EE610 Assignment 1 Report (Basic Image Editor)

*Note: Basic Image Editor consist of GUI and image editing features in compliance with the requirement document issued by faculty EE610

Surbhika Rastogi
Electrical Engineering Department,

IITB
(Indian Institute of Technology, Bombay)

Bombay, India
213070056@iitb.ac.in

Abstract- This paper presents an image enhancement tool with some basic features added to modify different parameters of an image. The algorithm that is used to define the functionality of this tool came out from the concept of histogram equalization, gamma correction, log transformation, spatial filtering as part of image processing. Experimental results show that the developed tool can be used to perform several image enhancement operations which includes overall brightness, contrast details in shadow areas and overall sharpness of the image.

Keywords- Digital Image Processing, Image enhancement, spatial filtering

I. INTRODUCTION

Basic Image Editor software is a tool that basically improves several features stores in the image which are not easily perceivable to the human eye otherwise. Main objective of image editing tools is to modify certain attributes of an image in a way required to achieve a specific outcome. There is certain amount of subjectivity involved in outcomes of image enhancement due to human visual system. Enhancement of Images are mainly performed via : Spatial Domain and Frequency Domain. As part of current scope of the image editor tool, enhancement performed via Spatial Domain is implemented.

II. GUI DESIGN

A. Approach

Graphical User Interface provides users a mean to interact with the methods and functionalities defined in a software. It act as a medium to provide input and get results displayed on it. It is important to select intuitive controls and display widgets while designing of GUI since it is the only access point for a user to interact with the tool. And a good GUI greatly improves efficiency and productivity. Following principles were considered while designing of the interface:

