

IMAGE PROCESSING ASSIGNMENT-1

M.Arun kumar

173079004

ABSTRACT

I have developed tool to perform the basic intensity transformation on the Images. Images here can be grey scale or color but only images of the type png,jpeg,jpg,xmp are accepted as input. Intensity transforms that can be performed on this images using the tool are Histogram Equalization, Gamma transform, Log transform, Blurring Image, Sharpening Image and Edge detection using Sobel operators is added as the additional transform. All of these transforms are applied on some of the images and the results are shown in this report.

1. INTRODUCTION

Main idea behind this tool is to implement various transformation techniques on any of the image. Images can be of many types like color, grey scale etc. Different Images have different underlying data representations like color images can be represented with 3 arrays if the pixel intensities are represented in RGB format, Same image if represented in HSV format will have arrays corresponding to H,S,V respectively each array element will be the value of Hue,Saturation and Value. Similarly grey scale is one in which the value of each pixel represents intensity information. This tool will convert any image given to it into its HSV data and performs all the operations on the Intensity(V) array and then merge it with the original Hue(H) and Saturation(S) arrays to get back the original image.

2. BACKGROUND READ

This tool is implemented using Python[1]. I have used PyQt4[2] binding for implementing GUI it runs on Windows, Linux, Mac OS X and various UNIX platforms. It does not support Android and iOS. To handle color and grey scale images opencv[3] library is used. Any image uploaded will be considered as color image and its HSV arrays[4] are extracted. To perform operations on the Intensity values numpy[5] library is used

3. APPROACH

Apart from transformations there are some file handling and operation control features in this tool. Every operation is as-

sociated with a button or scroll bar(only for sharpening image). Each button in turn when clicked calls the corresponding method to perform the operation on the image. Before any change is made to the data it is stored in a global variable so that the Undo functionality can be implemented easily. Each of the transform that is implemented by this tool and approach followed to implement it is listed below

3.1. Histogram Equalization

Histogram equalization[3] is implemented by taking the CDF of the pixel intensities, for each intensity level number of such pixels in the image are found and this result is stored in a variable(CDF) and it is updated for every intensity level by adding the previous value also the new intensity is calculated by dividing it with the size of the image and then multiplying it with maximum intensity i.e.,255.

3.2. Gamma transform

Each intensity value of the original image is normalized and raised to the power of $1/\gamma$ given by the user this result is multiplied with a constant value to return the intensity range of the original image. In this tool constant for gamma transform[6] is chosen to be 255. So if the user gives gamma to be less than 1 all the intensity values are reduced and the image appears darker. Similarly if gamma is greater than 1 image appears brighter.

3.3. Log transform

Each intensity value of the original image is multiplied by a constant and log of that value is taken as new intensity. In this tool constant for log transform is chosen to be 46 because the intensity range from images is from 0 to 255. log of these values would result in values from 0 to 5.54 to keep the range of intensities same constant is chosen to be 46.

3.4. Blurring Image

To blur an image blurring kernel is used. Gaussian kernel is used and the window size of the kernel is determined by the sigma(variance) value given by the user larger the sigma value more blurring happens and vice-verse. After sigma is taken from the user kernel of size $(2 * \sigma, 2 * \sigma)$ is formed

and the image is convolved with the filter(kernel) to get the blurred image.

3.5. Sharpening Image

To Sharpen an image kernel is used. Laplacian kernel is used and the window size of the kernel (3, 3) is formed and the image is convolved with the image to get the sharpened image. Extent of sharpening is controlled with the scroll bar minimum value of the scroll bar is 1 and maximum is 10. If scroll bar is set to 1 the resultant image is the sum of original image and convolved image. Similarly if the scroll bar is set to 10 the resultant image is sum of the original image and 10 times the convolved image.

3.6. Sobel operators(Horizontal and vertical edge detection)

To detect edges kernel is used. Here Sobel kernel is used and the window size of the kernel (3, 3) is formed and the image is convolved with the image to get the sharpened image. Two kernels are used to detect 2 edges (horizontal and vertical) and the result of both of them is added to display the horizontal and vertical edges.

4. SELECTION OF TEST IMAGES

Following are the images used for testing the various transformation techniques.

4.1. Histogram Equalization



Test image for Histogram equalization[7]

This is image is chosen for histogram equalization because its Histogram is more skewed towards low intensity levels. Results of histogram equalisation will be clearly visible with this image.

