IMAGE PROCESSING ASSIGNMENT-2

M.Arun kumar

173079004

ABSTRACT

I have developed tool to perform the Image restoration on the degraded Images. Images here images are color but only images of the type png,jpeg,jpg,xmp are accepted as input where as kernel is assumed to be grey scale. Several restoration techniques that can be performed on images using the tool are Inverse Filtering, Radial Inverse Filtering, Weiner Filtering, LS Filtering All of these Filters 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 Image restoration techniques on the degraded Images. .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 degraded image given to it into its RGB data and performs all the operations on the on R,G,B arrays separately and then merge it to get back the restored image.

2. BACKGROUND READ

This tool is implemented using Python[**?**]. I have used PyQt4[**?**] 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[**?**] library is used. Any image uploaded will be considered as color image and its RGB arrays[**?**] are extracted. To perform operations on the R,G,B separately numpy[**?**] library is used. Also to compute SSIM skimage is used.

3. APPROACH

Apart from transformations there are some file handling and operation control features in this tool. Every operation is associated with a button . Each button in turn when clicked calls the corresponding method to perform the operation on the image. Each of the transform that is implemented by this tool and approach followed to implement it is listed below

3.1. Assumption

Here degraded images are taken form the source provided. Four different kernels are taken and the performance metric are calculated. Results for the four images are shown all the images are assumed to be degraded by the same kernel

3.2. Inverse Filtering

Inverse Filtering can be done easily in Frequency domain. After getting the degraded image perform FFT on it to get the Frequency transform. Similarly do the FFt separately for R,G,B channels of the degraded image. Since we are operating on pixels values directly before preforming any operation just normalize the kernel such that the average intensity of the image does not change.

3.3. Radial Inverse Filtering

Radial inverse filtering is same as the inverse filtering but because of the noise present in the image will be amplified because the kernel values at the noise will low so noise will be increased. To avoid magnifying noise truncated kernel is used. A cutoff from user is taken and a the result we obtain after getting the inverse filtering is passed through the ideal low pass filter generated by making pixel values as 0 when the range exceeds else it is 1.

3.4. Weiner Filtering

To perform Weiner Filtering gamma value is required. This value is taken from the user and then the Fourier Transform of the image is obtained by taking the FFT of B,G,R channels separately. Similarly Fourier Transform of the kernel is obtained after proper padding. Weiner Filtering is modification of Radial Inverse filtering i.e., truncation of the kernel happens as the noise factor in the degraded image increases but this tool is used to implement approximate weiner filter in this case the values is taken from the user and applied in the approximation.

3.5. LS Filtering

Constrained Least Square Filtering is implemented by taking the gamma from the user. Decision for optimum gamma is left for the user. So for every image optimum gamma changes and is depends on many factors such as degradation kernel, degraded image, ground truth etc., Here after finding the Fourier transform of the kernel(after sufficiently padding) and Fourier transform of the image with R,G,B values separately. Here the filter is obtained by dividing the conjugate of the Fourier of the padded kernel dividing with sum of the square of the absolute of the Fourier of the padded kernel added with gamma multiplied with square of the absolute of the Fourier of the padded kernel.

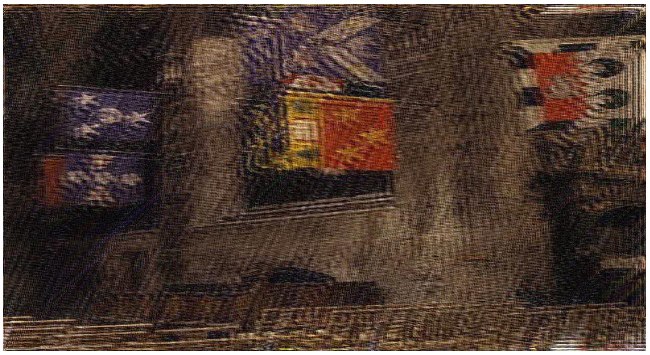
3.6. Calculate performance Metrics

Two metrics are used to calculate the performance of the restoration techniques used here. PSNR is calculated after computing the Mean Square error of the restored image with ground truth. After calculating MSE, PSNR is obtained by taking log of the MSE divided by square root of the MSE. PSNR is in dB and high PSNR is good image quantitatively but it may not be good image because judgment is based on the user perspective, SSIM metric offer such analysis to some extent and here skimage library is used to compute SSIM from the restored image and ground truth. SSIM ranges from 0 to 1. values closer to one are desired outputs.

4. SOME RESULTS

Following are the images after applying various restoration techniques.

4.1. Inverse Filtering

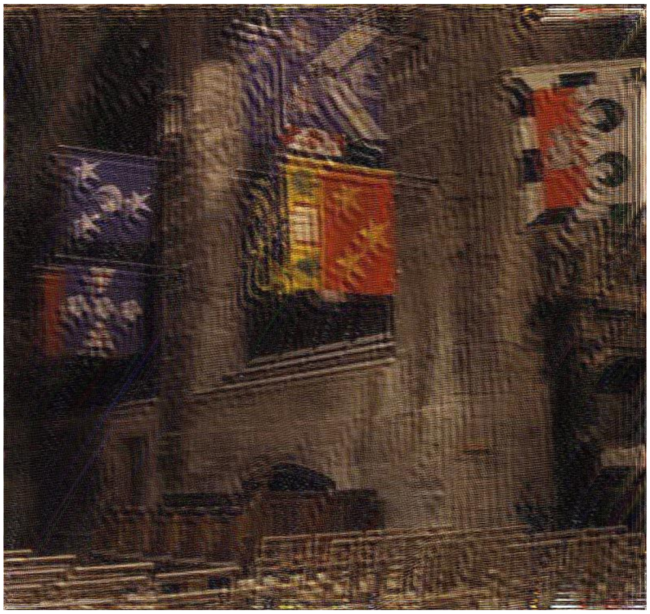


Test image after performing Inverse Filtering

PSNR:17.920233880580266 dB
SSIM:0.008234314887168165

4.2. Radial Inverse Filtering

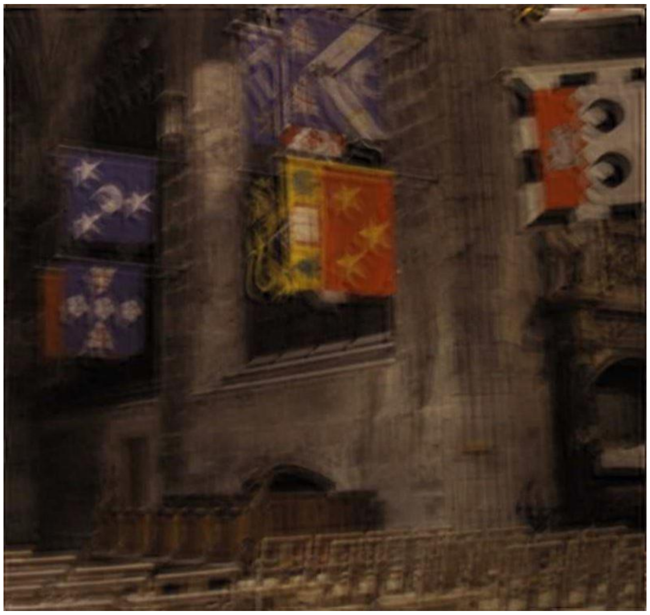
4.2.1. gamma = 150



Test image after performing Radial inverse filtering

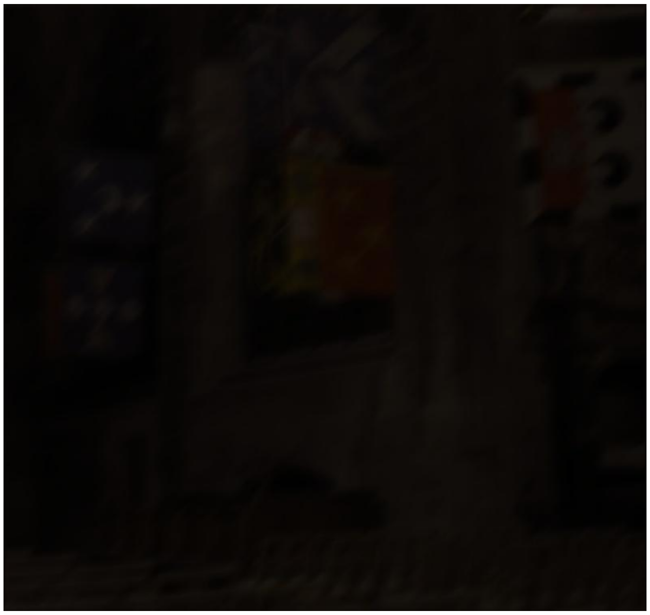
PSNR:17.920233880580266 dB
SSIM:0.008234314887168165