- Simplicity in Layout
- Common UI Elements

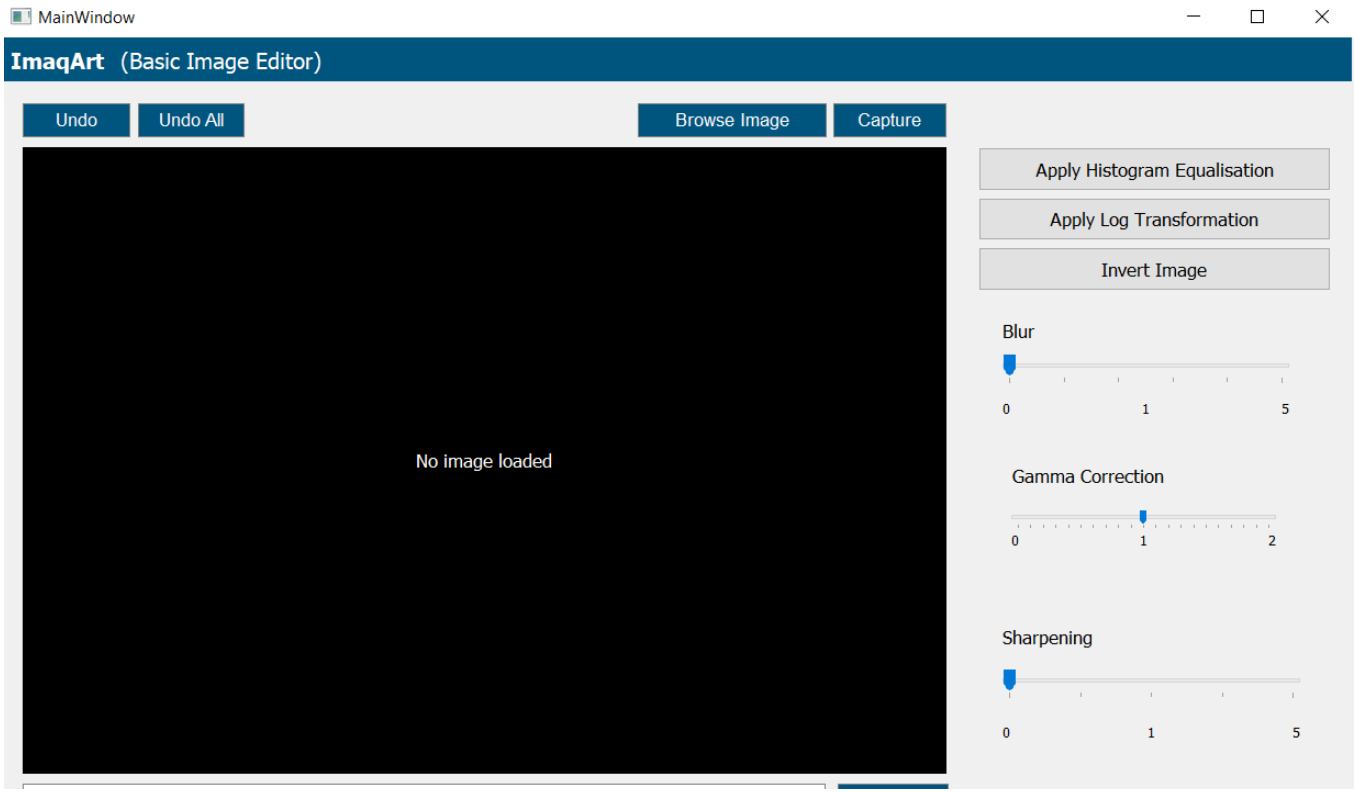


Figure 1 Graphical User Interface of Basic Image Editor

- Distinct colours and textures
- Appropriate font style

B. GUI Features

Graphical User Interface provides the way for users to interact with the software and manipulate the image in order to get results as per the expectations. The snippet of GUI that provides a frontend for manipulation of image parameters can be seen in Figure 1.

The features implemented and controlled by GUI are discussed below:

- Image Selection
- Histogram Equalization
- Gamma Correction
- Log Transform
- Blur effect Intensity control
- Sharpening Intensity control
- Undo/ Undo All
- Save Image

Other than the aforementioned features/operations implemented as part of this editor tool, some additional functionalities are added to the list:

- Image Inversion
- Direct capture the image from camera
- Dynamic Console

A basic approach is to upload the image from available or supported sources to the tool. Once image is uploaded successfully, it can be seen in the tool's Image Display Area in the left and Image transformation operations at right side of the display area. User can click the buttons available for Equalization, Gamma Correction etc. and also can vary the Intensity of the Sharpness and Blur effect using the slider control. Dynamically output can be realised in the image display area, once any operation is performed and also string is logged to dynamic console confirming that the image operation has been performed.

C. Code Overview

Standard libraries are used for development of the Image Editor. QT toolkit is used for creating the graphical user interfaces. Qt allows cross-platform applications that run the software on multiple environments (Linux, Windows, macOS). Python is selected as the programming language for both front-end and back-end development.

QT also has the Python binding, called as PyQt which is used to develop fast and efficient GUI for this Image Editor.

III. IMAGE PROCESSING OPERATIONS

A. Image Selection

The graphical user interface of the Image Editor provides two methods to load an image into the editor for further editing steps:

- Browse Button
- Capture Button

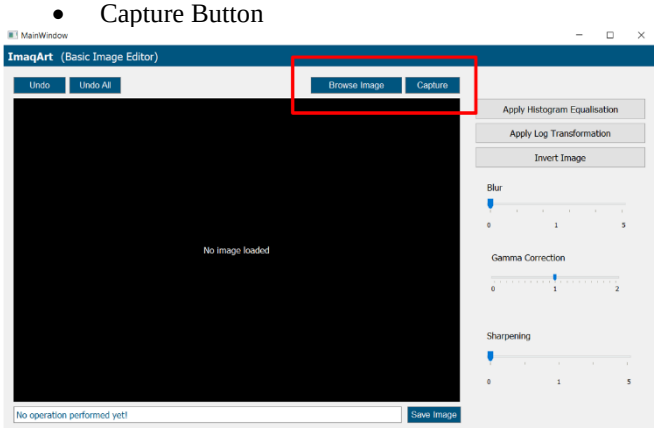


Figure 2 Image loading into the editor can be done through two methods: 1)Browse the file explorer 2)Capture through camera

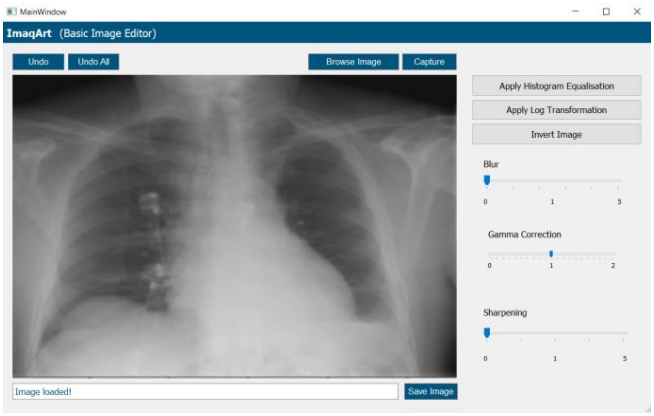


Figure 3 Image displayed on GUI

Browse Button: Clicking on this button opens the file explorer in the current working directory (where the application is stored and running). The user can select any image in *.png, *.jpg, *.gif formats through this file explorer. The selected image will be displayed on the Image Display Area in Image Editor afterwards.