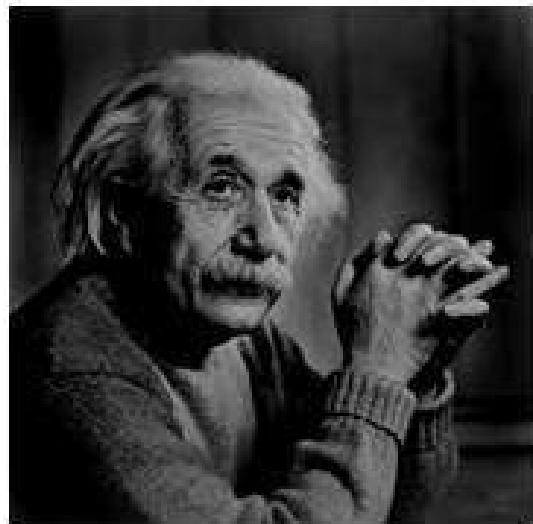
4.2. Gamma Correct



Test image for Gamma correction[6]

This is image is chosen for Gamma correction because it is brighter image and the effect of gamma transform for different gamma values can be clearly observed.

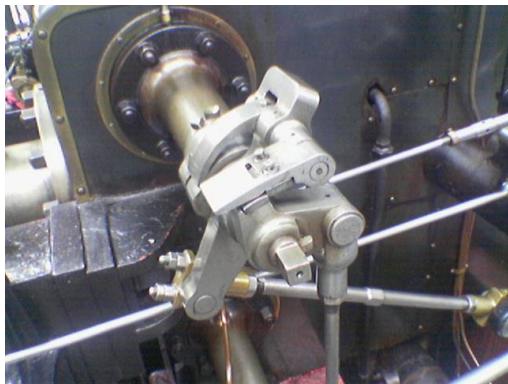
4.3. Log Transform



Test image for Log Transform[8]

Effect of Log transformation is more clearly visible in Grey scale image, So grey scale image is chosen for testing Log transformation.

4.4. Blur Image



Test image for Blurring[9]

Any image can chosen for blurring image but image which has more details in it will give the best output so this image is chosen.

4.5. Sharpen Image



Test image for Sharpening[10]

4.6. Sobel operation



Test image for Sobel opeartion[11]

Image which has more horizontal and vertical edges will be required since Sobel operators for detecting horizontal and vertical edges is used as kernel

5. RESULTS ON TEST IMAGES

Following are the images after appying various transformation techniques.

5.1. Histogram Equalization



Test image after applying Histogram equalization

5.2. Gamma Correct

5.2.1. $\text{gamma} = 0.1$



Test image after applying Gamma correction with $\text{gamma} = 0.1$

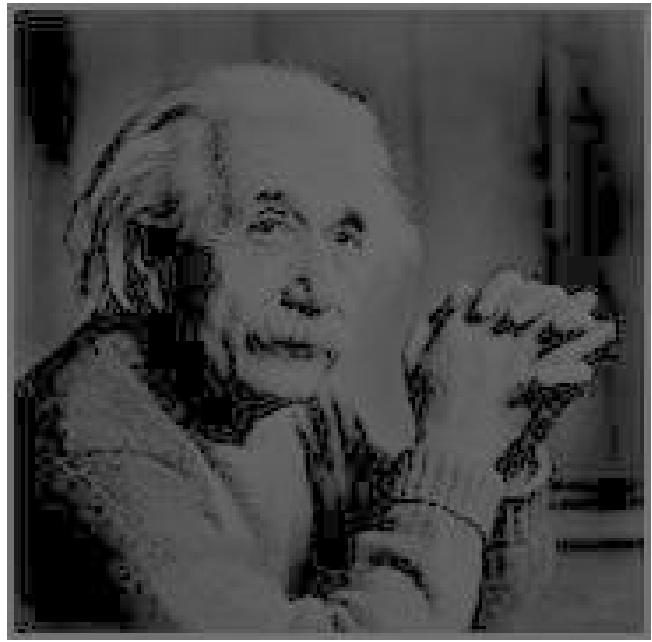
5.2.2. $\text{gamma} = 2.2$

Image after applying gamma transform



Test image after applying Gamma correction with gamma = 2.2

5.3. Log Transform

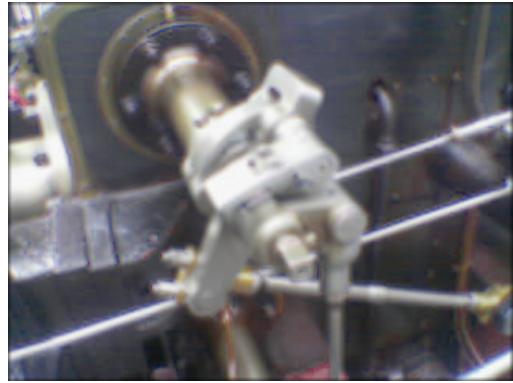


Test image after applying Log Transform

5.4. Blur Image

Results of blurring with different Gaussian kernels

5.4.1. Blur using Gaussian kernel with variance 5



Test image after Blurring with Gaussian kernel with variance 5

5.4.2. Blur using Gaussian kernel with variance 20



Test image after Blurring with Gaussian kernel with variance 20

5.5. Sharpen Image

Result after applying Laplacian (3, 3) kernel



Test image after applying sharpening filter

5.6. Sobel operation

Result of applying Sobel kernel to detect horizontal and vertical edges. Here the test image is convolved with the kernel to detect horizontal edges at the same time another kernel is applied to the test image to detect vertical edges to detect vertical edges. Result of the convolution of these is added to get the horizontal and vertical edges in image