4.3. Weiner Filtering



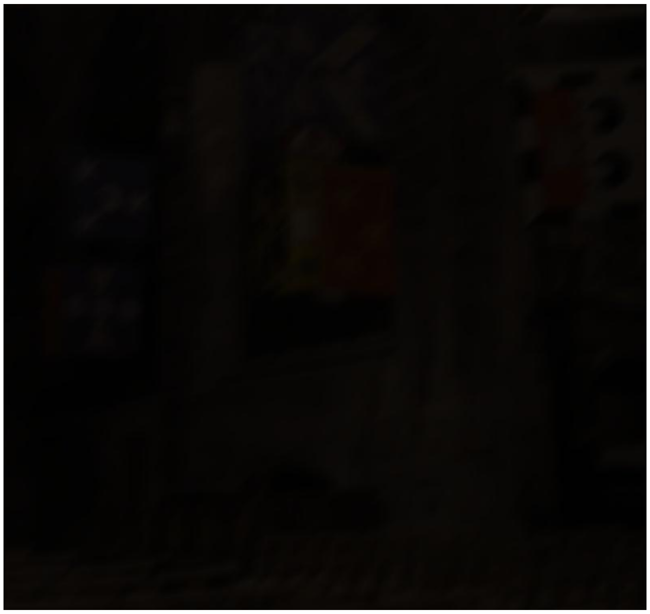
Test image after Weiner Filtering with k = 0.1

PSNR:23.148784387462406 dB
SSIM:0.12899297868054202



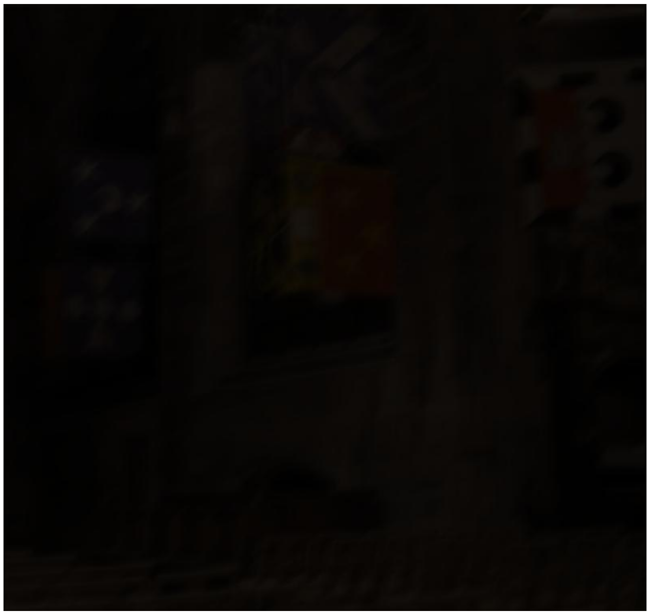
Test image after Weiner Filtering with k = 5

PSNR:13.556953153524585 dB
SSIM:0.014628983425784947



Test image after Weiner Filtering with k = 8

PSNR:13.038407792002904 dB
SSIM:0.006734134067439941



Test image after Weiner Filtering with k = 15

PSNR:12.611057048090903 dB
SSIM:0.0023642272982815336

4.4. Constrained LS Filtering

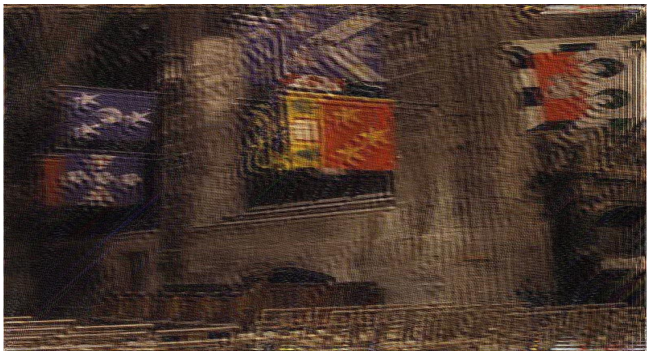


Image after applying LS filtering with gamma = 0

PSNR:17.920233880580266 dB
SSIM:0.008234314887168165

4.5. Own Image

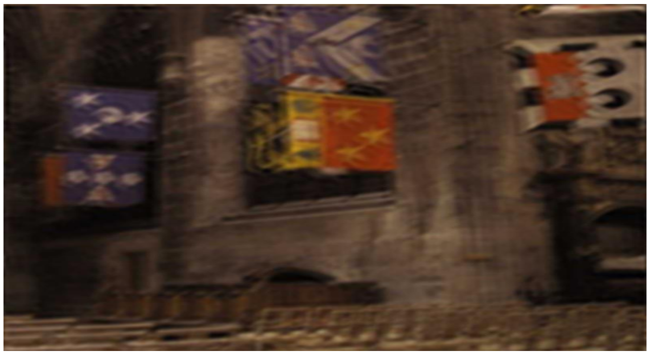


Image after applying LS filtering with gamma = 5

PSNR:24.42661193863731 dB
SSIM:00.19684344630868653

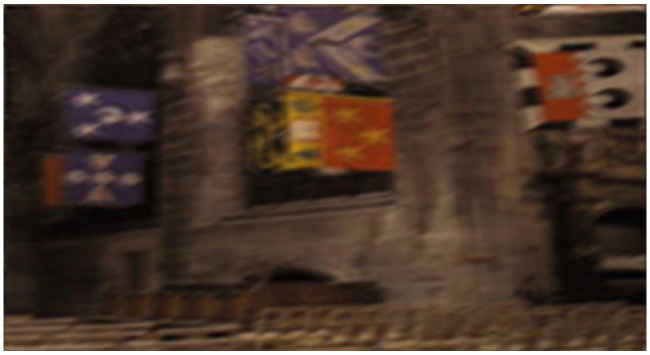


Image after applying LS filtering with gamma = 15

PSNR:24.43738047045613 dB
SSIM:0.18830657461332967

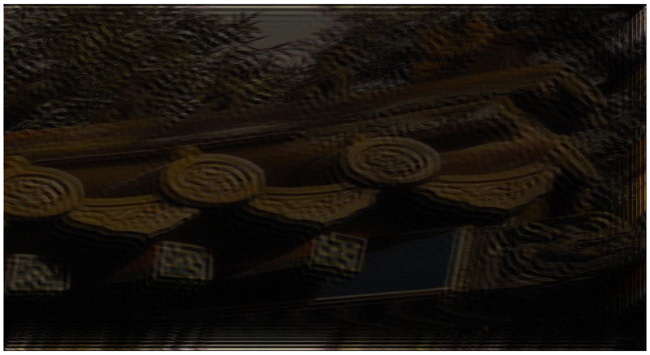
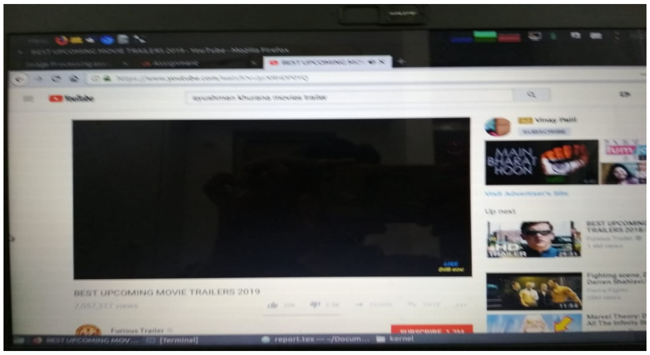