Python code to achieve this task is defined in following codeblock:

```
def browse_files(self):
        """
        Browse for the selected file in current working directory and set the filname
        """

        filter = "Image files (*.jpg *.gif *.png)" #image formats allowed for the editor
        filename                                                = QtWidgets.QFileDialog.getOpenFileName(None,   'Open file',os.getcwd() ,filter)       #browse images in current working directory
        self.filename   =   filename[0]   #load   the filename selected
        self.display_file(self.filename)  #set  the filename for display
        img = cv2.imread(self.filename) #image read for display
        self.org_image      =      cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #store original image copy for further use
        #print(np.shape(img))
```

```
        self.ConsoleDisplay.setText("Image
loaded!") #update the console on GUI.
```

Capture Button: By clicking on this button, the first camera of the device automatically initializes and captures an image which is further displayed on the display area in Image Editor. This function does not require to browse through the file explorer, but it however takes few seconds depending upon the machine to warm up the camera and capture the image.

```
def capture_image(self):
        """
        This method is used to capture an image
directly from the first camera of the device.
        The image captured is also displayed on the
GUI
        """

        #self.ConsoleDisplay.setText("Camera
inilisation...")
        camera = cv2.VideoCapture(0) #camera
initialise
        #self.ConsoleDisplay.setText("Image being
captured...")
        for i in range(10): #9 images read before
saving the 10th one to warm up the camera
            return_value, image = camera.read()

        img = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB) #convert the image to RGB
        self.org_image = img #save the image for
further use
        self.display_image(img) #display on GUI

        del(camera) #delete camera instance
        #print('image capture done')

        self.ConsoleDisplay.setText("Image
captured") #update the console on GUI.
```

The image selected through any method is further converted to the RGB uint8 format for proper display. The image pixels and aspect ratio are also scaled appropriately to adjust in the Image Editor display area.

B. Histogram Equalization

Histogram equalization is used to adjust the pixel values in an image for enhancing the contrast by distributing the intensities more uniformly across the scale of pixel values.

The mathematical formula which is used for this transformation is as follows:

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

$$s_k = \sum_{j=0}^{k} \frac{n_j}{N}$$

```
def enhance_contrast(self,image_matrix, bins=256):
        #img_ary = np.asarray(image_matrix)
        image_flattened = image_matrix.flatten()
#flatten the 2D array image
        image_hist = np.zeros(bins) #empty array
for storing histogram

        # frequency count of each pixel
        for pix in image_matrix:
            image_hist[pix] += 1


        cum_sum = np.cumsum(image_hist) #
cummulative sum
        norm = (cum_sum - cum_sum.min()) * 256

        n_ = cum_sum.max() - cum_sum.min() #
normalization of the pixel values
        uniform_norm = norm / n_
        uniform_norm = uniform_norm.astype('int')
#convert back to default uint8 tyep for display

        # flat histogram
        image_eq = uniform_norm[image_flattened]
        # reshaping the flattened matrix to its
original shape
        image_eq = np.reshape(a=image_eq,
newshape=image_matrix.shape)

        return image_eq
```

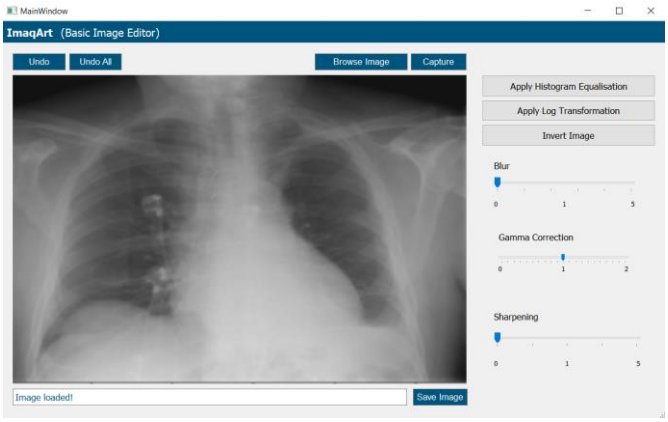


Figure 4 Histogram Equalisation experiment: Original image

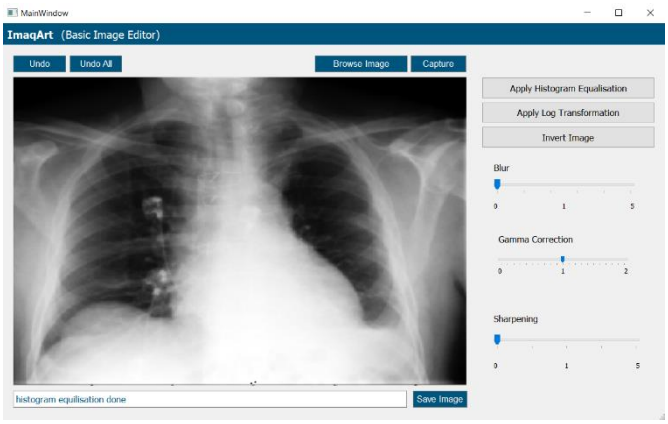


Figure 5Histogram equalisation experiment: transformed image

Histogram equalisation adjusts the intensities of the whole image. Hence it is better suited to images which have a gradient. It can also cause information loss on images otherwise.[1]

Earliest application of histogram equalization has been in improving contrast of landscape images taken from satellite or in foggy environment.

