ImProME - Image Processing Made Easy

Submitted in partial fulfillment of the requirements of the course of
EE610 Image Processing, Electrical Engineering
Indian Institute of Technology Bombay
2021

Eeshaan Jain (19D070022)
Electrical Engineering
IIT Bombay

***Abstract*—The goal of this assignment was to build a graphical user interface (GUI) for an image editor which handles images in the HSI color model and works with the intensity channel. ImProME - Image Processing Made Easy is a GUI tool built with the help of Qt Designer and PyQt5 and is able to perform basic image operations such as rotation and flip, and in addition can perform manipulations such as histogram equalization, gamma correction, logarithmic transform, various types of controlled blurring, various types of controlled sharpening, color inversion, channel selection and conversion of the image to gray-scale. All the manipulations have been written from scratch efficiently using vectorization operations. Undo operations have been taken care of using the stack data structure. In addition to the above mentioned operations, we can either scrape the internet for an image, or generate a random colored image or load a random cat image from a local directory.**

***Index Terms*—Image Processing, GUI, Python, Qt, Intensity Transforms**

I. Introduction

The GUI tool `ImProME - Image Processing Made Easy` is developed for easy access to many of the image manipulations tasks present via a graphical user interface made in PyQt5 with the aid of Qt Designer. The assignment task can be broken down into three main parts:

A. Objectives

The assignment was to design a GUI tool in Python to perform manipulations such as intensity transforms on a given image.

B. Design Approach

A friendly and modular approach is followed in building the GUI. A standard application size is used. All related features are clubbed together for easier access. Majorly, a binary color scheme: Deep Cove + Java is used.

C. Features

The list of features which the application can do is:

- Load a new image
- Save the current image
- Flip the image horizontally, vertically and diagonally
- Rotate the image by 90° clockwise or anticlockwise
- Rotate the image by a specified amount of degrees
- Perform Histogram Equalization
- Perform Gamma Correction (Power law transform)
- Perform Logarithmic Transform
- Blur the image
- Sharpen the image
- Invert the color of the images
- Get a certain channel of the image
- Convert image to grayscale
- Fetch a random image
- Load a cat image from local folder

II. GUI Design

The graphical user interface is designed in PyQt5 which is a comprehensive set of Python bindings for Qt v5 which is written in C++. Qt designer was used as the tool to create the design of the GUI and generate basic boilerplate code to work upon.

Fig. 1: GUI Design

As seen in Figure 1, majority of the application follows a binary color scheme - Deep Cove + Java. A semi-modern design approach is kept in mind while creating the GUI by adding rounding to buttons, and adding different hover/click effects. A simple logo is made with a logo generator available online [link: https://howtostartanllc.com/logo-maker]. Specific features of the GUI are:

A. Main Window

A standard size of 1200×900 is followed to suit all devices while designing the GUI. A menu-bar is added with the File and Help menu-items. The File menu-item can be used to load or save an image. The Help menu-item opens up the documentation. Majority of the space of the main window is used by the image viewing area. Below that are some buttons for quick operations such as loading a new image or performing rotations. To the right are image manipulations, and two random features - loading a random image from the internet, or load a random cat image from the `Images/cats` directory. At the very bottom is the section containing details of the assignment. All buttons are QPushButtons and text/labels are QLabels.

B. Image Viewing Area

The QGraphicsView element is used with a fixed size of 951×661 to load the image. If the size of the image is smaller than this, then it is automatically centred leaving blank space around. If the size of the image is larger than this, then automatically a scroll-bar is added to view around the image (note that the size of the image view is fixed and doesn't change). When the application loads, a sample image is loaded located at `Images/sample.jpg` which is clicked by **Ji Guo (China)** available at https://fujifilm-x.com/en-us/products/cameras/gfx100s/sample-images/

C. General Features below Image View

The options to load a new image, save the current image, undo once and undo all (revert) are added to the left below the image viewing area. Options to flip the image vertically, flip the image horizontally, flip the image diagonally, rotate the image clockwise or anticlockwise by $90°$ and rotate the image by any arbitrary number of degrees (note that this causes the remaining part of the image to be filled with black) is added to the right below the image viewing area.

D. Image Manipulations

To the right of the image viewing area is the image manipulation area which encompasses the features asked in the assignment, along with options to invert the color of the image, get a specific (R/G/B) channel of the image and convert the image to gray-scale.