Image after applying LS filtering with gamma = 30

PSNR:24.38289276706215 dB
SSIM:0.17594902052330422



Degraded Image

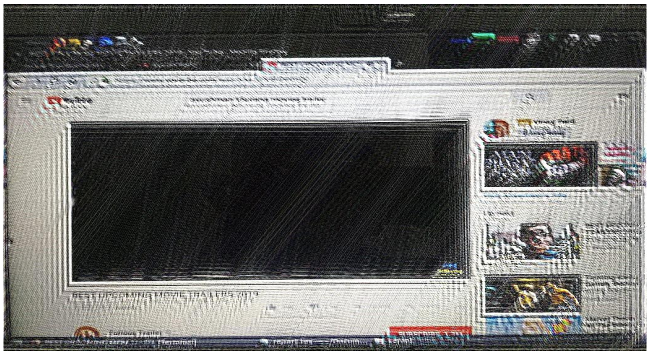


Image after applying Inverse filtering

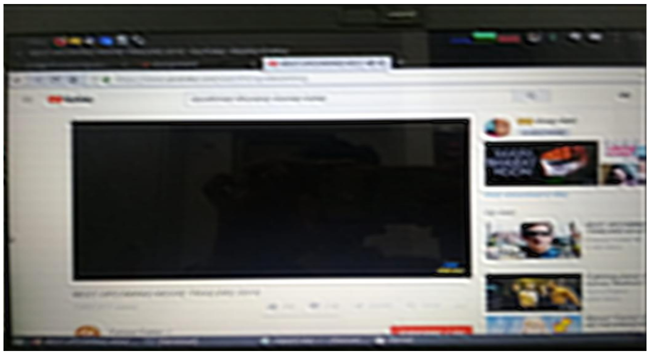


Image after applying LS filtering with gamma = 10

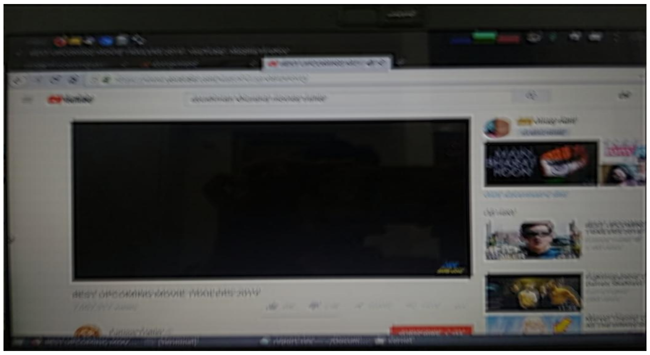


Image after applying Weiner filtering with k = 0.5

5. IMAGE 1 AND KERNEL 1

Technique	PSNR(dB)	SSIM
Inverse Filtering	17.920233880580266	0.008234314887168165
Radial Filtering	17.920233880580266	0.008234314887168165
WEiner Filtering(k=0.1)	23.148784387462406	0.12899297868054202
WEiner Filtering(k=5)	13.556953153524585	0.014628983425784947
WEiner Filtering(k=8)	13.038407792002904	0.006734134067439941
WEiner Filtering(k=15)	12.611057048090903	0.0023642272982815336
LS Filtering(gamma = 0)	17.920233880580266	0.008234314887168165
LS Filtering(gamma = 5)	24.42661193863731	0.19684344630868653
LS Filtering(gamma = 15)	24.43738047045613	0.18830657461332967
LS Filtering(gamma = 30)	24.38289276706215	0.17594902052330422

6. IMAGE 2 AND KERNEL 2

Technique	PSNR(dB)	SSIM
Inverse Filtering	15.743600177961191	0.01806763382100661
Radial Filtering	15.743600177961191	0.01806763382100661
WEiner Filtering(k=0.1)	17.870996924435516	0.11471044241070671
WEiner Filtering(k=5)	9.92043654165898	0.026220985543765552
WEiner Filtering(k=8)	9.403203138009308	0.013683632943321195
WEiner Filtering(k=15)	8.970344208679567	0.0054784068913342225
LS Filtering(gamma = 0)	15.743600177961191	0.01806763382100661
LS Filtering(gamma = 5)	18.499401965077773	0.17800703206835747
LS Filtering(gamma = 15)	18.862352697811325	0.1969149651622446
LS Filtering(gamma = 30)	19.048974999571342	0.2013364186005355

7. IMAGE 3 AND KERNEL 3

Technique	PSNR(dB)	SSIM
Inverse Filtering	21.20097119935902	0.007630150027224443
Radial Filtering	21.20097119935902	0.007630150027224443
WEiner Filtering(k=0.1)	21.338794848945035	0.031498087762106323
WEiner Filtering(k=5)	10.872693095819962	0.011393571537853303
WEiner Filtering(k=8)	10.335695115933085	0.009885694513261946
WEiner Filtering(k=15)	9.894650159302431	0.002362223152195035
LS Filtering(gamma = 0)	21.20097119935902	0.007630150027224443
LS Filtering(gamma = 5)	22.456208024404198	0.03928754782333967
LS Filtering(gamma = 15)	22.769949599027072	0.06380004263981759
LS Filtering(gamma = 30)	22.995828804296377	0.08294978860243511

8. IMAGE 4 AND BLUR KERNEL 4

Technique	PSNR(dB)	SSIM
Inverse Filtering	11.624179804769792	0.01806763382100661
Radial Filtering	11.624179804769792	0.012311038805806343
WEiner Filtering(k=0.1)	17.393370370384392	0.03811339259355565
WEiner Filtering(k=5)	10.30338149892981	0.018064667816945604
WEiner Filtering(k=8)	9.805246795009085	0.014313951901248873
WEiner Filtering(k=15)	9.379381457812695	0..008265350751804708
LS Filtering(gamma = 0)	11.624179804769792	0.012311038805806343
LS Filtering(gamma = 5)	17.620551846304174	0.05629157888061678
LS Filtering(gamma = 15)	17.834248734468147	0.06413095520355722
LS Filtering(gamma = 30)	18.003170165298194	0.07069987527622389

9. DISCUSSION AND CONCLUSION

Entire Implementation is carried out under the assumption that the degraded images obtained are the result of convolution of the spatially invariant kernel i.e., image is uniformly blurred by the same kernel. But the images taken are not spatially invariant. So the restored images are not likely to be as good as the ground truth, but the performance of several image restoration techniques can be studied.

Inverse Filtering performs better only when there is no Noise or low Noise and also when the degradation is a result of the spatially invariant kernel. It is highly unsuitable for restoration in other cases.

Radial Filtering is same as Inverse Filtering Except that it is passed through a Low pass filter after the Inverse Filtering. So it performs better than inverse filtering if the noise is low and kernel is spatially invariant. But in other cases it too fails miserably

Weiner Filtering is better that Radial and inverse Filters because it takes into account the noise distributed across the image and tries to preserve it so that the noise is not amplified. In this tool approximation of Weiner Filter is used by getting the ratio from the user which may cause performance loss. By looking at the tables above we can see that as the k value increases the Filtering becomes worse. So Weiner filtering gives best results when K is low (k¡ 1).

LS filtering also a good restoration technique. As the Implementation in this tool is a loose approximation of the actual filtering technique it may not give best result. The the gamma value is taken by the user but in actual technique it found in a iterative process and the optimum gamma would give good results. Also looking at the table one can say that as the gamma value increases the PSNR and SSIM increase. LS filtering works well for higher value of gamma but if these metircs are not taken into account and a subjective analysis of the output is done we can observe that the restored image is getting more and more blurred as the gamma is increased.

An image which is taken from my mobile is assumed to be degraded by the kernel 1. I tried to restore it by applying several techniques. Inverse filter and radial filter did not give satisfying results this might be because of the fact that the noise is not low and also the kernel is not spatially invariant. Weiner worked to some extent but it could not give exact restored image as the ground truth does not exist for this image it is difficult to say how good the restoration was. LS filtering gave the best results of all but as the gamma is increased more of blurring is happening.

9.0.1. Learning

Image restoration is not an exact maths as there are several factors which effect the performance of an restoration technique. The Techniques implemented here are functional under a lot of constrains . Machine Learning will be of great use because if the kernel and noise are estimated properly restoration is easy.

