iFilter Image Editor

Sufiyan Adhikari

August 26, 2018

Indian Institute of Technology Bombay

Abstract

Freeware Applications that can edit image properties are hard to find for Personal Computers(PC). We in this project present a simple image editor Gui Application written in python that can edit common image properties like Equalize Histogram, Gamma Correction, Log Transforms, Blur the image with a slider to control the extent of Blur and Image Sharpening. In order to facilitate this and further development, we introduce a new iImage Class that has built in history of applied changes that can be used to checkout earlier version of the image, inspired by Git.

Keywords— iFilter Image Editor Gui iImage Git

1 Introduction

Freeware Applications that can edit image properties are hard to find for Personal Computers(PC). We in this project present a simple image editor Gui Application written in python that can edit common image properties like Equalize Histogram, Gamma Correction, Log Transforms, Blur the image with a slider to control the extent of Blur and Image Sharpening. In this project we present a simple Image Editor Gui called **iFilter** that can import image, apply the above mentioned transformations and allow the user to **checkout** any of the edited and original version on the image with a single click and save any or all versions individually if the user desires to. In order to facilitate ease of development and further development, we introduce a

new **iImage** class that takes in an RGB *numpy* array of the image and saves it as a HSV Image. Only V Channel of the image is manipulated and the new image can be easily retrieved for visualization with *iImage.RGB* property of the iImage class. There are other class methods like *iImage.load* that calls the constructor after loading the image from path and *iImage.save* that saves the current image with a passed name argument, both inspired by and takes direct use of *Python Image Library (PIL)*

The Gui is written in **Qt**. **Qt Creator** was used to design the Gui first and then was converted into python with the *pyuic* executable that comes with Python wrapper for Qt **PyQt5** package.

The Gui uses two different backends to display image, namely Matplotlib's *FeatureCanvas*[1] and Qt's *QPixmap* that the user can switch between before running the code using *use_matplotlib_backend* flag.

2 Background Read

In order to accomplish this project, we had to learn about how to apply a function on each element of an *ndarray*. This can be achieved by first writing the function for a single element and then passing it to ***numpy.vectorize***[2] that returns a function that can apply the initial function to all the elements of the passed *ndarray*. This is faster than making a list composition of the *ndarray* and then converting it back to *ndarray*.

We had to learn about **Qt**, all its classes and how to use **Qt Creator** to write a *ui* application that can then be converted to a *python* file. Installing Qt backend was highly buggy on *Linux* and turned out to be unnecessary as all we needed was **Qt Creator** tool.

We had to read about other Qt classes like QSizePolicy, QLayout and QSpacerItem that facilitate in making the Gui cleaner and re-sizable and implemented then in out code. A lot of buttons were later added manually.

We also had to read a lot of **wikipedia** pages and **StackExchange** for different approaches like displaying an Frequency image, centering it with *scipy.fftpack.fftshift* and it's inverse and other tricks.

We had to learn about the Matplotlib Backend for displaying images in the Application and how to display and redisplay a new image to this backend. We also learned about the *QPixmap* class in Qt that can be used to display image inside *QLabel*. We have implemented both backends for use. QPixmap takes QImage class object and hence this was added to iImage class

later as *iImage.ImageQ* property.

3 Approach

We first implemented an iImage class that takes in an Image array and saves it's HSV Image. the iImage class has all the implemented transforms as methods that transform the V Channel of the image and the either save it in the iImage class object they were called from, or return a new iImage object depending on *save* argument. These *save* argument is defaulted to *False* and there is another set of method with underscore (_)at the end of method name to represent inplace operation of these transforms inspired by *pytorch*. hence, *iImageObject.logTransform(save=False)* with default *save=False* argument will return new iImage object whereas *iImageObject.logTransform_()* will perform log transform on the object's image and save it in the object's history and return the same object.