E. Random

The random section has two options - either load a random image from the directory `Images/random/` or load a random cat image from the directory `Images/cats/`

F. Undo operations

The undo operations are performed using the stack data structure. Each layer of the stack contains two items - the image itself and the path to where the image is saved. Originally the stack only contains the {Sample image, `Images/sample.jpg`}. Whenever an operation is performed, the image and path is added to the stack. If the image is loaded from somewhere, this is taken care of easily as the QFileDialog returns the path of the loaded file. An issue comes in whenever we perform an operation. At that time, the image is formatted in place and hence there is no path. To resolve this the `temp/` directory is created. Whenever a non local-load operation is performed, the image is stored in the directory with a random name of length 12 chosen from ASCII uppercase (26), ASCII lowercase (26) and digits (10). Though this can generate non-unique names, the collision probability will be very less.

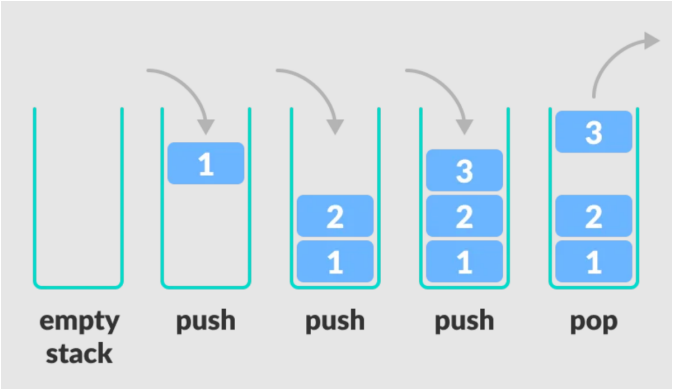


Fig. 2: Stack data structure. Pic credits: programiz

$$\mathbb{P}_{collision} = \left(\frac{1}{62}\right)^{12} \approx 3.1 \times 10^{-22} \tag{1}$$

In Python, lists can be used to mimic the stack data structure and that is done in the variable `undoStack`. Whenever an operation is done, the new image is loaded. Whenever the undo button is used, the last item from stack is popped off, and the image corresponding to the top of stack now is displayed. Hence we see that the undo operations have been performed correctly (though this is not space efficient and can be made better with hashing, but that is avoided for now), and performing revert/undo all is just a matter of popping everything of the stack.

G. Directory Structure

Instead of having all the code in one place, a sort of modular approach has been used to distribute the code for easy access and bug fixing. The file structure is as follows:

`ImageProcessingGUI/`: Main project folder

- `GUI/`: Contains the `main.ui` file generated for the GUI
- `Images/`: Contain sample and GUI images and 2 folders:
  - `cats/`: Contain sample cat images from pixabay
  - `random/` Contain random images from pixabay
- `Results/`: Stores the histograms generated by the histogram equalization operation
- `Scripts/`: Contains all the scripts made for the GUI
  - `basic_transforms.py`: Basic rotation and flip operations

- ◇ `convolution.py`: Convolution with filter
- ◇ `filters.py`: Gaussian, Box and Laplacian Filters
- ◇ `hist_eq.py`: Histogram equalization
- ◇ `image_conversion.py`: RGBA to HSV, HSV to RGBA and channel extraction
- ◇ `test.py`: For local testing of scripts
- ◇ `transforms.py`: Intensity transforms and inversion

◇ `temp/`: Stores the local results of undoStack
◇ `app.py`: Main file which contains all the code
◇ `gui.py`: Contains the python equivalent code of `main.ui`
◇ `README.md`: Github README file
◇ Other files include images, assignment statement and MIT License

H. Others

◇ All operations are verbose, in sense that the command line interface from which the `app.py` file is run shows the recent operation.
◇ At the start of every session, the `temp/` and `Results/` directory is cleared. Hence after each session, if forgotten to save, the results and intermediate stages are saved in these directories, although with arbitrary names.
◇ All images are loaded and worked with in the RGBA format where A is the alpha channel, which is not tampered with throughout.
◇ All buttons have a tooltip, hence hovering over them will tell what the button does.
◇ Each input is matched using regex, and if something invalid is given, the operation doesn't happen. Examples of inputs have been added in the dialog boxes.
◇ Libraries used are PyQt5, NumPy, Scipy, (Numba)