10. GIT HUB LINK

”https://github.com/M-ark17/Image-processing/tree/master/Assignment2”

11. REFERENCES

· **FFT** ”https://stackoverflow.com/questions/19739503/dft-matrix-in-python”

· **padding** ”https://docs.scipy.org/doc/numpy/reference/generated/numpy.pad.html”

· **Metrics** ”https://www.pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python/”

· **Data Types** ”https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.dtypes.html”

· **MSE** ”https://stackoverflow.com/questions/20271479/what-does-it-mean-to-get-the-mse-mean-error-squared-for-2-images”

· **SSIM** ”http://scikit-image.org/docs/dev/api/skimage.measure.html”

· **AND operator** ”https://stackoverflow.com/questions/609972/how-to-use-boolean-and-in-python ”

· **Numpy product of all elements** ”https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.prod.html”

· **PSNR** ”https://en.wikipedia.org/wiki/Peak$_{s}ignal - to - noise_{r}atio''$

12. APPENDIX

12.1. Code using inbuilt FFT and IFFT functions

```
#!/usr/bin/python

import sys # import libraries needed
import PyQt4
import math
import numpy as np
import cv2 as cv
from scipy.signal import convolve2d
from skimage.measure import compare_ssim
# import PythonQwt as qwt
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("Image␣Restoration␣Tool") # 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.__img_height = None # height of the Image
      self.__img_width = None # widht 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.lbl_ker_img = QtGui.QLabel(self) # create a Qlabel object to display
         ↪ kernel
      self.lbl_ker = QtGui.QLabel(self) #create a Qlabel object to display title for
         ↪  kernel
      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 displat output
         ↪ image
      self.lbl_s3 = QtGui.QLabel(self) # create a Qlabel object to display text for
         ↪ text editor
      self.lbl_s4 = QtGui.QLabel(self) # create a Qlabel object to display text for
         ↪ text editor
      self.lbl_s5 = QtGui.QLabel(self) # create a Qlabel object to display text for
         ↪ text editor
      self.lbl_s6 = QtGui.QLabel(self) # create a Qlabel object to display metrics
         ↪ ssim and mse
      self.e2 = QtGui.QLineEdit(self) # create a QLineEdit object to display scroll
         ↪ title
      self.e3 = QtGui.QLineEdit(self) # create a QLineEdit object to display scroll
         ↪ title
      self.e4 = 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("Upload␣Kernel␣",self)
    btn1.clicked.connect(self.file_open_kernel) # go to file_open_kernel method
        ↪ when clicked on Upload Kernel
    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("Inverse␣Filter",self)
    btn2.clicked.connect(lambda: self.inverse_fliter(-1)) # go to inverse_fliter
        ↪ method when clicked on Inverse Filter 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("Get␣blur␣image",self)
    btn3.clicked.connect(self.inv_inbuilt) # go to inv_inbuilt method when clicked
        ↪  on Get blur image 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("Radial␣Filtering",self)
    btn4.clicked.connect(self.radial_filter_threshold) # 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("Weiner␣Filtering",self)
    btn5.clicked.connect(self.weiner_filtering) # go to weiner_filtering 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("LS␣Filtering",self)
    btn7.clicked.connect(self.ls_filtering_gamma) # go to ls_filtering_gamma
        ↪ method when clicked on LS Filtering button
    btn7.resize(200,40) # resize the button to the required size
    btn7.move(500,350 ) # reposition the button at the required position
    btn8 = QtGui.QPushButton("Calculate␣Metrics␣",self)
    btn8.clicked.connect(self.metrics) # go to metrics method when clicked on
        ↪ Calculate Metrics button
    btn8.resize(200,40) # resize the button to the required size
    btn8.move(500,400 ) # 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)).astype(np.float) #
            ↪ upload the image from the dialog box using imread in opencv library
        # get image properties.
        self.__img_b,self.__img_g,self.__img_r = cv.split(self.__ip_img)
        self.__img_height,self.__img_width = self.__img_r.shape
        # Image.merge("RGB",(imr,img,imb))
        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(200,140) # 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(50,50) # 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
        else: #if the image is not uploaded then
            print("Could␣not␣upload␣Image") # print status to the terminal or IDE

    def file_open_kernel(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.__kernel = (cv.imread(str(name),cv.IMREAD_GRAYSCALE)).astype(np.float) #
            ↪ upload the image from the dialog box using imread in opencv library
        self.__kernel = np.true_divide(self.__kernel,np.sum(self.__kernel),dtype=np.
            ↪ float)
        # get image properties.
        self.__kernel_height,self.__kernel_width = self.__kernel.shape
```

```
      # Image.merge("RGB",(imr,img,imb))
      if upld_img.load(name): # if the image is uploaded properly then upld_img.load
         ↪  will be true
         self.lbl_ker.clear() # clear the past content in label if any is present
         self.lbl_ker.setText("kernel") # Set title for the input image to display
         self.lbl_ker.move(225,10) # position the title
         self.lbl_ker.show() # show the title
         pixmap = QtGui.QPixmap(upld_img) #convert the opencv image to pixmap to
            ↪ display it on GUI
         self.__pixmap = pixmap.scaled(100, 125, QtCore.Qt.KeepAspectRatio) # scale
            ↪ the pixmap to display it on GUI keep the Aspect Ratio of the original
            ↪  image
         self.lbl_ker_img.clear() # clear the past content in label if any is
            ↪ present
         self.lbl_ker_img.resize(100,125) # set the size of the input pixmap to
            ↪ 100*125
         self.lbl_ker_img.move(200,25) # position the input pixmap
         self.lbl_ker_img.setSizePolicy(QtGui.QSizePolicy.Minimum, QtGui.QSizePolicy
            ↪ .Minimum)
         self.lbl_ker_img.setScaledContents(False)
         self.lbl_ker_img.setPixmap(self.__pixmap) # set the pixmap to the label
         self.lbl_ker_img.show()# show the pixmap as image
         print("Selected␣Kernel␣uploaded") #print status to the terminal or IDE
      else: #if the image is not uploaded then
         print("Could␣not␣upload␣kernel") # print status to the terminal or IDE

   def FFT_matrix(self,N,sign=1): #function to compute FFT matrix
      i, j = np.meshgrid(np.arange(N), np.arange(N)) #create index matix
      omega = np.exp( sign * -2 * np.pi * 1J / N ) #compute the twiddle factor
      W = np.power( omega, i * j ) / np.sqrt(N) #multiply it with the sum of index
      return W #return the twiddle factor matrix

   def DFT(self,img,ker=0):# this method performs the Discreet fourier Transform
      if(ker == 1): #if given image is kernel
         # rows = self.FFT_matrix(img.shape[0]) #to do fft to rows
         # cols = self.FFT_matrix(img.shape[1]) #to do fft to columns
         # img = rows.dot(img).dot(cols) #first perform fft to rows then for columns
         img = np.fft.fft2(img)
         img = np.fft.fftshift(img) #since the dft is not centered shift it
         # cv.imwrite("DFT.jpg",np.absolute(img))
         return img #return the DFT
      else: #if other images are given they are color images