Test image after applying Sobel operator

6. DISSCUSSION

Main problem faced during development is choice of language. I got to know that Matlab has a drag and drop type of making GUI, but it is licensed i wanted to work with open source. The choice I was left with was c++ and python. I was not well versed in both of them. Since python syntax is easy i chose python. Implementation of GUI created a lot of problems I did not know which python binding to use, after searching online and found that tkinter is the basic and primitive binding in python so i chose that and started working with it but after spending entire day on it nothing much was happening so i shifted to PyQt after finding a tutorial[2] on PyQt and completed the GUI part using PyQt. For image operations OpenCV [12]and numpy[5] libraries are used though it was not easy documentation is available online.

If more time was given i would have implemented convolution using vectors to reduce the computations and GUI also can improved by making the window responsive adding short cuts to all the operations etc.,

7. REFERENCES

- [1] “Python oops,” <https://thepythonguru.com/python-inheritance-and-polymorphism/>.

- [2] “Pyqt tutorial,” <https://pythonprogramming.net/basic-gui-pyqt-tutorial/>.
- [3] “Opencv documentation,” https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html.
- [4] “Arrays in python,” <https://www.geeksforgeeks.org/array-python-set-1-introduction-functions/>.
- [5] “numpy documentation page,” <https://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.matrix.max.html>.
- [6] “Lena image,” https://www.dfxstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-6-gamma-correction/lena_gamhi.png.
- [7] “church image,” https://www.photovideoedu.com/Portals/0/Exposure/Ch_01/Ch_01_Lesson_01/Ch_01_Lesson_01_01_Church.jpg.
- [8] “Image for log transform,” <https://www.tutorialspoint.com/dip/images/graylevel3.jpg>.
- [9] “image for blurring,” https://en.wikipedia.org/wiki/Sobel_operator/media/File:Valve_original_1.PNG.
- [10] “Image for sharpening,” <http://1.bp.blogspot.com/-iwX8XxmyXwM/Udf2USu4I/AAAAAAAANS/rT7Yo7OtQ/s>
- [11] “sobel operator image,” <https://upload.wikimedia.org/wikipedia/commons/3/3f/Bikesgray.jpg>.
- [12] “About opencv,” <https://ags.cs.uni-ss18/ImageProcessingOpenCV.pdf>.