III. Image Processing Operations

A. RGB to HSL

When the image is loaded, it is in the RGB: [red, green, blue] format. We need to work with HSL: [hue, saturation, lightness] (especially the L layer) and thus need to convert it. Firstly, we normalize R, G, B values by dividing by 255.
Define

$$M = \max(R, G, B)$$
$$m = \min(R, G, B)$$
$$\delta = M - m$$

The mathematical definitions are:

◇ Hue (H)

$$H' = \begin{cases} \text{undefined}, & \text{if } \delta = 0 \\ \frac{G-B}{\delta} \bmod 6, & \text{if } M = R \\ \frac{B-R}{\delta} + 2, & \text{if } M = G \\ \frac{R-G}{\delta} + 4, & \text{if } M = B \end{cases} \tag{2}$$

We often specify it in degrees, with $H = H' \times 60^\circ$

◇ Lightness (L)

$$L = \frac{M+m}{2} \tag{3}$$

This gives $L \in [0, 1]$. We multiply it by 255 and round it off.

◇ Saturation (S)

$$S = \begin{cases} 0, & \text{if } L = 1 \text{ or } L = 0 \\ \frac{\delta}{1-|2L-1|}, & \text{otherwise} \end{cases} \tag{4}$$

This gives $S \in [0, 1]$. We multiply it by 100 and round it off.

B. HSL to RGB

Once we have done our image manipulation in HSL color, we need to convert it back to RGB to be able to display it. Firstly, we normalize S and L. Define

$$C = (1 - |2 \times L - 1|) \times S$$
$$X = C \times (1 - |H' \bmod 2 - 1|)$$
$$m = L - \frac{C}{2}$$

$$(R_1, G_1, B_1) = \begin{cases} (0,0,0) & \text{if } H' \text{ is undefined} \\ (C,X,0) & \text{if } 0 \le H' < 1 \\ (X,C,0) & \text{if } 1 \le H' < 2 \\ (0,C,X) & \text{if } 2 \le H' < 3 \\ (0,X,C) & \text{if } 3 \le H' < 4 \\ (X,0,C) & \text{if } 4 \le H' < 5 \\ (C,0,X) & \text{if } 5 \le H' < 6 \end{cases}$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m) \tag{5}$$

We finally multiply R, G, B with 255 and round them to get the result in $[0, 255]$.
Note: Both RGBtoHSL and HSLtoRGB functions have been optimized with the use of Numba for faster calculation: http://numba.pydata.org/. On the sample image (576×768), without numba, the execution time taken for converting all pixels from RGB to HSL is around 7.5 seconds, and with numba, the execution time taken reduced to 1.5 seconds, providing a $5\times$ speedup. But in spite of that, the inbuilt colorsys module after numpy vectorization proved to be faster, and hence was used.

C. Histogram Equalization

Histogram equalization is a method to process images in order to adjust the contrast of an image by modifying the intensity distribution of the histogram. The objective of this technique is to give a linear trend to the cumulative probability function associated to the image.
Say we have a grayscale image $\mathcal{I}$ of size $M \times N$ with intensity values $\in [0, 255]$. We have the frequency distribution of the intensities in $\mathcal{H}$. For every intensity i, it is mapped to

$$i' = \frac{255 \times \sum_{k=0}^{i} \mathcal{H}_k}{MN} \tag{6}$$

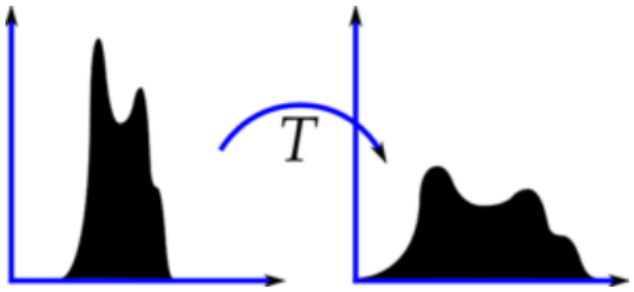


Fig. 3: Histogram Equalization. Pic credits: cvexplained

If $i' \notin \mathbb{N}$ then we round the result off. Thus we are essentially converting a discrete distribution of intensities to discrete distribution of probabilities.
The old and new histograms after histogram equalization are stored in the `Results/` directory. A similar naming scheme as to *F. Undo operations* has been employed with the naming scheme of the plot as

`histogram_equalization_result<random>`

where <random> denotes the random string of length 12.
Uses:

1) Achieve better quality black and white images for medical uses such as MRIs.
2) Can be used to display noise in the image in some cases.