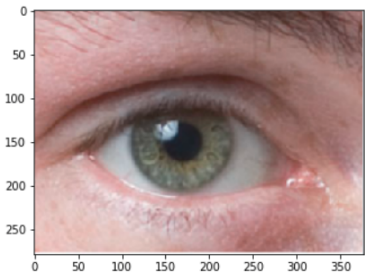
The Gui as discussed earlier in this report is written in Qt and uses iImage class for all the operations on image. All the image handling is done by *iImage* class and the *Ui_MainWindow* uses this *iImage* class for image handling and operations. The *Ui_MainWindow* also uses *MplCanvas* class for displaying the current image inside the Ui.

4 Selection of images

Many Images were used interchangeably for testing the code as we went on adding different methods and found some test examples on the internet for each transform.
Below Image form Wikipedia[3] was used for testing gamma correction. The page provides different reference images of this image at different gamma values, making it easy to analyze the performance of our application.

Figure 1: Floating lady [3]

The below Eye Image[4] was used to check Image sharpening of the application. There were other images as well that were interchangeably used for testing multiple things. The below Lichtenstein image is one such example from internet.



(a) a. Eye[4]

(b) b. Lichtenstein

Figure 2: 2 Image Used

Other Images used for testing are the Grayscale and Color Photo of Lena Soderberg that has been used in image processing as test image since 1973

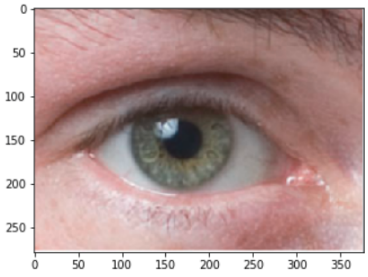
(a) a. Lena Color

(b) b. Lena Grayscale

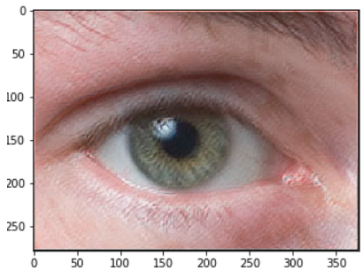
Figure 3: Images OF Lena Used for Testing

5 Results on Test Images

Below are the results on test images using our iIMage Class in Jupyter Notebook



(a) Original Image



(b) After Unmask Sharpening

Figure 4: Eye Image Sharpening

```
In [400]: ratio=0.1
f, ((ax1,ax2),(ax3,ax4))=plt.subplots(2,2, figsize=(15,15))
mapable=image.show(axis=ax1); f.colorbar(mapable, ax=ax1)
mapable=image.fft().apply_freq_filter(mode='highpass', shape='radial', parameter_ratio=ratio).show(isFFT=True,
axis=ax3)
f.colorbar(mapable, ax=ax3)
im=iImage.load("/home/sufiyan/Dropbox/Image Processing/Assignments/iFilter Image Editor
Gui/images/Lichtenstein.png")
mod_image=image.fft().apply_freq_filter(mode='highpass', shape='radial', parameter_ratio=ratio).ifft()
im.ImageV+20*mod_image.ImageV
im=im.RGB
mapable=mod_image.show(axis=ax2); f.colorbar(mapable, ax=ax2)
mapable=iImage(im*(255/im.max())).show(axis=ax4)
f.colorbar(mapable, ax=ax4)
plt.show()

/home/sufiyan/anaconda3/envs/fastai/lib/python3.6/site-packages/ipykernel_launcher.py:171: ComplexWarning: Casting
complex values to real discards the imaginary part
```