         b,g,r = cv.split(img) #split the image to get r,g,b channels
         # rows = self.FFT_matrix(self.__img_height) #get fft matrix for rows
         # cols = self.FFT_matrix(self.__img_width) #get fft matrix for columns
         # b = rows.dot(b).dot(cols) # do fft to blue channel
         b = np.fft.fft2(b)
         b = np.fft.fftshift(b) #since the dft is not centered shift it
         # g = rows.dot(g).dot(cols) # do fft to green channel
         g = np.fft.fft2(g)
         g = np.fft.fftshift(g) #since the dft is not centered shift it
         # r = rows.dot(r).dot(cols) # do fft to red channel
         r = np.fft.fft2(r)
         r = np.fft.fftshift(r) #since the dft is not centered shift it
         # cv.imwrite("DFT.jpg",img)
         return b,g,r

   def IDFT(self,img,ker=0):# this method performs the Inverse Discreet fourier
       ↪ Transform
      # rows = self.FFT_matrix(img.shape[0],-1)#get ifft matrix for rows
      # cols = self.FFT_matrix(img.shape[1],-1)#get ifft matrix for columns
      # img = rows*colrows.dot(img).dot(cols) #first perform fft to rows then for
          ↪ columns
      img = np.fft.ifft2(img)
      img = np.fft.ifftshift(img) #since the idft is not centered shift it
      # cv.imwrite("IDFT.jpg",np.absolute(img))
      return img

   def padder(self,img,truth=0): #to padd every image
      if(truth == 1): #if the given image is ground truth just resize it
         rem_row = self.__img_height-img.shape[0] #count the number to delete in
             ↪ image
         rem_col = self.__img_width -img.shape[1] #count the number to delete in
             ↪ image
         padd_img = np.delete(img, abs(rem_row), 0) # delete from image
         padd_img = np.delete(padd_img, abs(rem_col), 1) # delete from image
      else: #if the given image is not groung truth
         rw_add = np.ceil((self.__img_height-img.shape[0])/2) #no of rows to add
             ↪ above and below
         rw_add = rw_add.astype(int) #convert float to int
         col_add = np.ceil((self.__img_width-img.shape[1])/2) #no of columns to add
             ↪ above and below
         col_add = col_add.astype(int) #convert float to int
         # if(rw_add > 0 & col_add> 0 ):
         padd_img = np.append(np.zeros((rw_add,img.shape[1])), img, axis=0)#padd
             ↪ with zeros
         padd_img = np.append(padd_img,np.zeros((rw_add,padd_img.shape[1])), axis=0)
             ↪ #padd with zeros
         padd_img = np.append(np.zeros((padd_img.shape[0],col_add)), padd_img,axis
             ↪ =1)#padd with zeros
         padd_img = np.append(padd_img,np.zeros((padd_img.shape[0],col_add)),axis=1)
             ↪ #padd with zeros
         rem_row = self.__img_height-padd_img.shape[0] #count how many rows to
             ↪ remove
         rem_col = self.__img_width -padd_img.shape[1] #count how many rows to
             ↪ remove
         if(rem_row>0):
            self.__ip_img = np.delete(self.__ip_img, rem_row, 0) #delete the rows
                ↪ from the image
            self.__img_height -= rem_row #set the image size accordingly

         if(rem_col>0):
            self.__ip_img = np.delete(self.__ip_img, rem_col, 1) #delete the columns
                ↪  from the image
            self.__img_width -= rem_col #set the image size accordingly
      return padd_img #return the padded image

   def inverse_fliter(self,sigma = -1): # method to do inverse filtering
```

```
        padd_kernel = self.padder(self.__kernel) # get the padded image
        H = self.DFT(padd_kernel,1) # find the DFT of the padded image
        string = "␣" # string to show title text
        F = np.ones_like(H) #get a array with all elements as one
        if(sigma != -1): # if the method is called only for radial inverse filtering
            for index, x in np.ndenumerate(F): #get each index for F
                if (np.sqrt(index[0]*index[0]+index[1]*index[1])>sigma): #if the index
                    ↪ is in range keep it as 1
                    F[index[0],index[0]] = 1
                else: # since it is not in range make it 0
                    F[index[0],index[0]] = 0
        B,G,R = self.DFT(np.true_divide(self.__ip_img,255.0)) #get the DFT the
            ↪ normalised image
        INV_B = (B/H)*F # do the radial or only inverse filtering depending on sigma
        INV_G = (G/H)*F # do the radial or only inverse filtering depending on sigma
        INV_R = (R/H)*F # do the radial or only inverse filtering depending on sigma
        ib = self.IDFT(INV_B)*255.0 #get back to normal range and perform IDFT
        ig = self.IDFT(INV_G)*255.0 #get back to normal range and perform IDFT
        ir = self.IDFT(INV_R)*255.0 #get back to normal range and perform IDFT
        self.__img_b = (np.absolute(ib)).astype(self.__ip_img.dtype) #put them in
            ↪ global variables
        self.__img_g = (np.absolute(ig)).astype(self.__ip_img.dtype) #put them in
            ↪ global variables
        self.__img_r = (np.absolute(ir)).astype(self.__ip_img.dtype) #put them in
            ↪ global variables
        cv.imwrite("temp.jpg",cv.merge((self.__img_b,self.__img_g, self.__img_r))) #
            ↪ write it to the temp image
        self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
            ↪ IMREAD_COLOR)) # read the temp image to show in GUI
        if(sigma != -1): # for title and display text
            string += "Radial␣Inverse␣Filter␣Applied␣with␣cutoff␣=␣"+str(sigma)
        else:
            string = "Inverse␣Filter␣Applied"
        self.disp(string) # display image
        print(string) #print status to terminal

    def inv_inbuilt(self): #to get the blurred image from kernel
        motion_blr = cv.filter2D(self.__ip_img,-1,np.divide(self.__kernel,np.sum(self.
            ↪ __kernel).astype(self.__ip_img.dtype))) #perform convolution
        cv.imwrite("temp.jpg",motion_blr) # write to temp image
        self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
            ↪ IMREAD_COLOR)) #read from temp to show
        self.disp("Blurred␣Image") # display it in GUI
        print("Blurring␣using␣kernel") # Print status to terminal or IDE

    def radial_filter_threshold(self):#to get the cutoff from user
        self.lbl_s3.resize(500,50)#label to display title for output image
        self.lbl_s3.setText("Please␣Enter␣an␣value␣for␣Threshold")#title text
        self.lbl_s3.move(100,590) #positioning
        self.lbl_s3.show() #display title
        self.e2.setValidator(QDoubleValidator())#text box setting to allow only
            ↪ integer values
        self.e2.move(500,600) #positioning
```

```
      radial_threshold = QPushButton('OK', self) #button to click ok to start
         ↪ operaion on the input
      radial_threshold.resize(50,30) #resize the button
      radial_threshold.move(610, 600) #positioning
      self.lbl_s5.clear()
      self.lbl_s4.clear()
      self.e4.clear()
      self.e3.clear()
      radial_threshold.show() #display button
      self.e2.show() #display text box
      radial_threshold.clicked.connect(lambda: self.inverse_fliter(np.float(self.e2.
         ↪ text()))) #call inverse_fliter when clicked

   def weiner_filtering(self): #to get the k image from user
      self.lbl_s4.resize(500,50)#label to display title for output image
      self.lbl_s4.setText("Please␣Enter␣an␣Integer␣value␣of␣K")#title text
      self.lbl_s4.move(100,590) #positioning
      self.lbl_s4.show() #display title
      self.e3.setValidator(QDoubleValidator())#text box setting to allow only