8. APPENDIX

```
#!/usr/bin/python

import sys # import libraries needed
import PyQt4
import math
import numpy as np
import cv2 as cv
from PyQt4 import QtGui, QtCore
from PyQt4.QtGui import *
from PyQt4.QtCore import *
from PyQt4.QtCore import pyqtSlot, SIGNAL, SLOT

class Window(QtGui.QMainWindow): #create a class to display a window

    def __init__(self): #method to declare attributes of the class
        super(Window, self).__init__()
        self.setGeometry(50,50,1400,650) # to set the size of the window to 1400*650
        self.setWindowTitle("Basic_Image_Editor") # give title to the window
        self.home() #method called home will have all the main features of the GUI
        self.__pixmap = None # create pixmap attribute to display image to GUI
        self.__mdfd_img_lstchg = None # to store the last the changed image data
        self.__mdfd_img = None # to store the current image data
        self.__img_h = None # empty attribute to store hue values of image
        self.__img_s = None # empty attribute to store saturation values of image
        self.__img_v = None # empty attribute to store intensity value of image
        self.__img_height = None # height of the Image
        self.__img_width = None # width of the Image
        self.lbl = QtGui.QLabel(self) # create a QLabel object to display input image
        self.lbl1 = QtGui.QLabel(self) # create a QLabel object to display title for
            ↪ input image
        self.lbl2 = QtGui.QLabel(self) # create a QLabel object to display title for
            ↪ output image
        self.lbl3 = QtGui.QLabel(self) # create a QLabel object to display output
            ↪ image
        self.lbl_s1 = QtGui.QLabel(self) # create a QLabel object to display scroll
            ↪ title "High"
        self.lbl_s2 = QtGui.QLabel(self) # create a QLabel object to display scroll
            ↪ title "Low"
        self.lbl_s3 = QtGui.QLabel(self) # create a QLabel object to display text for
            ↪ text editor
        self.s2 = QtGui.QScrollBar(self) # create a QScrollBar object to display
            ↪ scrollbar
        self.e2 = QtGui.QLineEdit(self) # create a QLineEdit object to display scroll
            ↪ title

    def home(self): # home method of the QMainWindow
        btn = QtGui.QPushButton("Upload_Image",self) # button for uploading image
        btn.clicked.connect(self.file_open) # go to file_open method when clicked on
            ↪ Upload Image button
        btn.resize(200,40) # resize the button to the required size
        btn.move(500,50 ) # reposition the button at the required position
        btn1 = QtGui.QPushButton("Equalize_histogram",self)
```

```

btn1.clicked.connect(self.hist_equal) # go to hist_equal method when clicked
    ↪ on qualize histogram
btn1.resize(200,40) # resize the button to the required size
btn1.setSizePolicy(QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Expanding)
btn1.move(500,100 )
btn2 = QtGui.QPushButton("Gamma_correct",self)
btn2.clicked.connect(self.gamma_correct_btn) # go to gamma_correct_btn method
    ↪ when clicked on Gamma correct button
btn2.resize(200,40) # resize the button to the required size
btn2.move(500,150 ) # reposition the button at the required position
btn3 = QtGui.QPushButton("Log_transform",self)
btn3.clicked.connect(self.log_transform) # go to log_transform method when
    ↪ clicked on Log transform button
btn3.resize(200,40) # resize the button to the required size
btn3.move(500,200 ) # reposition the button at the required position
btn4 = QtGui.QPushButton("Blur_Image",self)
btn4.clicked.connect(self.blur_img_scr_bar) # go to blur_img_scr_bar method
    ↪ when clicked on Blur Image button
btn4.resize(200,40) # resize the button to the required size
btn4.move(500,250 ) # reposition the button at the required position
btn5 = QtGui.QPushButton("Sharpening",self)
btn5.clicked.connect(self.sharpen_img_scr_bar) # go to sharpen_img_scr_bar
    ↪ method when clicked on Sharpeninge button
btn5.resize(200,40) # resize the button to the required size
btn5.move(500,300 ) # reposition the button at the required position
btn6 = QtGui.QPushButton("Sobel_Operator",self)
btn6.clicked.connect(self.edge_detect) # go to save_image method when clicked
    ↪ on Sobel operator button
btn6.resize(200,40) # resize the button to the required size
btn6.move(500,350 ) # reposition the button at the required position
btn7 = QtGui.QPushButton("Undo_last_Change",self)
btn7.clicked.connect(self.undo) # go to undo method when clicked on Undo last
    ↪ Change button
btn7.resize(200,40) # resize the button to the required size
btn7.move(500,400 ) # reposition the button at the required position
btn8 = QtGui.QPushButton("Undo_All_Changes",self)
btn8.clicked.connect(self.undoall) # go to undoall method when clicked on Undo
    ↪ All Changes button
btn8.resize(200,40) # resize the button to the required size
btn8.move(500,450 ) # reposition the button at the required position
btn9 = QtGui.QPushButton("Save_Image",self)
btn9.clicked.connect(self.save_image) # go to save_image method when clicked
    ↪ on Save Image button
btn9.resize(200,40) # resize the button to the required size
btn9.move(500,500 ) # reposition the button at the required position
btn10 = QtGui.QPushButton("Close_Window",self)
btn10.clicked.connect(self.win_close) # go to win_close method when clicked on
    ↪ Close Windo button
btn10.resize(200,40) # resize the button to the required size
btn10.move(500,550 ) # reposition the button at the required position
self.show() #show the window

def file_open(self): #method to open file

```

```

name = QtGui.QFileDialog.getOpenFileName(self,'Open_File','','Images_(*.png*.
→ *.xpm*.*.jpg*.*.jpeg)') #this will open a dialog box to upload image only
→ png,xpm,jpg,jpeg images are supported
upld_img = QtGui.QImage() # create QImage object to store the uploaded image
→ data
self.__ip_img = cv.imread(str(name),cv.IMREAD_COLOR) # upload the image from
→ the dialog box using imread in opencv library
img_hsv = cv.cvtColor(self.__ip_img, cv.COLOR_BGR2HSV) #convert color image to
→ HSV using cvtColor of opencv
# get image properties.
self.__img_h,self.__img_s,self.__img_v = cv.split(img_hsv)
self.__img_height,self.__img_width = self.__img_v.shape
# print self.__img_v.shape
self.__mdfd_img_lstchg = None
self.__mdfd_img = None
self.__mdfd_img_lstchg = self.__img_v # update the last changed value to
→ uploaded image
self.__mdfd_img = self.__img_v # update the current changed image to uploaded
→ image
if upld_img.load(name): # if the image is uploaded properly then upld_img.load
→ will be true
    self.lbl1.clear() # clear the past content in label if any is present
    self.lbl1.setText("Orignal_Image") # Set title for the input image to
→ display
    self.lbl1.move(200,50) # position the title
    self.lbl1.show() # show the title
    pixmap = QtGui.QPixmap(upld_img) #convert the opencv image to pixmap to
→ display it on GUI
    self.__pixmap = pixmap.scaled(400, 650, QtCore.Qt.KeepAspectRatio) # scale
→ the pixmap to display it on GUI keep the Aspect Ratio of the original
→ image
    self.lbl.clear() # clear the past content in label if any is present
    self.lbl.resize(400,650) # set the size of the input pixmap to 400*650
    self.lbl.move(50,0) # position the input pixmap
    self.lbl.setSizePolicy(QtGui.QSizePolicy.Minimum, QtGui.QSizePolicy.Minimum
→ )
    self.lbl.setScaledContents(False)
    self.lbl.setPixmap(self.__pixmap) # set the pixmap to the label
    self.lbl.show()# show the pixmap as image
    print("Selected_Image_uploaded") #print status to the terminal or IDE
    self.dialog = QProgressDialog('In_progress_please_wait...', 'Cancel', 0,
→ self.__img_v.size, self) # create a progress bar to display the
→ progress of methods we use
    self.dialog.setWindowModality(Qt.WindowModal)
else: #if the image is not uploaded then
    print("Could_not_upload_Image") # print status to the terminal or IDE

def hist_equal(self):# this method is for Histogram Equalization
    self.__mdfd_img_lstchg = self.__mdfd_img # store the last changed image data
→ for undo method
    sum = 0 # initialise sum variable to store the cdf
    hist_equal_img = self.__mdfd_img # assign the image data to temporary variable
→ to do operations on it