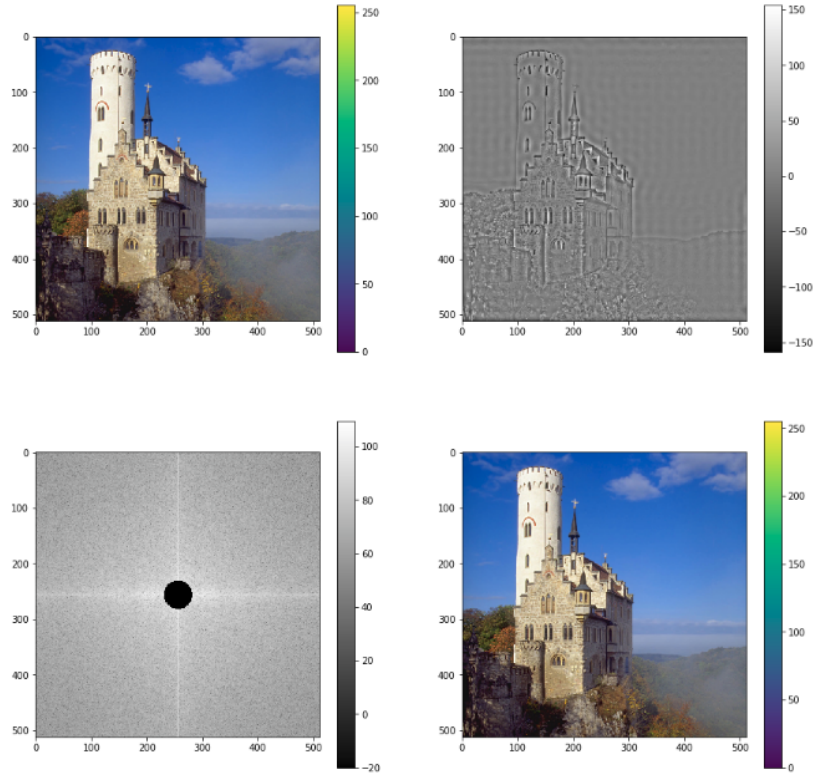


Figure 5: Highpass filtering on Lichtenstein

Highpass Filtering was working too as can be seen in the above image, but taking the weighted sum of Filtered image and original image wasn't looking sharp at all. Hence Sharpening is also currently done with Convolution filters. But the above figure shows the filtered mask correctly.

Lowpass Filtering of Images in Frequency Domain. This Feature was still buggy and hence is not default way of Bluring in the application. Default method is Convolving with Filters. Frequency Filtering can be activated by changing a small line of code in iIMage class.

```
In [326]: f, ((ax1,ax2),(ax3,ax4))=plt.subplots(2,2, figsize=(15,15))
          mapable=image.show(axis=ax1); f.colorbar(mapable, ax=ax1)
          mapable=image.fft().show(isFFT=True, axis=ax2); f.colorbar(mapable, ax=ax2)
          mapable=image.fft().apply_freq_filter(mode='lowpass', shape='radial', radius_ratio=0.1).show(isFFT=True, axis=ax3)
          f.colorbar(mapable, ax=ax3)
          mapable=image.fft().apply_freq_filter(mode='lowpass', shape='radial', radius_ratio=0.1).ifft().show(axis=ax4)
          f.colorbar(mapable, ax=ax4)
          plt.show()

/home/sufiyan/anaconda3/envs/fastai/lib/python3.6/site-packages/ipykernel_launcher.py:164: ComplexWarning: Casting c
omplex values to real discards the imaginary part
```