C. Gamma Correction

Gamma Correction is a nonlinear correction applied to image pixels mainly to adjust the colors for various kinds of displays effectively.

The mathematical formula for gamma transform is given by the following equation

$$V_{\text{out}} = AV_{\text{in}}^{\gamma},$$

where A is the scaling constant,
Vin are the input image pixels,
Vout are the transformed image pixels, and
Γ is the correction factor.

```
def gamma_correction(self):

        alpha   =   self.gamma_slider.value()   /10
#assign alpha from front-end slider

        #print(alpha)

        img   =   self.image   #store   imgae   for
transformation

        size = len(np.shape(img)) #image shape
        if size == 3:
            #if image is colored, then convert it
to HSV format
            img_hsv       =       cv2.cvtColor(img,
cv2.COLOR_RGB2HSV)
            #store  V  channel  of  HSV  image  for
transformation
            img_temp = img_hsv[:,:,2]


        elif size == 2:
            #if  image  is  black  and  white  only,
directly apply transformation to the image
            img_temp = img

        img_temp_gamma                              =
((img_temp/255)**(alpha))*255 #gamma transformation
        img_temp_gamma                              =
img_temp_gamma.astype(np.uint8)  #convert  back  to
default uint8 tyep for display

        if size == 3:
            #if image is colored, then convert it
back to RGB format
            img_eq_hsv     =     np.dstack(tup     =
(img_hsv[:,:,0], img_hsv[:,:,1], img_temp_gamma))
            img_gamma   =   cv2.cvtColor(img_eq_hsv,
cv2.COLOR_HSV2RGB)


        elif size == 2:
            #if image is black and white, directly
send to display
            img_gamma = img_temp_gamma


        self.display_image(img_gamma)
       # print('gamma correction done')
        self.ConsoleDisplay.setText("gamma
correction done") #update the console on GUI.
```

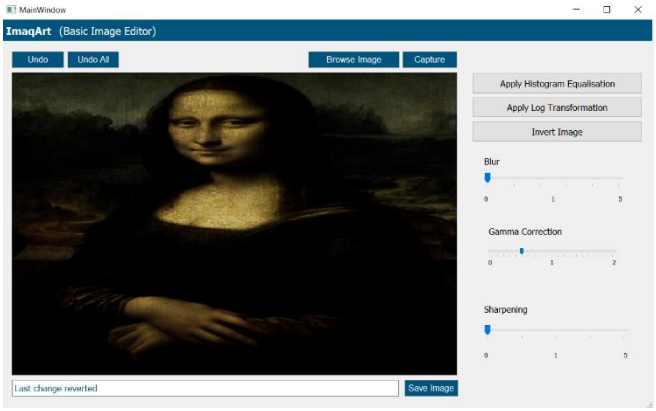


Figure 6 Gamma correction experiment:original image

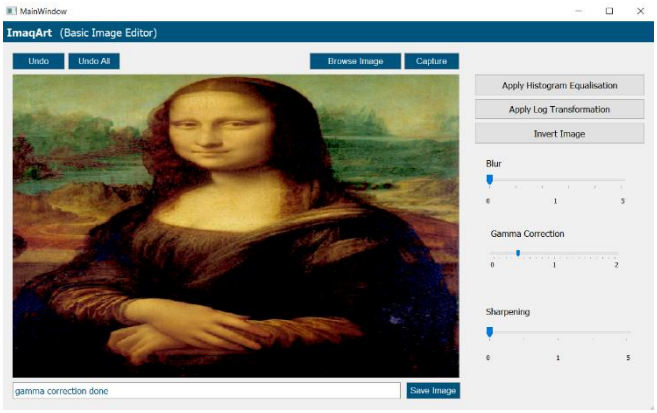


Figure 7 Gamma correction experiment: transformed image

If $\gamma < 1$, the image is enhanced towards the lighter shade and hence increases the brightness. If $\gamma > 1$, the image colors shift towards darker shade.

Different display devices (monitors, projectors etc.) display colors at different intensities and clarity. It has also been the earliest use case of gamma correction. All the display devices has built in gamma correction function for proper display of colors using standard calibration methods.