         ↪ integer values
      self.e3.move(500,600) #positioning
      weiner_k = QPushButton('OK', self) #button to click ok to start operation on
         ↪ the input
      weiner_k.resize(50,30) #resize the button
      weiner_k.move(610, 600) #positioning
      weiner_k.show() #display button
      self.lbl_s5.clear()
      self.lbl_s3.clear()
      self.e4.clear()
      self.e2.clear()
      self.e3.show() #display text box
      weiner_k.clicked.connect(lambda: self.weiner(np.float(self.e3.text()))) #call
         ↪ weiner when clicked

   def weiner(self,k): #to perform weiner filtering
      padd_kernel = self.padder(self.__kernel) # get the padded image
      H = self.DFT(padd_kernel,1)# find the DFT of the padded image
      B,G,R = self.DFT(np.true_divide(self.__ip_img,255.0))# find the DFT of the
         ↪ image
      INV_B = np.multiply(B,np.divide(np.power(np.absolute(H),2),(np.multiply(H,np.
         ↪ power(np.absolute(H),2)+k)))) #perform weiner filter and get the channel
      INV_G = np.multiply(G,np.divide(np.power(np.absolute(H),2),(np.multiply(H,np.
         ↪ power(np.absolute(H),2)+k)))) #perform weiner filter and get the channel
      INV_R = np.multiply(R,np.divide(np.power(np.absolute(H),2),(np.multiply(H,np.
         ↪ power(np.absolute(H),2)+k)))) #perform weiner filter and get the channel

      ib = self.IDFT(INV_B)*255.0 #perform IDFT
      ig = self.IDFT(INV_G)*255.0 #perform IDFT
      ir = self.IDFT(INV_R)*255.0 #perform IDFT
      self.__img_b = (np.absolute(ib)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      self.__img_g = (np.absolute(ig)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      self.__img_r = (np.absolute(ir)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      cv.imwrite("temp.jpg",cv.merge((self.__img_b,self.__img_g, self.__img_r))) #
         ↪ write it to temp image
      self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
         ↪ IMREAD_COLOR))# read the temp image to show in GUI
      self.disp("Weiner␣Filter␣Applied␣for␣k␣=␣"+str(k)) #display it to GUI
      print("Weiner␣Filter␣Applied␣for␣k␣=␣"+str(k)) #print status to terminal

   def ls_filtering_gamma(self): # to get gamma values
      self.lbl_s5.resize(500,50)#label to display title for output image
      self.lbl_s5.setText("Please␣Enter␣an␣Integer␣value␣of␣gamma")#title text
      self.lbl_s5.move(100,590) #positioning
      self.lbl_s5.show() #display title
      self.e4.setValidator(QIntValidator())#text box setting to allow only integer
         ↪ values
      self.e4.move(500,600) #positioning
      gamma = QPushButton('OK', self) #button to click ok to start operaion on the
         ↪ input
      gamma.resize(50,30) #resize the button
      gamma.move(610, 600) #positioning
      gamma.show() #display button
      self.lbl_s3.clear()
      self.lbl_s4.clear()
      self.e2.clear()
      self.e3.clear()
      self.e4.show() #display text box
      gamma.clicked.connect(lambda: self.ls_filter(int(self.e4.text()))) #call
         ↪ blur_img when clicked

      print("gamma␣value␣given␣is␣=␣"+ str(self.e4.text())) # Print status to
         ↪ terminal or IDE
   def ls_filter(self,gamma=1): # to perform LS filtering
      p = np.array([[0,-1,0],[-1,4,-1],[0,-1,0]]) # blur kernel
      padd_p = self.padder(p) # get the padded image
      P = self.DFT(padd_p,1) # find the DFT of the padded image
      h = self.padder(self.__kernel)# get the padded image
      H = self.DFT(h,1) # find the DFT of the kernel image
      B,G,R = self.DFT(np.true_divide(self.__ip_img,255.0)) #get the DFT the
         ↪ normalised image
      filter = np.divide(np.conj(H),(np.power(np.absolute(H),2)+gamma*np.power(np.
         ↪ absolute(P),2))) #get LS filter
      R_trans = np.multiply(filter,R)# do the LS filtering depending on gamma
      G_trans = np.multiply(filter,G)# do the LS filtering depending on gamma
      B_trans = np.multiply(filter,B)# do the LS filtering depending on gamma
      ib = self.IDFT(B_trans)*255.0#get back to normal range and perform IDFT
      ig = self.IDFT(G_trans)*255.0#get back to normal range and perform IDFT
      ir = self.IDFT(R_trans)*255.0#get back to normal range and perform IDFT
      self.__img_b = (np.absolute(ib)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      self.__img_g = (np.absolute(ig)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      self.__img_r = (np.absolute(ir)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
```

```
    cv.imwrite("temp.jpg",cv.merge((self.__img_b,self.__img_g, self.__img_r))) #
        ↪ write it to the temp image
    self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
        ↪ IMREAD_COLOR)) # read the temp image to show in GUI
    self.disp("LS␣Filter␣Applied␣for␣gamma␣=␣"+str(gamma))# display image
    print("LS␣Filter␣Appliedfor␣gamma␣=␣"+str(gamma))#print status to terminal

def metrics(self): #to undo the last change done on the image
    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
    grd_truth = (cv.imread(str(name),cv.IMREAD_COLOR)).astype(np.float) #get the
        ↪ image name from user and read it
    restored_img = cv.merge((self.__img_b,self.__img_g, self.__img_r)).astype(np.
        ↪ float) #merge to get restored image
    grd_truth_resize = self.padder(grd_truth,1) #resize the ground truth to our
        ↪ size
    difference_squared = (grd_truth_resize.astype(np.float) -restored_img) ** 2 #
        ↪ get the square of the difference
    summ_diff_square = np.sum(difference_squared) #sum the difference
    pixels_size = np.prod(grd_truth_resize.shape) #get the pixel size of the image
    mse = summ_diff_square / pixels_size #find out the mean square error
    psnr = 20.0*np.log10(255/np.sqrt(mse)) #calculate PSNR
    ssim = compare_ssim(grd_truth_resize.astype(np.float), restored_img,
        ↪ multichannel=True) #get the SSIM
    self.lbl_s6.clear() # clear the past content in label if any is present
    self.lbl_s6.resize(200,70)
    self.lbl_s6.setText("PSNR␣="+str(psnr)+"dB\n"+’SSIM␣=␣’+str(ssim)) # Set PSNR
        ↪ and SSIM
    self.lbl_s6.move(500,440) # position the text
    self.lbl_s6.show()
    print(’ssim=␣’+str(ssim),’psnr␣=␣’+str(psnr)+"dB") #print status to terminal

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_b,self.__img_g, self.__img_r])#merge intensity
        ↪ with the hue and saturation
    itos = cv.cvtColor(itos, cv.COLOR_BGR2RGB)#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␣at␣"+str(name)) # 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,fft=0,img = np.empty_like([256,256])): # this method
      ↪ is used to display the transformed image to GUI
      if (fft == 0): #whether to clear some labels or not is decided by this flag
         ↪ variable
         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
         if (flag == 0 ):
            img_pix1 = cv.merge((self.__img_b,self.__img_g, self.__img_r)) #merge
               ↪ the v with h and s using cv.merge
            img_pix1 = cv.cvtColor(img_pix1, cv.COLOR_BGR2RGB)
            pix_img = QtGui.QPixmap(QtGui.QImage(img_pix1,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
      else:
         pix_img = QtGui.QPixmap(QtGui.QImage(img,self.__img_width, self.
            ↪ __img_height,3*self.__img_width,QtGui.QImage.Format_Indexed8))
      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(930,0) #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()
```