Early citations

1) https://www.nxp.com/docs/en/application-note/AN4318.pdf
2) https://www.sciencedirect.com/science/article/abs/pii/S0734189X8780186X

D. Gamma Correction

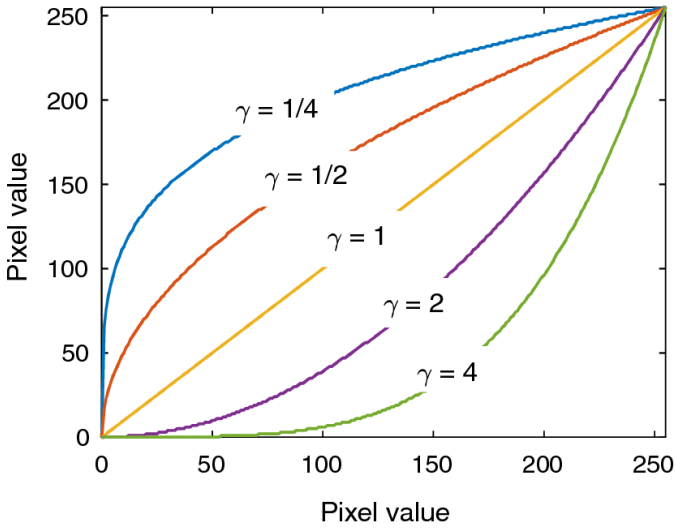


Fig. 4: V_{out} v/s V_{in} for different values of γ. Pic credits: [6]

Gamma correction (also called power-law transformation) is a non-linear operation used to encode or decode luminance in still images. It is implemented as

$$V_{out} = AV_{in}^{\gamma} \tag{7}$$

where A is the normalization constant. Gamma compression usually has $\gamma > 1$ and gamma expansion usually has $\gamma < 1$. In terms of perception, higher gamma makes the lighter regions darker while lower gamma makes darker regions lighter.
Since it is a power-law transform, we can find γ using the slope log-plot of the input and output intensities, i.e

$$\gamma = \frac{d(\log V_{out})}{d(\log V_{in})} \tag{8}$$

Uses:

1) Macintosh had partial gamma correction embedded in their hardware.

Early citations

1) https://poynton.ca/notes/TIDV/index.html
2) http://www.libpng.org/pub/png/spec/1.2/PNG-GammaAppendix.html

E. Logarithmic Transformation

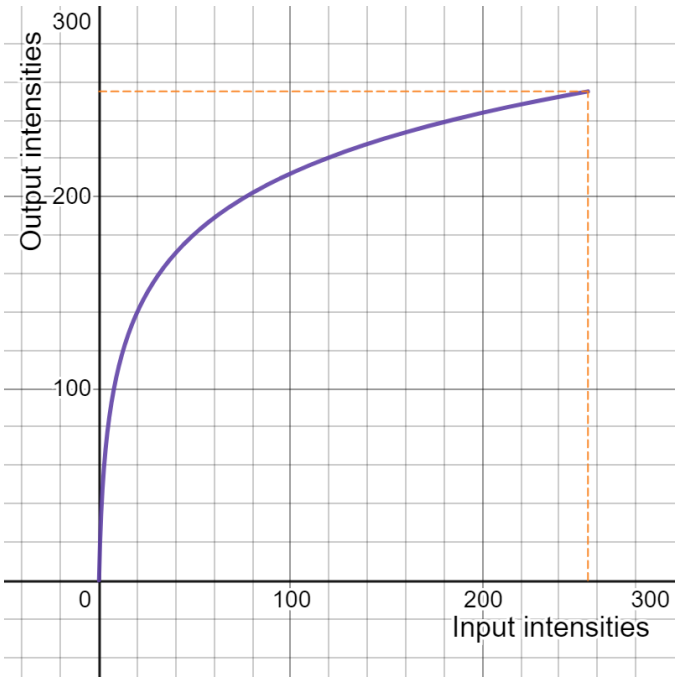


Fig. 5: Plot of V_{out} v/s V_{in} for log transform.

The log-transform takes the input pixel and outputs the logarithm of that pixel, scaled and shifted. Mathematically,

$$V_{out} = A\log(1 + V_{in}) \tag{9}$$

where A is the normalization constant. Note that the logarithm compresses the value range. If we take $V_{in(max)} = 255$, then

$$A = \frac{255}{\log(1 + 255)} = 105.886 \tag{10}$$

Hence as A is high, the lower intensities get matched to very high intensities, for example the intensity $i = 15$ gets matched to $i' = \lfloor 105.89 \times \log(16) \rfloor = 127$, but we do know that $i = 255$ is matched to $i' = 255$ and hence we can see the compression that occurs.