```

```

new_img=np.empty_like(hist_equal_img) # create a empty numpy array with the
    ↪ same size of the input image
# print(np.max(hist_equal_img))
for i in range (256): # for each intensity value
    idx = np.where(hist_equal_img == i) # get the indexes of the pixels with
        ↪ that image intensites
    i_intnsty_freq = len(idx[0]) # find the number of such pixels
    sum = sum +i_intnsty_freq # find the CDF
    new_intnsty = (float(sum)/hist_equal_img.size)*255.0 # calculate the new
        ↪ intensity
    # print new_intnsty
    new_img[idx] = new_intnsty # assign the new intensity to new array
    self.__mdfd_img = new_img # now move the data in temp variable to global
        ↪ variable
    self.disp("Histogram_Equalization") # to display the changed image
    print("Histogram_Equalized") # Print status to terminal or IDE

def gamma_correct_btn(self): # method to ask for gamma value when gamma correct
    ↪ button is clicked
    gamma,ok = QtGui.QInputDialog.getDouble(self,"Gamma_value","enter_a_number") #
        ↪ get the input from the user along with the status
    if ok: # if the user inputs any value
        print 'Gamma_value='+str(gamma) # Print status to terminal or IDE
        self.gamma_correct(gamma) # call the gamma_correct method with the given
            ↪ gamma
    else: # if user does not give any value
        print("No_input_gamma_value_given") # Print status to terminal or IDE

def gamma_correct(self,gamma): #method to do gamma correction
    self.__mdfd_img_lstchg = self.__mdfd_img # store the last changed image data
        ↪ for undo method
    gamma_correct_img = self.__mdfd_img # store the image data to temporary array
    new_img_2=np.empty_like(gamma_correct_img) # create a temporary array to store
        ↪ calculated values
    c = 255 # let c be 1
    for i in range (256): # for each intensity value
        idx = (gamma_correct_img == i) #get the indexes with the same intensity
            ↪ values
        new_intnsty = c*((float(i)/255.0)**(1/gamma)) # apply gamma transform on
            ↪ each intensity level
        new_img_2[idx] = int(new_intnsty) # store the values in temporary array
    self.__mdfd_img = new_img_2 # save the gamma corrected image to global
        ↪ variable
    self.disp("Gamma_transformation")# to display the changed image
    print("Gamma_transformation_Applied") # Print status to terminal or IDE

def log_transform(self):
    self.__mdfd_img_lstchg = self.__mdfd_img # store the last changed image data
        ↪ for undo method
    log_trnsfrm_img = self.__mdfd_img # temp array to do operations on
    new_img_3 = np.empty_like(log_trnsfrm_img) # empty array to assign new
        ↪ intensity values
    c = 47 # let c be 100
    for i in range (256): # for each instensity value of the image