12.2. Code without using inbuilt FFT and IFFT functions

```
#!/usr/bin/python

import sys # import libraries needed
import PyQt4
import math
import numpy as np
import cv2 as cv
from scipy.signal import convolve2d
from skimage.measure import compare_ssim
# import PythonQwt as qwt
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("Image␣Restoration␣Tool") # 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.__img_height = None # height of the Image
      self.__img_width = None # widht 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.lbl_ker_img = QtGui.QLabel(self) # create a Qlabel object to display
         ↪ kernel
      self.lbl_ker = QtGui.QLabel(self) #create a Qlabel object to display title for
         ↪  kernel
      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 displat output
         ↪ image
      self.lbl_s3 = QtGui.QLabel(self) # create a Qlabel object to display text for
         ↪ text editor
      self.lbl_s4 = QtGui.QLabel(self) # create a Qlabel object to display text for
         ↪ text editor
      self.lbl_s5 = QtGui.QLabel(self) # create a Qlabel object to display text for
         ↪ text editor
      self.lbl_s6 = QtGui.QLabel(self) # create a Qlabel object to display metrics
         ↪ ssim and mse
      self.e2 = QtGui.QLineEdit(self) # create a QLineEdit object to display scroll
         ↪ title
      self.e3 = QtGui.QLineEdit(self) # create a QLineEdit object to display scroll
         ↪ title
      self.e4 = 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("Upload␣Kernel␣",self)
      btn1.clicked.connect(self.file_open_kernel) # go to file_open_kernel method
         ↪ when clicked on Upload Kernel
      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("Inverse␣Filter",self)
      btn2.clicked.connect(lambda: self.inverse_fliter(-1)) # go to inverse_fliter
         ↪ method when clicked on Inverse Filter 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("Get␣blur␣image",self)
        btn3.clicked.connect(self.inv_inbuilt) # go to inv_inbuilt method when clicked
          ↪  on Get blur image 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("Radial␣Filtering",self)
        btn4.clicked.connect(self.radial_filter_threshold) # 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("Weiner␣Filtering",self)
        btn5.clicked.connect(self.weiner_filtering) # go to weiner_filtering 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("LS␣Filtering",self)
        btn7.clicked.connect(self.ls_filtering_gamma) # go to ls_filtering_gamma
          ↪ method when clicked on LS Filtering button
        btn7.resize(200,40) # resize the button to the required size
        btn7.move(500,350 ) # reposition the button at the required position
        btn8 = QtGui.QPushButton("Calculate␣Metrics␣",self)
        btn8.clicked.connect(self.metrics) # go to metrics method when clicked on
          ↪ Calculate Metrics button
        btn8.resize(200,40) # resize the button to the required size
        btn8.move(500,400 ) # 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)).astype(np.float) #
          ↪ upload the image from the dialog box using imread in opencv library
        # get image properties.
        self.__img_b,self.__img_g,self.__img_r = cv.split(self.__ip_img)
        self.__img_height,self.__img_width = self.__img_r.shape
```

```
      # Image.merge("RGB",(imr,img,imb))
      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(200,140) # 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(50,50) # 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
      else: #if the image is not uploaded then
         print("Could␣not␣upload␣Image") # print status to the terminal or IDE

   def file_open_kernel(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.__kernel = (cv.imread(str(name),cv.IMREAD_GRAYSCALE)).astype(np.float) #
         ↪ upload the image from the dialog box using imread in opencv library
      # self.__kernel = np.true_divide(self.__kernel,np.sum(self.__kernel),dtype=np.
         ↪ float)
      # get image properties.
      self.__kernel_height,self.__kernel_width = self.__kernel.shape
      # Image.merge("RGB",(imr,img,imb))
      if upld_img.load(name): # if the image is uploaded properly then upld_img.load
         ↪  will be true
         self.lbl_ker.clear() # clear the past content in label if any is present
         self.lbl_ker.setText("kernel") # Set title for the input image to display
         self.lbl_ker.move(225,10) # position the title
         self.lbl_ker.show() # show the title
         pixmap = QtGui.QPixmap(upld_img) #convert the opencv image to pixmap to
            ↪ display it on GUI
         self.__pixmap = pixmap.scaled(100, 125, QtCore.Qt.KeepAspectRatio) # scale
            ↪ the pixmap to display it on GUI keep the Aspect Ratio of the original
            ↪  image
         self.lbl_ker_img.clear() # clear the past content in label if any is
            ↪ present
         self.lbl_ker_img.resize(100,125) # set the size of the input pixmap to
            ↪ 100*125
         self.lbl_ker_img.move(200,25) # position the input pixmap
```

```
            self.lbl_ker_img.setSizePolicy(QtGui.QSizePolicy.Minimum, QtGui.QSizePolicy
                ↪ .Minimum)
            self.lbl_ker_img.setScaledContents(False)
            self.lbl_ker_img.setPixmap(self.__pixmap) # set the pixmap to the label
            self.lbl_ker_img.show()# show the pixmap as image
            print("Selected␣Kernel␣uploaded") #print status to the terminal or IDE
        else: #if the image is not uploaded then
            print("Could␣not␣upload␣kernel") # print status to the terminal or IDE

    def FFT_matrix(self,N,sign=1): #function to compute FFT matrix
        i, j = np.meshgrid(np.arange(N), np.arange(N)) #create index matix
        omega = np.exp( sign * -2 * np.pi * 1J / N ) #compute the twiddle factor
        W = np.power( omega, i * j ) / np.sqrt(N) #multiply it with the sum of index
        return W #return the twiddle factor matrix

    def DFT(self,img,ker=0):# this method performs the Discreet fourier Transform
        if(ker == 1): #if given image is kernel
            rows = self.FFT_matrix(img.shape[0]) #to do fft to rows
            cols = self.FFT_matrix(img.shape[1]) #to do fft to columns
            img = rows.dot(img).dot(cols) #first perform fft to rows then for columns
            img = np.fft.fftshift(img) #since the dft is not centered shift it
            # cv.imwrite("DFT.jpg",np.absolute(img))
            return img #return the DFT
        else: #if other images are given they are color images
            b,g,r = cv.split(img) #split the image to get r,g,b channels
            rows = self.FFT_matrix(self.__img_height) #get fft matrix for rows
            cols = self.FFT_matrix(self.__img_width) #get fft matrix for columns
            b = rows.dot(b).dot(cols) # do fft to blue channel
            b = np.fft.fftshift(b) #since the dft is not centered shift it
            g = rows.dot(g).dot(cols) # do fft to green channel
            g = np.fft.fftshift(g) #since the dft is not centered shift it
            r = rows.dot(r).dot(cols) # do fft to red channel
            r = np.fft.fftshift(r) #since the dft is not centered shift it
            # cv.imwrite("DFT.jpg",img)
            return b,g,r

    def IDFT(self,img,ker=0):# this method performs the Inverse Discreet fourier
        ↪ Transform
        rows = self.FFT_matrix(img.shape[0],-1)#get ifft matrix for rows
        cols = self.FFT_matrix(img.shape[1],-1)#get ifft matrix for columns
        img = rows.dot(img).dot(cols) #first perform fft to rows then for columns
        img = np.fft.ifftshift(img) #since the idft is not centered shift it
        # cv.imwrite("IDFT.jpg",np.absolute(img))
        return img

    def padder(self,img,truth=0): #to padd every image
        if(truth == 1): #if the given image is ground truth just resize it
            rem_row = self.__img_height-img.shape[0] #count the number to delete in
                ↪ image
            rem_col = self.__img_width -img.shape[1] #count the number to delete in
                ↪ image
            padd_img = np.delete(img, abs(rem_row), 0) # delete from image
            padd_img = np.delete(padd_img, abs(rem_col), 1) # delete from image
        else: #if the given image is not groung truth
```

```
         rw_add = np.ceil((self.__img_height-img.shape[0])/2) #no of rows to add
            ↪ above and below
         rw_add = rw_add.astype(int) #convert float to int
         col_add = np.ceil((self.__img_width-img.shape[1])/2) #no of columns to add
            ↪ above and below
         col_add = col_add.astype(int) #convert float to int
         # if(rw_add > 0 & col_add> 0 ):
         padd_img = np.append(np.zeros((rw_add,img.shape[1])), img, axis=0)#padd
            ↪ with zeros
         padd_img = np.append(padd_img,np.zeros((rw_add,padd_img.shape[1])), axis=0)
            ↪ #padd with zeros
         padd_img = np.append(np.zeros((padd_img.shape[0],col_add)), padd_img,axis
            ↪ =1)#padd with zeros
         padd_img = np.append(padd_img,np.zeros((padd_img.shape[0],col_add)),axis=1)
            ↪ #padd with zeros
         rem_row = self.__img_height-padd_img.shape[0] #count how many rows to
            ↪ remove
         rem_col = self.__img_width -padd_img.shape[1] #count how many rows to
            ↪ remove
         if(rem_row>0):
            self.__ip_img = np.delete(self.__ip_img, rem_row, 0) #delete the rows
               ↪ from the image
            self.__img_height -= rem_row #set the image size accordingly

         if(rem_col>0):
            self.__ip_img = np.delete(self.__ip_img, rem_col, 1) #delete the columns
               ↪  from the image
            self.__img_width -= rem_col #set the image size accordingly
      return padd_img #return the padded image

   def inverse_fliter(self,sigma = -1): # method to do inverse filtering
      padd_kernel = self.padder(self.__kernel) # get the padded image
      H = self.DFT(padd_kernel,1) # find the DFT of the padded image
      string = "␣" # string to show title text
      F = np.ones_like(H) #get a array with all elements as one
      if(sigma != -1): # if the method is called only for radial inverse filtering
         for index, x in np.ndenumerate(F): #get each index for F
            if (np.sqrt(index[0]*index[0]+index[1]*index[1])>sigma): #if the index
               ↪ is in range keep it as 1
               F[index[0],index[0]] = 1
            else: # since it is not in range make it 0
               F[index[0],index[0]] = 0
      B,G,R = self.DFT(np.true_divide(self.__ip_img,255.0)) #get the DFT the