D. Log Transform

Logarithmic transformation of images is applied on images to achieve brighter intensity of values for darker images, thus making the details present in the darker region more perceivable to the human eye.

The mathematical formula which is used for this transformation is as follows:
s = c * log (1 + r)

where c is the scaling constant to keep the intensity levels in range (0-255),
r is the input intensity pixel values, and
s is the output intensity pixel values.

```
def log_transformation(self):

    #print('log transformation in process...')

    img = self.image
    size = len(np.shape(img))
    if size == 3:
```

```
        #if image is colored, then convert it to HSV
format
        img_hsv         =         cv2.cvtColor(img,
cv2.COLOR_RGB2HSV)
        #store   V   channel   of   HSV   image   for
transformation
        img_temp = img_hsv[:,:,2]

    elif size == 2:
        #if image is black and white only, directly
apply transformation to the image
        img_temp = img

    h,w = np.shape(img_temp)
    print(h,w)

    # img_temp_blur = np.zeros(shape=(h,w))
    img_temp = img_temp.astype(int)

    k = 255 / np.log(1 + np.max(img_temp)) #scaling
factor calculation
    img_temp_log  =  k  *  (np.log(img_temp))  #log
transformation of image
    img_temp_log  =  img_temp_log.astype(np.uint8)
#convert back to default uint8 tyep for display

    if size == 3:
        #if image is colored, then convert it back
to RGB format
        img_log_hsv      =      np.dstack(tup      =
(img_hsv[:,:,0], img_hsv[:,:,1], img_temp_log))
        img_log     =     cv2.cvtColor(img_log_hsv,
cv2.COLOR_HSV2RGB)


    elif size == 2:
        #if image is black and white, directly send
to display
        img_log = img_temp_log


   # img = cv2.blur(img,(val,val))
    self.display_image(img_log)

    #print('log transformation done')
    self.ConsoleDisplay.setText("Log
transformation done") #update the console on GUI.
```

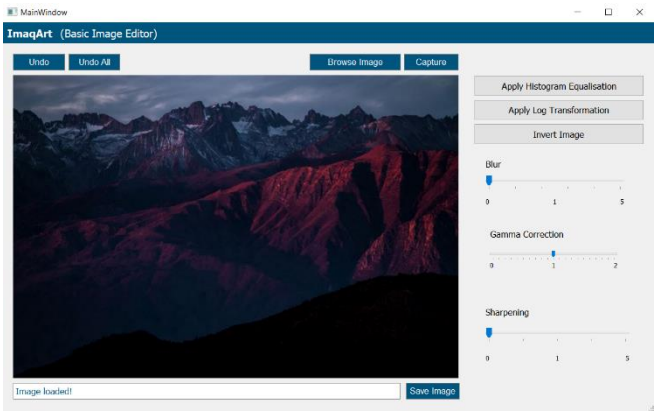


Figure 8 Log transform experiment: original image

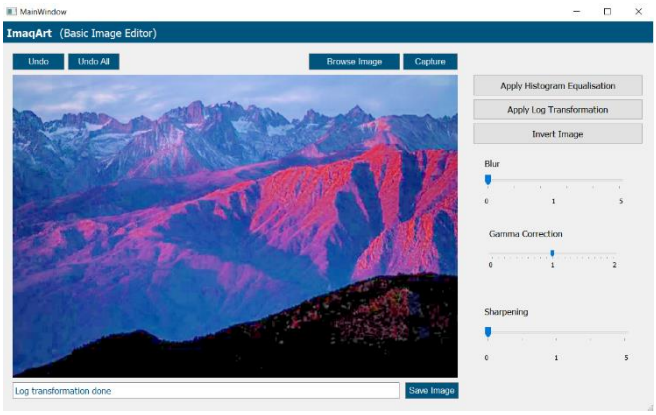


Figure 9 Log transform experiment: transformed image

E. Blur Effect

Many times, image processing requires to smoothen the rapid transition of intensity values such as edge detection algorithm to reduce high frequency noise.

A weighted average of intensity values of neighboring pixels is used to compute the intensity of each pixel in an image.