Uses:

1) Compression of dynamic range.
2) Expand dark pixels in image, and contract light pixels.

F. Inversion

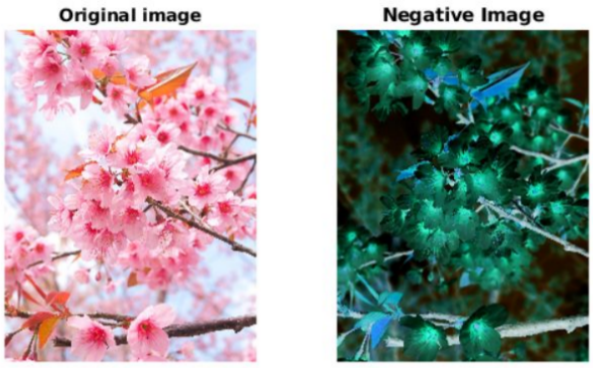


Fig. 6: Negative of an image. Pic credits: GeeksforGeeks

Inversion in this context means finding the negative of an image. For grayscale images, for any intensity $i \in [0, 255]$, the new intensity i' is mapped as

$$i' = 255 - i \tag{11}$$

For color images, similar transform is applied on each of the RGB values of each pixel i.e

$$R' = 255 - R, \quad G' = 255 - G, \quad B' = 255 - B \tag{12}$$

G. Convolution and Kernels

A kernel is a small matrix used in image processing for blurring, sharpening, edge detection etc. by utilizing the convolution operation. Denoting the kernel by ω, the output image $g(x, y)$ in terms of input image $f(x, y)$ after convolution with kernel ω is given as:

$$g(x, y) = \omega \star f(x, y) \tag{13}$$

where

$$\omega \star f(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s, t) f(x + s, y + t) \tag{14}$$

1) Blurring: For the blurring operation, two filters have been implemented.

Gaussian Filter: This kernel blurs the image using the gaussian function. In 2 dimensions, the function is given as

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{15}$$

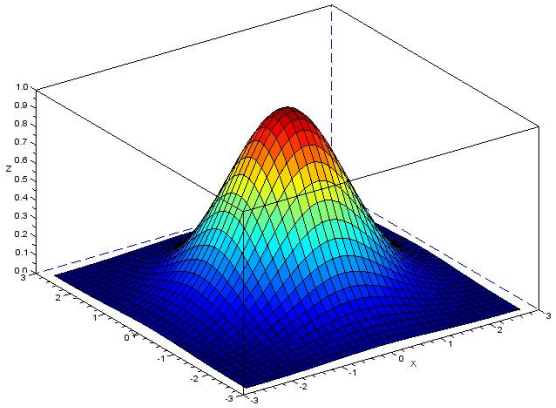


Fig. 7: Gaussian kernel in 3-D. Pic credits: James D. McCaffrey

where x is the horizontal distance from origin, y is the vertical distance from origin, and σ is the standard deviation of the gaussian function. In practice, the gaussian filter acts as a low pass filter and attenuates high frequency signals. In implementation, we give the kernel size and standard deviation to get the kernel. It is suggested to use $\sigma = \sqrt{size}$ and that has been done unless explicitly specified.

Box/Mean Filter: For any dimension $n \times n$, the mean filter is the average of the neighbouring pixel values. Thus

$$\mathcal{B} = [b_{ij}]_{n \times n} \text{ where } b_{ij} = \frac{1}{n^2} \forall\, i, j \tag{16}$$

In case of the Gaussian filter, the user is asked to input either just "G", or "G <size>" or "G <size> <sigma>". In case of Box filter is asked to input either just "B" or "B <size>.

Uses:

1) Removal of outlier pixels which may be noise.

2) Sharpening: The procedure for sharpening used is unsharp masking.

The unsharp filter is a sharpening operator which enhances edges (and other high frequency components in an image) via a procedure which subtracts a smoothed, version of an image from the original image. First, an edge image is produced from image $f(x, y)$ as follows:

$$g(x, y) = f(x, y) - f_{smooth}(x, y) \tag{17}$$

The sharpened image, is now produced by adding a part of the mask image generated above with the original image. Calling α the extent,

$$f_{sharp}(x, y) = f(x, y) + \alpha g(x, y) \tag{18}$$

Usually $\alpha \in [0.2, 0.7]$

Internally, a gaussian filter of size 5×5 and standard deviation $\sigma = \sqrt{5}$ has been used. The extent of sharpening is controlled by the parameter α.