         ↪ normalised image
      INV_B = (B/H)*F # do the radial or only inverse filtering depending on sigma
      INV_G = (G/H)*F # do the radial or only inverse filtering depending on sigma
      INV_R = (R/H)*F # do the radial or only inverse filtering depending on sigma
      ib = self.IDFT(INV_B)*255.0 #get back to normal range and perform IDFT
      ig = self.IDFT(INV_G)*255.0 #get back to normal range and perform IDFT
      ir = self.IDFT(INV_R)*255.0 #get back to normal range and perform IDFT
      self.__img_b = (np.absolute(ib)).astype(self.__ip_img.dtype) #put them in
         ↪ global variables
      self.__img_g = (np.absolute(ig)).astype(self.__ip_img.dtype) #put them in
         ↪ global variables
      self.__img_r = (np.absolute(ir)).astype(self.__ip_img.dtype) #put them in
         ↪ global variables
      cv.imwrite("temp.jpg",cv.merge((self.__img_b,self.__img_g, self.__img_r))) #
         ↪ write it to the temp image
      self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
         ↪ IMREAD_COLOR)) # read the temp image to show in GUI
      if(sigma != -1): # for title and display text
         string += "Radial␣Inverse␣Filter␣Applied␣with␣cutoff␣=␣"+str(sigma)
      else:
         string = "Inverse␣Filter␣Applied"
      self.disp(string) # display image
      print(string) #print status to terminal

   def inv_inbuilt(self): #to get the blurred image from kernel
      motion_blr = cv.filter2D(self.__ip_img,-1,np.divide(self.__kernel,np.sum(self.
         ↪ __kernel).astype(self.__ip_img.dtype))) #perform convolution
      cv.imwrite("temp.jpg",motion_blr) # write to temp image
      self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
         ↪ IMREAD_COLOR)) #read from temp to show
      self.disp("Blurred␣Image") # display it in GUI
      print("Blurring␣using␣kernel") # Print status to terminal or IDE

   def radial_filter_threshold(self):#to get the cutoff from user
      self.lbl_s3.resize(500,50)#label to display title for output image
      self.lbl_s3.setText("Please␣Enter␣an␣Integer␣value␣Threshold")#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
      radial_threshold = QPushButton('OK', self) #button to click ok to start
         ↪ operaion on the input
      radial_threshold.resize(50,30) #resize the button
      radial_threshold.move(610, 600) #positioning
      self.lbl_s5.clear()
      self.lbl_s4.clear()
      self.e4.clear()
      self.e3.clear()
      radial_threshold.show() #display button
      self.e2.show() #display text box
      radial_threshold.clicked.connect(lambda: self.inverse_fliter(int(self.e2.text
         ↪ ()))) #call inverse_fliter when clicked

   def weiner_filtering(self): #to get the k image from user
      self.lbl_s4.resize(500,50)#label to display title for output image
      self.lbl_s4.setText("Please␣Enter␣an␣Integer␣value␣of␣K")#title text
      self.lbl_s4.move(100,590) #positioning
      self.lbl_s4.show() #display title
      self.e3.setValidator(QIntValidator())#text box setting to allow only integer
         ↪ values
      self.e3.move(500,600) #positioning
      weiner_k = QPushButton('OK', self) #button to click ok to start operation on
         ↪ the input
      weiner_k.resize(50,30) #resize the button
      weiner_k.move(610, 600) #positioning
      weiner_k.show() #display button
      self.lbl_s5.clear()
      self.lbl_s3.clear()
      self.e4.clear()
      self.e2.clear()
      self.e3.show() #display text box
      weiner_k.clicked.connect(lambda: self.weiner(int(self.e3.text()))) #call
         ↪ weiner when clicked

   def weiner(self,k): #to perform weiner filtering
      padd_kernel = self.padder(self.__kernel) # get the padded image
      H = self.DFT(padd_kernel,1)# find the DFT of the padded image
      B,G,R = self.DFT(np.true_divide(self.__ip_img,255.0))# find the DFT of the
         ↪ image
      INV_B = np.multiply(B,np.divide(np.power(np.absolute(H),2),(np.multiply(H,np.
         ↪ power(np.absolute(H),2)+k)))) #perform weiner filter and get the channel
      INV_G = np.multiply(G,np.divide(np.power(np.absolute(H),2),(np.multiply(H,np.
         ↪ power(np.absolute(H),2)+k)))) #perform weiner filter and get the channel
      INV_R = np.multiply(R,np.divide(np.power(np.absolute(H),2),(np.multiply(H,np.
         ↪ power(np.absolute(H),2)+k)))) #perform weiner filter and get the channel

      ib = self.IDFT(INV_B)*255.0 #perform IDFT
      ig = self.IDFT(INV_G)*255.0 #perform IDFT
      ir = self.IDFT(INV_R)*255.0 #perform IDFT
      self.__img_b = (np.absolute(ib)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      self.__img_g = (np.absolute(ig)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      self.__img_r = (np.absolute(ir)).astype(self.__ip_img.dtype)#put them in
         ↪ global variables
      cv.imwrite("temp.jpg",cv.merge((self.__img_b,self.__img_g, self.__img_r))) #
         ↪ write it to temp image
      self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
         ↪ IMREAD_COLOR))# read the temp image to show in GUI
      self.disp("Weiner␣Filter␣Applied␣for␣k␣=␣"+str(k)) #display it to GUI
      print("Weiner␣Filter␣Applied␣for␣k␣=␣"+str(k)) #print status to terminal

   def ls_filtering_gamma(self): # to get gamma values
      self.lbl_s5.resize(500,50)#label to display title for output image
      self.lbl_s5.setText("Please␣Enter␣an␣Integer␣value␣of␣gamma")#title text
      self.lbl_s5.move(100,590) #positioning
      self.lbl_s5.show() #display title
      self.e4.setValidator(QIntValidator())#text box setting to allow only integer
         ↪ values
      self.e4.move(500,600) #positioning
      gamma = QPushButton(’OK’, self) #button to click ok to start operaion on the
         ↪ input
      gamma.resize(50,30) #resize the button
      gamma.move(610, 600) #positioning
      gamma.show() #display button
      self.lbl_s3.clear()
      self.lbl_s4.clear()
      self.e2.clear()
```

```
        self.e3.clear()
        self.e4.show() #display text box
        gamma.clicked.connect(lambda: self.ls_filter(int(self.e4.text()))) #call
            ↪ blur_img when clicked

        print("gamma␣value␣given␣is␣=␣"+ str(self.e4.text())) # Print status to
            ↪ terminal or IDE
    def ls_filter(self,gamma=1): # to perform LS filtering
        p = np.array([[0,-1,0],[-1,4,-1],[0,-1,0]]) # blur kernel
        padd_p = self.padder(p) # get the padded image
        P = self.DFT(padd_p,1) # find the DFT of the padded image
        h = self.padder(self.__kernel)# get the padded image
        H = self.DFT(h,1) # find the DFT of the kernel image
        B,G,R = self.DFT(np.true_divide(self.__ip_img,255.0)) #get the DFT the
            ↪ normalised image
        filter = np.divide(np.conj(H),(np.power(np.absolute(H),2)+gamma*np.power(np.
            ↪ absolute(P),2))) #get LS filter
        R_trans = np.multiply(filter,R)# do the LS filtering depending on gamma
        G_trans = np.multiply(filter,G)# do the LS filtering depending on gamma
        B_trans = np.multiply(filter,B)# do the LS filtering depending on gamma
        ib = self.IDFT(B_trans)*255.0#get back to normal range and perform IDFT
        ig = self.IDFT(G_trans)*255.0#get back to normal range and perform IDFT
        ir = self.IDFT(R_trans)*255.0#get back to normal range and perform IDFT
        self.__img_b = (np.absolute(ib)).astype(self.__ip_img.dtype)#put them in
            ↪ global variables
        self.__img_g = (np.absolute(ig)).astype(self.__ip_img.dtype)#put them in
            ↪ global variables
        self.__img_r = (np.absolute(ir)).astype(self.__ip_img.dtype)#put them in
            ↪ global variables
        cv.imwrite("temp.jpg",cv.merge((self.__img_b,self.__img_g, self.__img_r))) #
            ↪ write it to the temp image
        self.__img_b,self.__img_g,self.__img_r = cv.split(cv.imread("temp.jpg",cv.
            ↪ IMREAD_COLOR)) # read the temp image to show in GUI
        self.disp("LS␣Filter␣Applied␣for␣gamma␣=␣"+str(gamma))# display image
        print("LS␣Filter␣Appliedfor␣gamma␣=␣"+str(gamma))#print status to terminal

    def metrics(self): #to undo the last change done on the image
        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
        grd_truth = (cv.imread(str(name),cv.IMREAD_COLOR)).astype(np.float) #get the
            ↪ image name from user and read it
        restored_img = cv.merge((self.__img_b,self.__img_g, self.__img_r)).astype(np.
            ↪ float) #merge to get restored image
        grd_truth_resize = self.padder(grd_truth,1) #resize the ground truth to our
            ↪ size
        difference_squared = (grd_truth_resize.astype(np.float) -restored_img) ** 2 #
            ↪ get the square of the difference
        summ_diff_square = np.sum(difference_squared) #sum the difference
        pixels_size = np.prod(grd_truth_resize.shape) #get the pixel size of the image
        mse = summ_diff_square / pixels_size #find out the mean square error
        psnr = 20.0*np.log10(255/np.sqrt(mse)) #calculate PSNR
```

```
   ssim = compare_ssim(grd_truth_resize.astype(np.float), restored_img,
      ↪ multichannel=True) #get the SSIM
   self.lbl_s6.clear() # clear the past content in label if any is present
   self.lbl_s6.resize(200,70)
   self.lbl_s6.setText("PSNR␣="+str(psnr)+"dB\n"+’SSIM␣=␣’+str(ssim)) # Set PSNR
      ↪ and SSIM
   self.lbl_s6.move(500,440) # position the text
   self.lbl_s6.show()
   print(’ssim=␣’+str(ssim),’psnr␣=␣’+str(psnr)+"dB") #print status to terminal

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_b,self.__img_g, self.__img_r])#merge intensity
      ↪ with the hue and saturation
   itos = cv.cvtColor(itos, cv.COLOR_BGR2RGB)#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␣at␣"+str(name)) # 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,fft=0,img = np.empty_like([256,256])): # this method
   ↪ is used to display the transformed image to GUI
   if (fft == 0): #whether to clear some labels or not is decided by this flag
      ↪ variable
      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
      if (flag == 0 ):
         img_pix1 = cv.merge((self.__img_b,self.__img_g, self.__img_r)) #merge
            ↪ the v with h and s using cv.merge
         img_pix1 = cv.cvtColor(img_pix1, cv.COLOR_BGR2RGB)
         pix_img = QtGui.QPixmap(QtGui.QImage(img_pix1,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
   else:
      pix_img = QtGui.QPixmap(QtGui.QImage(img,self.__img_width, self.
         ↪ __img_height,3*self.__img_width,QtGui.QImage.Format_Indexed8))
   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(930,0) #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()
```