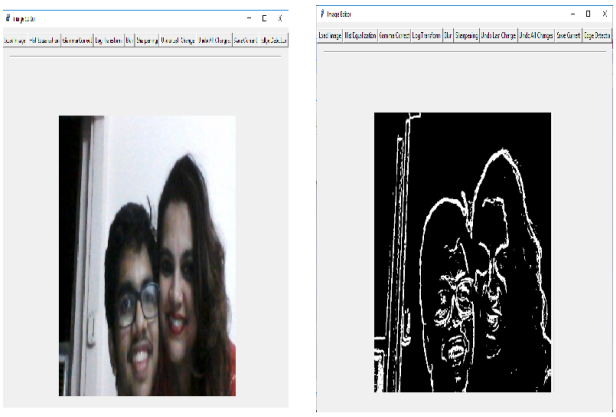

Fig. 15. (a) My selfie (b) The edges detected

We can see from the above picture, although the edges are a bit thick, the Sobel operators have successfully detected the edges in my picture.

H. Cascading of Transforms

As discussed earlier, in some cases of edge detection, we need to first smoothen out the image and then apply edge detection to it.

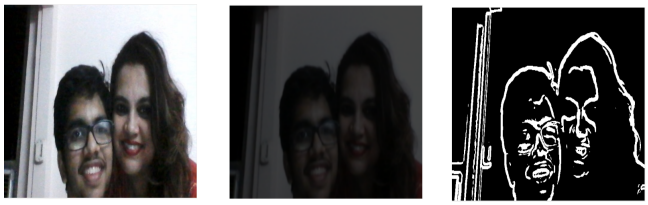

Fig. 16. (a) Original image (b) Gaussian Blurred (c) Sobel Edge Detector

We can see that as compared to previous case (Fig. 15), the edges have been detected better and are far less in number and smoother. This is the result of applying gaussian blurring before doing edge detection which helps in smoothening out the discontinuities and hence will remove extra edges.

IV. Conclusion and Discussion

Hence we can see from the above sections that by making a GUI we were able to study the various image transformation very easily and conveniently. Furthermore, we can extend this project in the following ways:

- Add more image processing transformations in the GUI
- Improve the GUI design to make it more user friendly and better looking
- Instead of matrix multiplication, use FFT for convolution (which will be faster for larger images)
- Instead of Sobel edge detector, use Canny Edge detector which is known to perform better than it generally but is a bit more tedious to implement
- Use optimized storage for undo last button. Currently there is a list that stores all the images till the curent instant. If the number of operations grow in size, the memory overload will grow which will cause the GUI to slow down. Hence there is need to optimize the storage for undo last button.

  The entire code for the project can be found at https://github.com/advaitkumar3107/EE610_GUI

References

[1] Wikipedia contributors, "Digital image processing — Wikipedia, the free encyclopedia," 2021, [Online; accessed 29-August-2021].

[2] Fredrik Lundh, "An introduction to tkinter," *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[3] Wikipedia contributors, "Histogram equalization — Wikipedia, the free encyclopedia," 2021, [Online; accessed 29-August-2021].

[4] Wikipedia contributors, "Gaussian blur — Wikipedia, the free encyclopedia," 2021, [Online; accessed 29-August-2021].

[5] Wikipedia contributors, "Unsharp masking — Wikipedia, the free encyclopedia," 2021, [Online; accessed 29-August-2021].