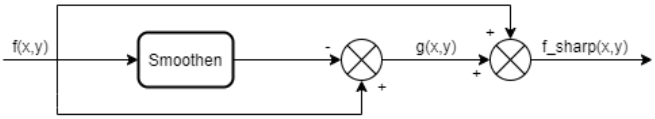


Fig. 8: Flow diagram for unsharp masking

Uses:

1) Sharpening is used to enhance edges in the image.

Note that all the filters have been normalized, i.e

$$\sum_{i,j} \mathcal{F}_{ij} = 1 \tag{19}$$

H. Other operations

1) Rotation: The rotation operation can simply be defined as follows: Given the center of rotation (x_0, y_0) and the point to be rotated (x_1, y_1) by angle φ, the output (x_2, y_2) is given as

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{bmatrix} \tag{20}$$

where the matrix $R(\varphi)$ is the rotation matrix in the x, y plane. The image editor has three options - either rotate clockwise by 90 degrees, rotate anticlockwise by 90 degrees or rotate arbitrarily.

2) Flip: : Flipping of an image can be done either horizontally, vertically or diagonally. Note that

$$diagonal = horizontal + vertical \tag{21}$$

i.e doing a horizontal flip and a vertical flip results in a diagonal flip.

I. Conversion to grayscale

The NTSC convention has been used to convert image to grayscale. The formula is

$$Y = 0.299R + 0.587G + 0.114B \tag{22}$$

IV. Experiments and Results

A. Initial image

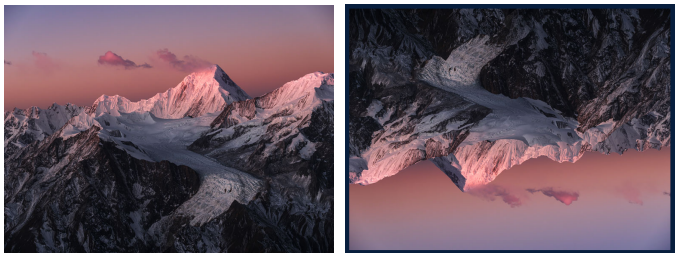
Initial image chosen is as described in *II. B.* [Fig 1]
This image has some dark intensities which can be brightened up, and also has lot of red shades, which can be used to display inversion. In others, wherever a different image is used, is displayed side-by-side for comparison.

B. Diagonal Flip

On choosing the diagonal flip, the resultant image is as follows [Fig 9]
We notice that the image is inverted across the diagonal.

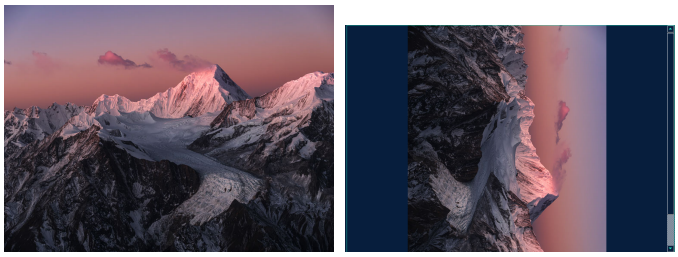
C. Rotating clockwise

On rotating the image clockwise by 90 degrees, the resultant image is as follows [Fig 10]
We notice that the image is rotated clockwise by 90 degrees, and as it didn't fit in the image window, a scroll bar was automatically added for navigation.



(a) Before (b) After

Fig. 9: Diagonal flip

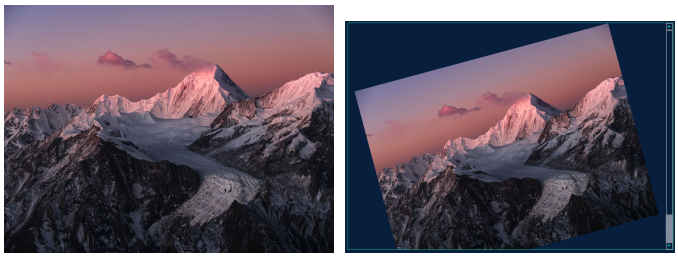


(a) Before (b) After

Fig. 10: 90 degrees clockwise rotation

D. Rotating arbitrarily

On choosing the arbitrary rotation button, and entering "15" in the dialog box that pops up, the resultant image we get is [Fig 11]



(a) Before (b) After

Fig. 11: 15 degrees anticlockwise rotation

We see that the image is rotated anticlockwise by 15 degrees, and again a scroll bar gets added. The size of the image changes, and the blank part of the image becomes transparent.