```

```

idx = (log_trnsfrm_img == i) # get the indexes with the same intensity
    ↪ levels
new_intnsty = float(c*(math.log10(i+1))) # apply log transformation
new_img_3[idx] = new_intnsty # assign the new intensity values to the
    ↪ temporary array
self.__mdfd_img = new_img_3 # store the modified data to the global variable
self.disp("Log_transformation") # to display the changed image
print("Log_transformation_Applied") # Print status to terminal or IDE

def blur_img_scr_bar(self):
    self.lbl_s3.resize(500,50) #label to display title for output image
    self.lbl_s3.setText("Please_Enter_an_Integer_value_Sigma_for_GaussianBlur") #
        ↪ title text
    self.lbl_s3.move(100,590) #positioning
    self.lbl_s3.show() #display title
    self.e2.setValidator(QIntValidator())#text box setting to allow only integer
        ↪ values
    self.e2.move(500,600) #positioning
    btn.blur_img = QPushButton('OK', self) #button to click ok to start operaion
        ↪ on the input
    btn.blur_img.resize(50,30) #resize the button
    btn.blur_img.move(610, 600) #positioning
    btn.blur_img.show() #display button
    self.e2.show() #display text box
    btn.blur_img.clicked.connect(lambda: self.blur_img(int(self.e2.text()))) #call
        ↪ blur_img when clicked

def blur_img(self,sigma):
    self.__mdfd_img_lstchg = self.__mdfd_img # store the last changed image data
        ↪ for undo method
    x_count = -1#initialise
    y_count = -1#initialise
    filter = np.zeros((2*sigma+1,2*sigma+1), dtype=np.float) #empty filter kernel
    blur_img = self.__mdfd_img#take the data to temp array
    padd.blur_img = np.append(np.zeros((sigma,self.__img_width)), blur_img, axis
        ↪ =0) #padd with zeros
    padd.blur_img = np.append(padd.blur_img,np.zeros((sigma,self.__img_width)),
        ↪ axis=0) #padd with zeros
    padd.blur_img = np.append(np.zeros((self.__img_height+2*sigma,sigma)),
        ↪ padd.blur_img, axis=1) #padd with zeros
    padd.blur_img = np.append(padd.blur_img,np.zeros((self.__img_height+2*sigma,
        ↪ sigma)),axis=1) #padd with zeros
    new_img_4 = np.empty_like(blur_img) #empty array for storing the output
    for x in range(-sigma,sigma+1):#for the rows of the filter
        x_count+=1
        y_count = -1
        for y in range(-sigma,sigma+1):#for the columns of the filter
            y_count+=1
            filter[x_count,y_count] = math.exp(-(x**2.0+y**2.0)/(2.0*sigma*sigma))#
                ↪ compute the gaussian blur kernel
    neighbourhood = np.zeros((2*sigma+1,2*sigma+1))#window
    progress = 0#to display progress in progres bar
    self.dialog.forceShow() # show the progress bar
    for j in range(sigma,self.__img_height+sigma):#for the rows of the image

```

```

for k in range(sigma, self.__img_width+sigma): #for the columns of the
    ↪ images
    neighbourhood = padd.blur_img[j-sigma:j+sigma+1,k-sigma:k+sigma+1] #take
        ↪ the pixels in neighbourhood of the pixel
    new_img_4[j-sigma,k-sigma] = np.sum(neighbourhood*filter, dtype=np.float)
        ↪ /np.sum(filter, dtype=np.float) #multiply window with filter and
        ↪ average over the filter
    progress = progress + 1#increment the status
    if(progress%5000==0):#display progress every 5000 loops
        self.dialog.setValue(progress) #to display progress
    if(self.dialog.wasCanceled()):#if the cancel button is pressed
        break # stop the loop
    self.dialog.setValue(progress) # set the progress
    self.__mdfd_img = new_img_4 #store the computed values in global variable
    self.disp("Blurred_Image",1) # to display the changed image
    print("Image_Blurred") # Print status to terminal or IDE

def sharpen_img_scr_bar(self):
    self.lbl_s1.resize(500,50) #label to display title for output image
    self.lbl_s1.setText("Please_Enter_an_Integer_value_Sigma_for_Laplacian_Kernel"
        ↪ ) #title text
    self.lbl_s1.move(100,590) #positioning
    self.lbl_s1.show() #display title
    self.e2.setValidator(QIntValidator()) #text box setting to allow only integer
        ↪ values
    self.e2.move(500,600) #positioning
    btn_sharp_img = QPushButton('OK', self) #button to click ok to start operation
        ↪ on the input
    btn_sharp_img.resize(50,30) #resize the button
    btn_sharp_img.move(610, 600) #positioning
    btn_sharp_img.show() #display button
    self.e2.show() #display text box
    btn_sharp_img.clicked.connect(lambda: self.sharpen_img(int(self.e2.text()))) #
        ↪ call sharpen_img when clicked

def sharpen_img(self,sigma): # to sharpen the image
    self.__mdfd_img_lstchg = self.__mdfd_img # store the last changed image data
        ↪ for undo method
    filter = np.zeros((3,3), dtype=np.float) #initialise filter with zeros
    sharp_img = self.__mdfd_img #take image data to a temp variable
    padd_sharp_img = np.insert(sharp_img,[0],0, axis = 0) #padd zeros to the image
    padd_sharp_img = np.insert(padd_sharp_img,[self.__img_height+1],0, axis = 0) #
        ↪ padd zeros to the image
    padd_sharp_img = np.insert(padd_sharp_img,[0],0, axis = 1) #padd zeros to the
        ↪ image
    padd_sharp_img = np.insert(padd_sharp_img,[self.__img_width+1],0, axis = 1) #
        ↪ padd zeros to the image
    new_img_5 = np.empty_like(sharp_img) #temp array to store new values
    # sigma = self.s2.value() # get the constant to multiply with the image
    filter = (np.array([[-1.0,-1.0,-1.0],[-1.0,8.0,-1.0],[-1.0,-1.0,-1.0]],dtype =
        ↪ "float")*sigma/10.0)+np.array
        ↪ ([[0.0,0.0,0.0],[0.0,1.0,0.0],[0.0,0.0,0.0]],dtype ="float")
    # filter = np.array([[0.0,1.0,0.0],[1.0,-4.0,1.0],[0.0,1.0,0.0]],dtype =
        ↪ "float")