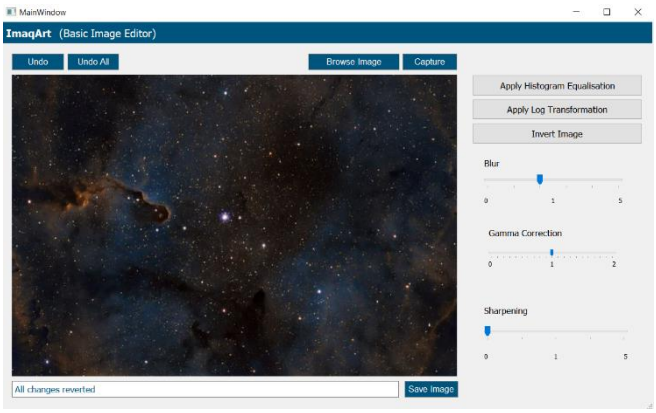


Figure 10 Blurring experiment: original image

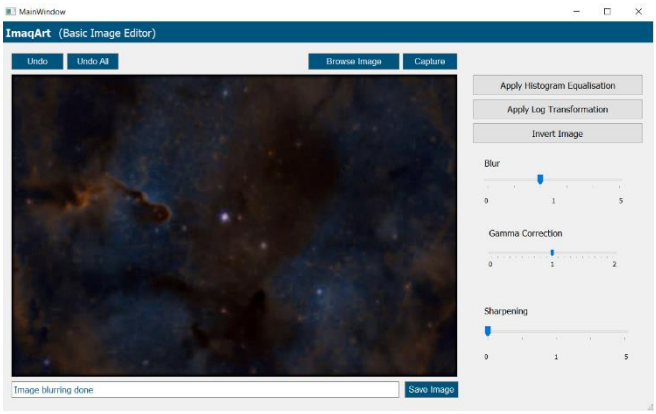


Figure 11 blurring experiment: transformed image

F. Sharpening

Most often it is required to improve the edges and fine details of an image since human eye is quite sensitive to them. Edges consist of high frequency component of the image and may degrade the quality of an image if washed out. Image sharpening improves the visuals of an image by highlighting the edges and fine details.

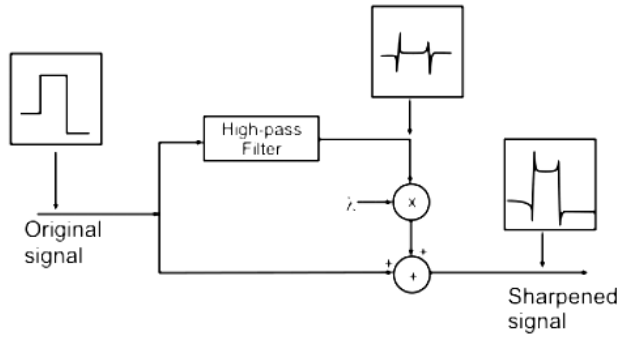
The mathematical formula applied to improve sharpening of image is as follows:

$S = R + \lambda F(x)$

where x is the original pixel value,
F(*) is the high pass filter,
Λ is the sharpening index (greater than or equal to zero), and
S is the sharpened output pixel intensity value.

The effectivity of the operation also depends upon choice of kernel used for the high pass filtering. Following kernel is used for HPF operation:

$$W = \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



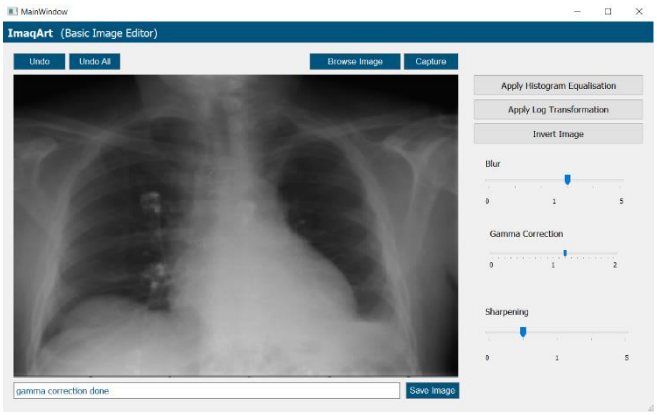


Figure 12 Image sharpening experiment: original image

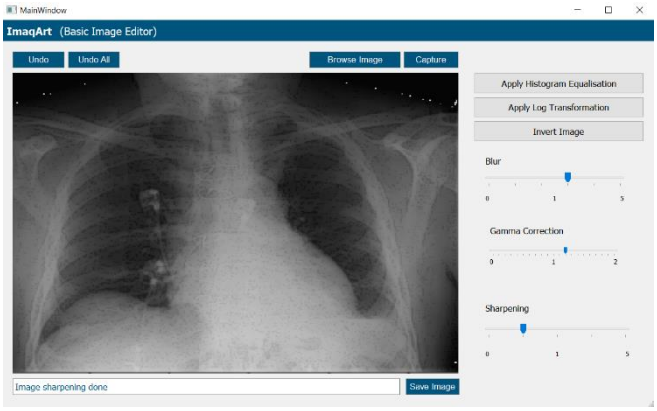


Figure 13 Image sharpening experiment: transformed image

G. Color Inversion

Color inversion feature produces negative of an image. Sometimes, it is useful to invert the intensity values of an image as the features appear more visible when inverted compared to original.