E. Histogram Equalization

The image we used for histogram equalization has intensities concentrated in a range, and hence becomes a good candidate.
On applying histogram equalization, the resultant image is as follows [Fig 12] The histogram (stored in the `Results/` directory) for the image is as follows [Fig 13]. We notice that initially we had lot of darker intensities present in the image, but after the process of equalization, the histogram tends to be more flatter, although not exactly flat.

(a) Before (b) After

Fig. 12: Histogram Equalization, img @ `Images/histeq2.jpg`

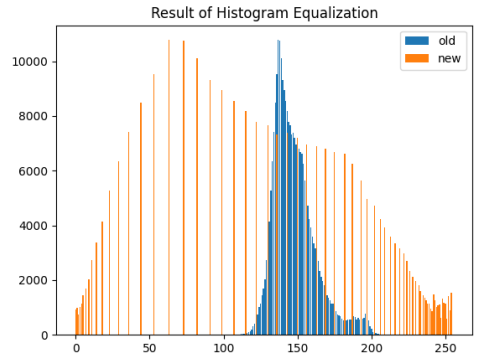
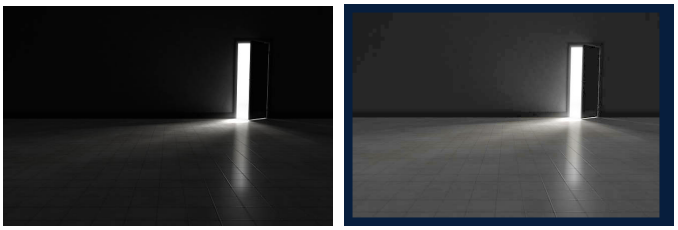


Fig. 13: Histogram of intensities of old and new images

F. Gamma Correction

The image used has lot of dark intensities, and hence is a good candidate for gamma correction for $\gamma < 1$.
On applying gamma correction, with $\gamma = 0.5$, the resultant image is as follows [Fig 14]

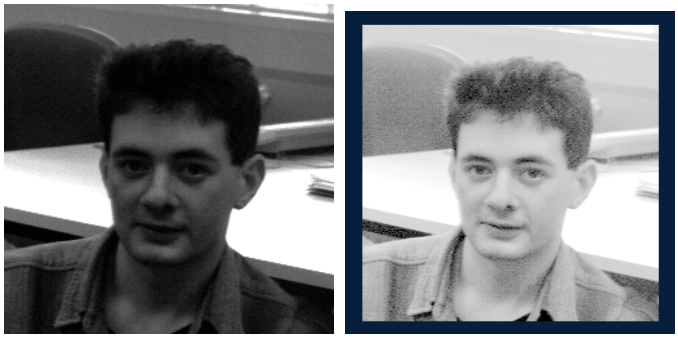


(a) Before (b) After

Fig. 14: Gamma Correction, img @ `Images/gammacc.jpg`

G. Logarithmic Transform

The following image is used for log transform because we can bear the loss of light pixels at cost of enhancing dark ones.
On applying log transform, the resultant image is as follows [Fig 15]
We notice that the intensities become much lighter, agreeing with the graph seen for log transformation.



(a) Before (b) After

Fig. 15: Logarithmic Transform, img @ `Images/dark.png`

H. Gaussian blur

On choosing blur, and entering "G 5 3" applies the Gaussian filter with size 5×5 and $\sigma = 3$. The resultant image is [Fig 16]

(a) Before (b) After

Fig. 16: Gaussian blur, img @ `Images/blur.jpg`

We notice slight blurring in the image. Apart from "G 5 3", we can also use "B 5", which gives a box filter of size 5×5.