```

```

neighbourhood = np.zeros((3,3), dtype = "float") # initialise window
# new_img_5 = cv.filter2D(sharp_img,-1,filter)
progress = 0 #initialise progress variable
self.dialog.forceShow() #show the progress bar
height,width = padd_sharp_img.shape
# print(padd_sharp_img[padd_sharp_img==np.max(padd_sharp_img)])
# print(np.max(padd_sharp_img),np.max(self.__mdfd_img),np.max(self.__img_v))
for j in range(1,height-1): #for row in image
    for k in range(1,width-1): #for columns in image
        progress = progress+1 # invrement progress value
        neighbourhood = padd_sharp_img[j-1:j+2,k-1:k+2] #neighbourhood of the
        ↪ pixel
        new_img_5[j-1,k-1] = np.sum(neighbourhood*filter, dtype="float")
        #print(neighbourhood*filter,np.sum(neighbourhood*filter),new_img_5[j-1,k
        ↪ -1],self.__img_v[j-1,k-1])
        if(progress%5000==0): #display progress for every 5000 loops
            self.dialog.setValue(progress) # display the progress
        if(self.dialog.wasCanceled()): # cancel button is pressed
            break # stop execution and return to the main window
        self.dialog.setValue(progress) #display the progress using progress bar
        self.__mdfd_img = new_img_5 # save the final sharpened image
        # np.clip(self.__mdfd_img,0,255,out=self.__mdfd_img) #clip the values to make
        ↪ them lie in (0,255)
        self.disp("Sharpened_Image",1,1) # to display the changed image
print("Image_Sharpened") # Print status to terminal or IDE

def edge_detect(self):
    self.__mdfd_img_lstchg = self.__mdfd_img # store the last changed image data
    ↪ for undo method
    filter_6 = np.zeros((3,3), dtype=np.float) #filter for edge detection
    filter_7 = np.zeros((3,3), dtype=np.float) #filter for edge detection
    blur_img = self.__mdfd_img #take the image data to a empty array
    padd.blur_img = np.append(np.zeros((1,self.__img_width)), blur_img, axis=0) #
    ↪ padd zeros to the image
    padd.blur_img = np.append(padd.blur_img,np.zeros((1,self.__img_width)), axis
    ↪ =0) #padd zeros to the image
    padd.blur_img = np.append(np.zeros((self.__img_height+2,1)), padd.blur_img,
    ↪ axis=1) #padd zeros to the image
    padd.blur_img = np.append(padd.blur_img,np.zeros((self.__img_height+2,1)),axis
    ↪ =1) #padd zeros to the image
    new_img_7 = np.empty_like(blur_img) # empty array to store the computed data
    new_img_6 = np.empty_like(blur_img) # empty array to store the computed data
    new_img_final = np.empty_like(blur_img) # store final image data
    filter_x = np.array([[1.0,0.0,-1.0],[2.0,0.0,-2.0],[1.0,-0.0,-1.0]]) #filter
    ↪ to detect horizontal edges
    filter_y = np.array([[1.0,2.0,1.0],[0.0,0.0,0.0],[-1.0,-2.0,-1.0]]) #filter to
    ↪ detect vertical edges
    # print(padd.blur_img[:,0],padd.blur_img[0,:],padd.blur_img[self.__img_height
    ↪ +1,:],padd.blur_img[:,self.__img_width+1])
    neighbourhood = np.zeros((3,3)) #window of size 3*3
    progress = 0 #variable to display progress
    self.dialog.forceShow() # show the progress dialog box
    for j in range(1,self.__img_height+1): #for rows in image
        for k in range(1,self.__img_width+1): #for columns in image