For a uint8 type image (intensity values range from 0-255), every pixel intensity value is subtracted from the maximum i.e. 255 value.

Vout = MaxIntensity – Vin

Where Vin is the input intensity value, and
Vout is the output intensity value.

```
def invert_image(self):
        """
        This method inverts the color of the given
image
        """
        img = self.image # image to be transformed
        img_inv = 255-img #invert the image by
subtracting from maximum value
        self.display_image(img_inv)    #send    for
display

        #print('image inversion done')
        self.ConsoleDisplay.setText("Image
inversion done") #update the console on GUI.
```

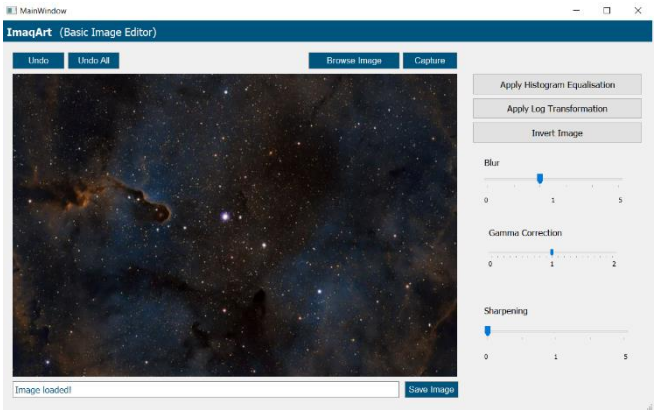


Figure 14 Color inversion experiment: original image

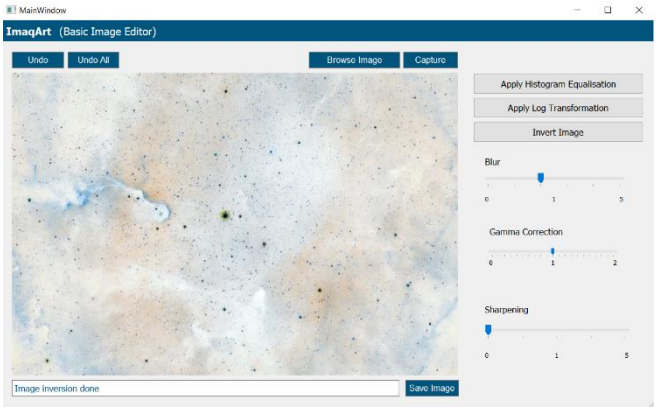


Figure 15 Image sharpening experiment: transformed image

IV. Experiments and Results

A single X-Ray image is hereby taken and transformed through following steps to improve the perceptibility of the image:

1. Gamma Correction
2. Image Sharpening
3. Image inversion

4. Histogram Equalisation

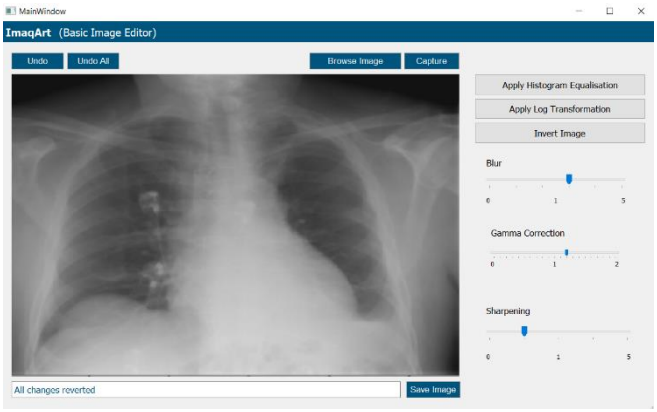


Figure 16 Multiple transformation experiment: original image

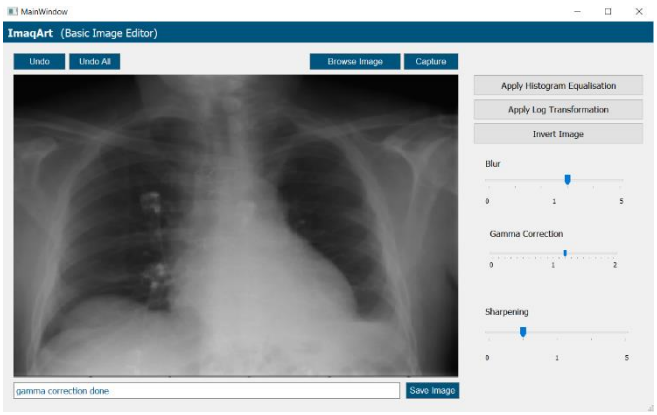


Figure 17Multple transformation experiment: 1st step - gamma correction applied

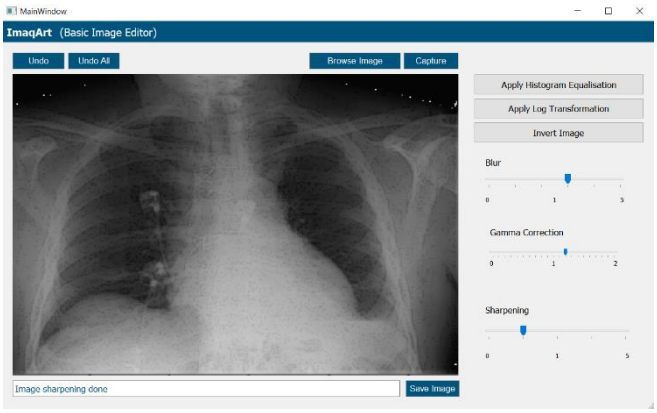


Figure 18 Multiple transformation experiment: 2nd step - Sharpening applied

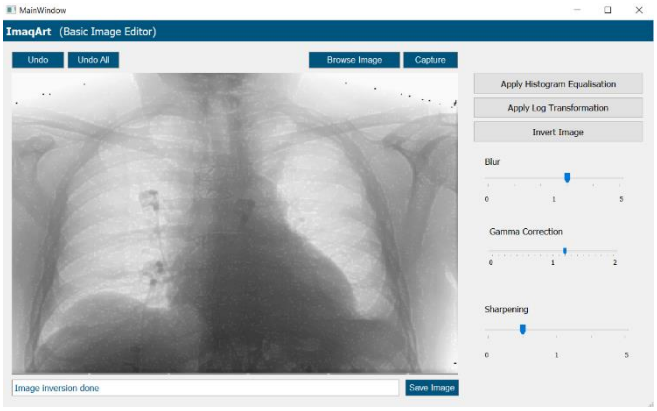


Figure 19 Multiple transformation experiment: 3rd step - color inversion applied

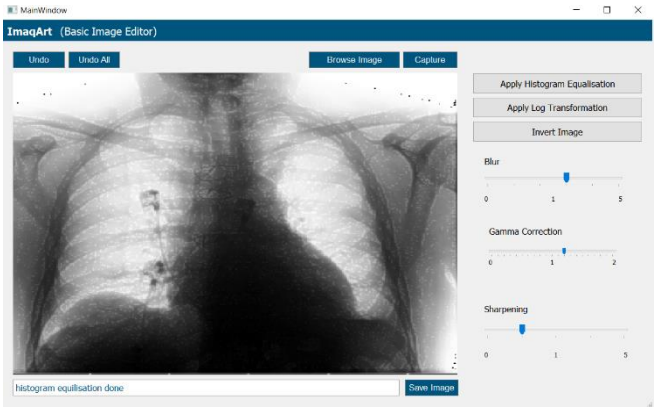


Figure 20 Multiple transformation experiment: 4th step - histogram equalization applied

V. Conclusion

Image editing methods range over a wide variety of modifications over images. However, the applicability of one method on an image dominantly depends upon the specific task that one wants to accomplish through image enhancement. Also, the computational costs of each algorithm play an important role while deploying in several systems which may either be required at high speed or at low computational power. As mentioned in the experiments and results part, the effectiveness of each algorithm can be attained but not limited to be applied separately. Most often these are applied in combinations to realize the practical tasks.

Image processing involves transformation of pixel intensity values from one form to another, such as color-mapping, datatype conversion etc. It has also been the main challenge as sometimes the pixel values clips off due to overflow of the datatype or saturates at maximum/minimum value as a result of an operation. These erroneous pixel values produce several artefacts in the final image. Handling of pixel values should be done carefully by proper scaling methods.

VI. References

Citations:

[1] https://en.wikipedia.org/wiki/Histogram_equalization

Internet Resource:

1. Image transformation:
    - https://arxiv.org/ftp/arxiv/papers/1003/1003.4053.pdf
    - https://nptel.ac.in/content/storage2/courses/117104069/chapter_8/8_32.html
2. Color Spaces conversion:
    - https://www.vocal.com/video/rgb-and-hsvhsihsl-color-space-conversion/
    - https://www.rapidtables.com/convert/color/rgb-to-hsv.html
    - https://www.geeksforgeeks.org/program-change-rgb-color-model-hsv-color-model/
3. histogram equilisation:
    - http://www.sci.utah.edu/~acoste/uou/Image/project1/Arthur_COSTE_Project_1_report.html#eqhistfunction
4. Qt designer:
    - https://realpython.com/qt-designer-python/#using-qt-designers-new-form-dialog
    - https://realpython.com/qt-designer-python/