I. Invert

On performing the invert operation (to take R, G, B → 255-R, 255-G, 255-B), the resultant image is [Fig 17]

(a) Before (b) After

Fig. 17: Image inversion

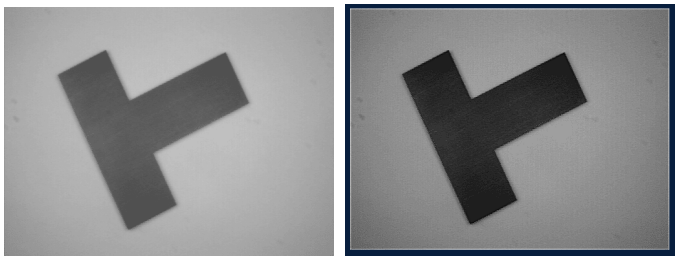
We notice that the image doesn't have much red shades, and predominantly has green/yellow shades and that is due to the fact that the original image had lot of red shades.

J. Red and Green channel

We can choose the Get Channel option, and in the input, put "R" to fetch the R channel of the image. The resultant

(a) Red channel (b) Green channel

Fig. 18: Channel separation



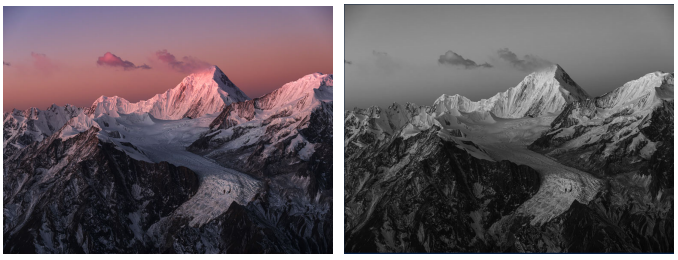
(a) Before (b) After

Fig. 20: Unmask sharpening, img @ `Images/um2.png`

image is as follows [Fig 18]
We notice that the sky part of the image has lot of high intensities, which is obvious due to the fact that this sunset image has red sky. On the other hand, when we input "G" to fetch the green channel of the image, the resultant image will have darker shades.
Here we notice that the sky has relatively low intensities, owing to the lack of green present in the image.

K. Grayscale

On performing the grayscale operation, the output image we get is as follows [Fig 19]



(a) Before (b) After

Fig. 19: Grayscale conversion

Note that different methods might give slightly different grayscale values. The NTSC convention is followed here.

L. Sharpening

The following image had slightly blended edges and hence was a good candidate for image sharpening.
Sharpening has been done on the image for $\alpha = 0.6$. The resultant image is as follows [Fig 20]

V. Conclusion and Discussion

A. Challenges overcome

- ◇ Although I have worked with PyQt5 before, none was at this scale hence understanding the interface took some time.
- ◇ My initial implementation of some algorithms wasn't as efficient, but a brush up of numpy broadcasting and other features helped to speed up times by a lot.
- ◇ As I was unfamiliar with colorspaces except RGB, understanding each one and their geometrical interpretation was time-taking.
- ◇ Implementing everything from scratch without using OpenCV anywhere was time taking.

B. Further work

- ◇ As mentioned earlier in the report, use of hashing can improve space efficiency as currently say we perform 10 clockwise rotation operations, the space taken is 10 times the space needed while with hashing, we would only need to store 4 copies of each of the images with correct hashes.
- ◇ Implementation of a redoStack just as an undoStack and work with them simultaneously.
- ◇ Addition of more filters such as median filter
- ◇ Implementation of more common image processing algorithms and intensity transforms
- ◇ Dynamic resizing of GUI window

References

[1] Rafael C. Gonzales, Richard E. Woods, "Digital Image Processing".
[2] Boney, L., Tewfik, A.H., and Hamdy, K.N., "Digital Watermarks for Audio Signals," *Proceedings of the Third IEEE International Conference on Multimedia*, pp. 473-480, June 1996.
[3] HSL and HSV, *Wikipedia - https://en.wikipedia.org/wiki/HSL_and_HSV*
[4] Grayscale, *Wikipedia - https://en.wikipedia.org/wiki/Grayscale*
[5] Histogram Equalization *http://www.sci.utah.edu/~acoste/uou/Image/project1/Arthur_COSTE_Project_1_report.html*
[6] Gopinath Palanisamy, Palanisamy Ponnusamy, Varun P. Gopi, "An improved luminosity and contrast enhancement framework forfeature preservation in color fundus images," *Signal Image and Video Processing, Springer*, June 2019.
[7] Robert Krutsch, David Tenorio, "Histogram Equalization", *Microcontroller Solutions Group*, June 2011.
[8] Unsharp Masking, *HIPR2 - https://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm*.
[9] PyQt5 documentation, *Qt for Python - https://doc.qt.io/qtforpython/*.
[10] NumPy documentation, *NumPy - https://numpy.org/doc/*.