```

```

neighbourhood = padd.blur_img[j-1:j+2,k-1:k+2] # get pixels intensites
    ↪ of the neighbourhood
new_img_7[j-1,k-1] = np.sum(neighbourhood*filter_x,dtype=np.float) #
    ↪ convolve with the filter to get horizontal edges
new_img_6[j-1,k-1] = np.sum(neighbourhood*filter_y,dtype=np.float) #
    ↪ convolve with the filter to get vertical edges
progress = progress + 1 #increment the progress value
new_img_final[j-1,k-1] = new_img_7[j-1,k-1]+new_img_6[j-1,k-1] #add the
    ↪ horizontal and vertical edge
if(progress%5000==0): # display the status every 5000 loops
    self.dialog.setValue(progress) #set the progress value
if(self.dialog.wasCanceled()): #if the cancel button is pressed
    break # stop the operation
self.dialog.setValue(progress)
np.clip(new_img_final,0,70 , out = new_img_final) #keep the intensites in the
    ↪ range(0,255)
self.__mdfd_img = new_img_final #store the transformed data in the global
    ↪ variable
self.disp("Image_Edges")# to display the changed image
print("Image_Edges_Detected") # Print status to terminal or IDE

def undoall(self): # to undo all changes done on the image
    self.__mdfd_img = self.__img_v # change the data in the current changed data
        ↪ to original image data
    # self.__mdfd_img_lstchg = self.__img_v
    self.disp("All_changes undone") # To Display title
    print("All_changes UNDONE") # Print status to terminal or IDE

def undo(self): #to undo the last change done on the image
    self.__mdfd_img = self.__mdfd_img_lstchg #update the current changed image as
        ↪ last changed image
    self.disp("Last_change undone") # To Display title
    print("Last_change UNDONE") # Print status to terminal or IDE

def save_image(self): # this method is used for saving the image to the file
    name = QtGui.QFileDialog.getSaveFileName(self, 'Save_File','','Images_(*.png_
        ↪ *.xpm_*.jpg_*.jpeg') # tp open a dialog box to input image
    itos = cv.merge([self.__img_h,self.__img_s, self.__mdfd_img])#merge intensity
        ↪ with the hue and saturation
    itos = cv.cvtColor(itos, cv.COLOR_HSV2RGB)#convert hsv to rgb image
    img_to_save = QtGui.QPixmap(QtGui.QImage(itos,self.__img_width, self.
        ↪ __img_height,3*self.__img_width, QtGui.QImage.Format_RGB888)) # convert
        ↪ opencv image to pixmap to display in gui
    if img_to_save.save(name):#if the image is saved
        print("Image_Saved_To_file") # Print status to terminal or IDE
    else:#if the could not be saved
        print("Could_not_save_the_Image_to_folder") # Print status to terminal or
            ↪ IDE

def win_close(self): # this method is used for closing the window
    print("Window_closed") # Print status to terminal or IDE
    sys.exit() #exit the application

```

```

def disp(self,txt,flag = 0,scroll = 0): # this method is used to display the
    → transformed image to GUI
    if (flag == 0): #whether to clear some labels or not is decided by this flag
        → variable
        self.s2.hide() #to hide the scroll bar
        self.lbl_s3.clear() #to clear the label to show new objects
        self.e2.clear() #to clear the label to show new objects
        self.e2.hide() #to hide the text box
        self.lbl_s1.clear() #to clear the label to show new objects
        self.lbl_s2.clear() #to clear the label to show new objects
    if (scroll == 0):#if the button other than sharpen is pressed
        self.s2.setValue(1) #reset the value every time
    img_pix1 = cv.merge([self.__img_h,self.__img_s, self.__mdfd_img]) #merge the v
        → with h and s using cv.merge
    img_color = cv.cvtColor(img_pix1, cv.COLOR_HSV2RGB) #convert the image to
        → color image
    pix_img = QtGui.QPixmap(QtGui.QImage(img_color,self.__img_width, self.
        → __img_height,3*self.__img_width, QtGui.QImage.Format_RGB888)) # convert
        → opencv image to pixmap to display it to the user
    self.lbl2.clear() #to clear the label to show new objects
    self.lbl2.setText(txt) #set the text to display
    self.lbl2.resize(300,50) #resize the label to required size
    self.lbl2.move(950,0) #positioning the label
    self.lbl2.show() #show the label
    pix_img= pix_img.scaled(600,600, QtCore.Qt.KeepAspectRatio)
    self.lbl3.clear() #to clear the label to show new objects
    self.lbl3.resize(600,600) #resize the label to required size
    self.lbl3.move(720,40) #positioning the label
    self.lbl3.setSizePolicy(QtGui.QSizePolicy.Minimum, QtGui.QSizePolicy.Minimum)
    self.lbl3.setScaledContents(False) #keep the image as it is while scaling
    self.lbl3.setPixmap(pix_img) #shoe the image
    self.lbl3.show() #show the label

def main(): # define a main class to call window created
    app = QtGui.QApplication(sys.argv) # start a qtgui application
    GUI = Window() #create an object of the window
    # GUI.disp() # display it
    sys.exit(app.exec_()) #close the window
main()

```