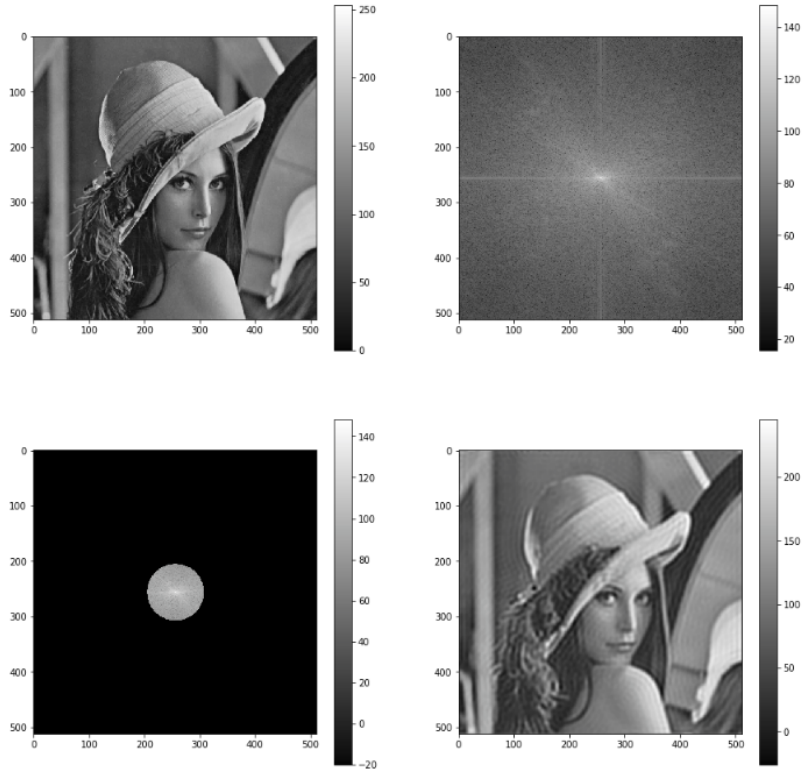


Figure 6: Lowpass Filtering on Lena

Bluring with mean filter and Log Transform can be seen below on Lena

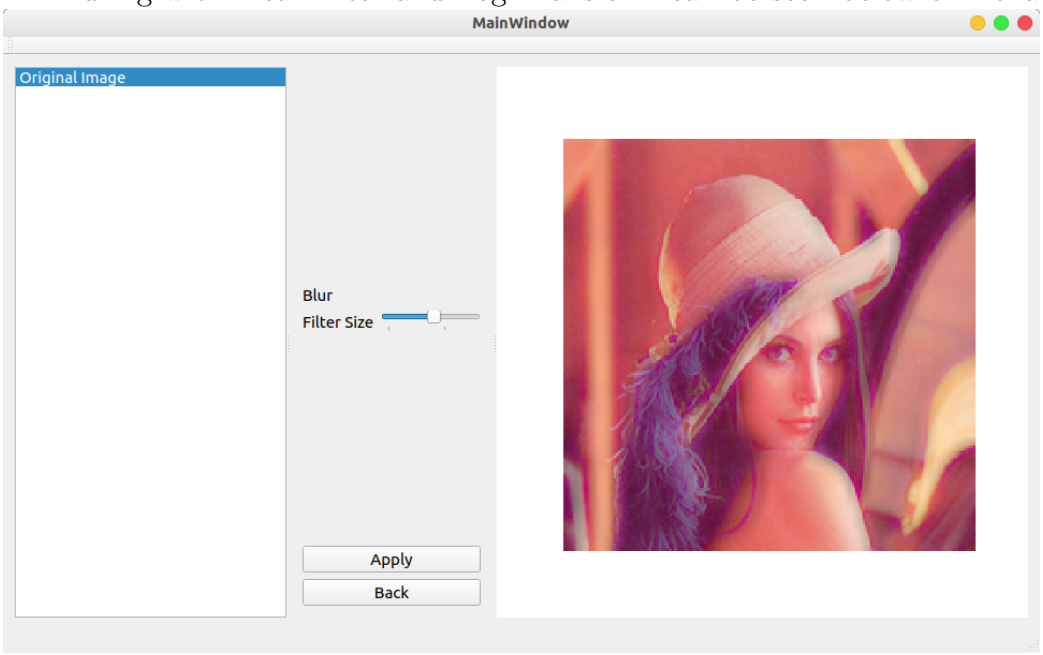


Figure 7: Lena in Color with Blur Applied

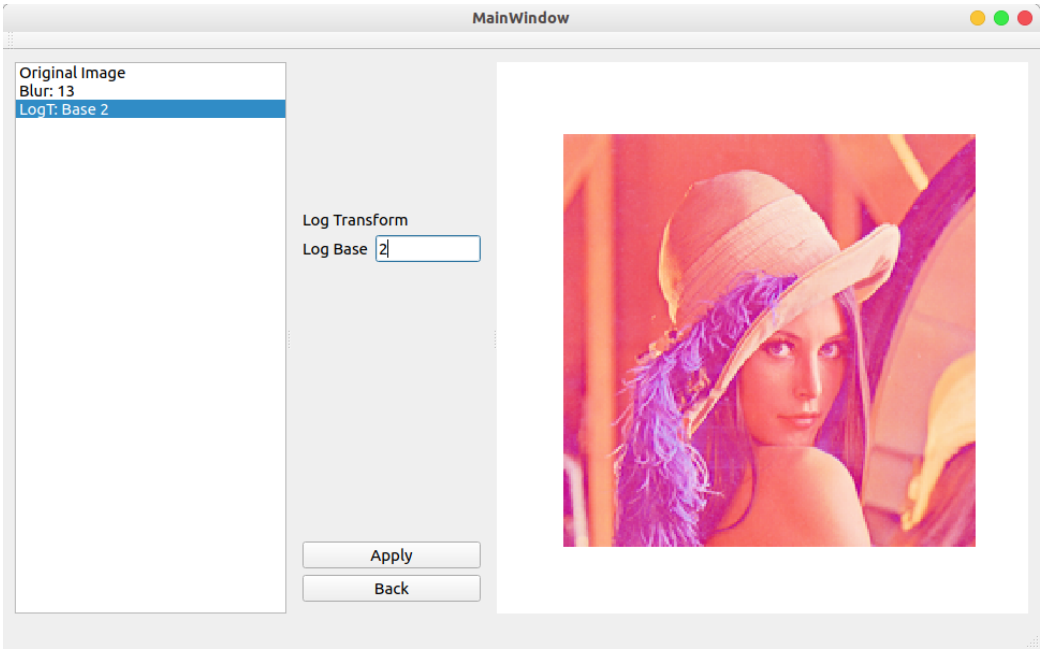


Figure 8: Lena in Color with Blur and Log Transform Applied

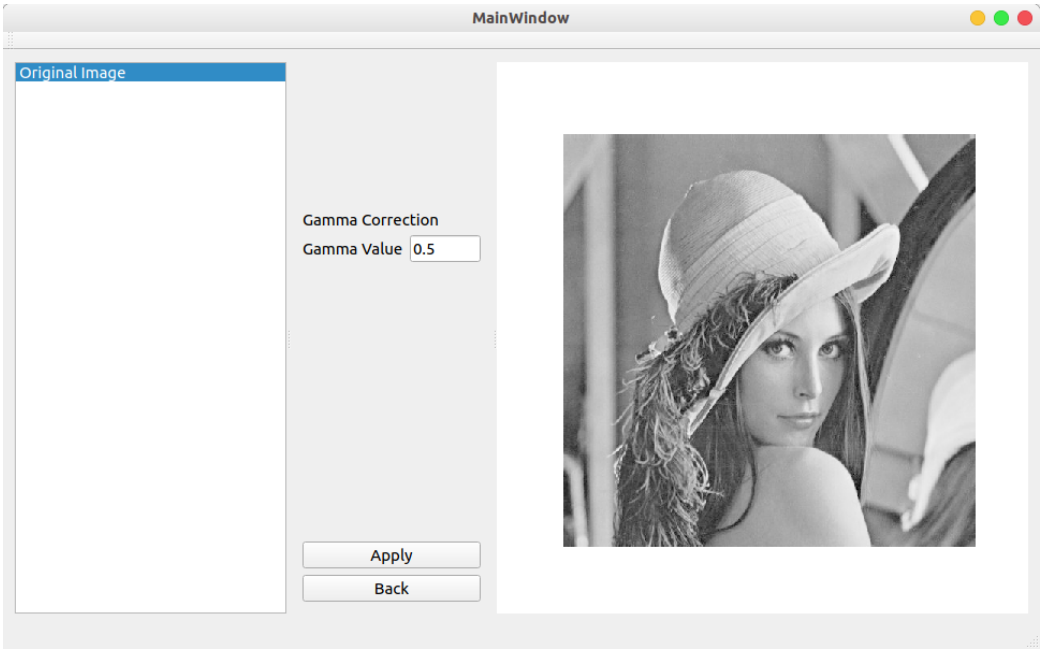


Figure 9: Lena in Grayscale with Gamma Corrected

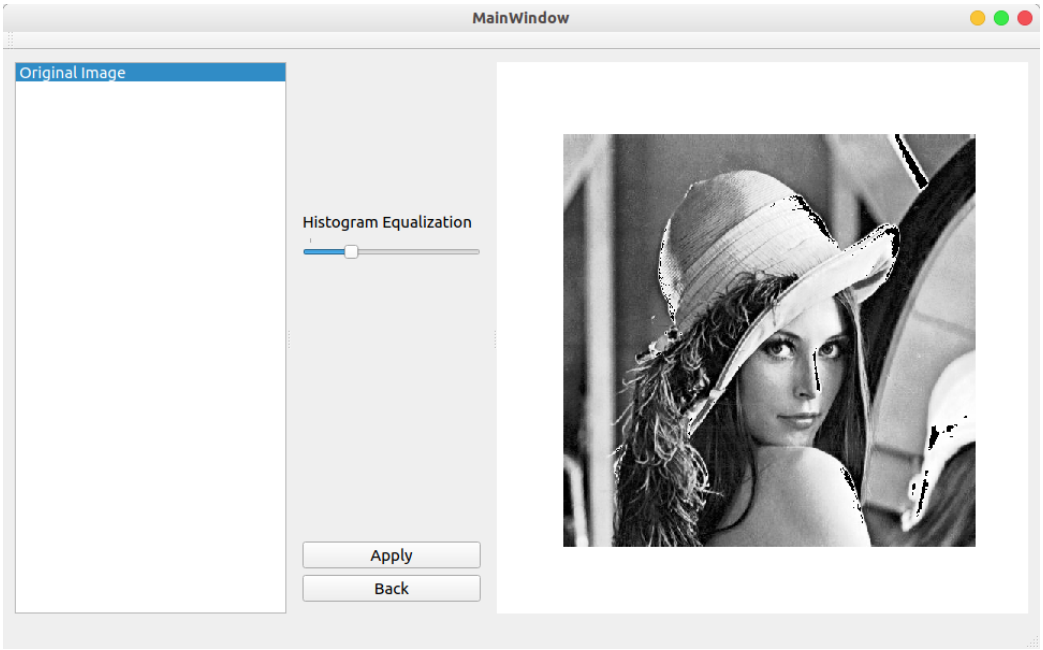


Figure 10: Lena in Grayscale with Histogram Equalization

6 Discussion

iFilter Image Editor was a great learning experience and there surely are a lot of things we would love to add to it in future. As discussed in above sections, the frequency transforms are a little bit buggy in terms of displaying the images in QPixmap. Since then, Matplotlib's FeatureCanvas was addopted at the last moment and we were not able to test the frequency transforms with FeatureCanvas. hence, convolution based bluring and unmask sharpning are the currently used techniques with frequency filtering completely implemented in the code. We would love to test it more thoroughly before adopting it as the default method of bluring and sharpening the images.

We wanted to add QRadioButtons in the Gui so as to be able to switch between the available methods like in Bluring and Sharpening where convolution and Frequency Filtering methods are both implemented, and being able to switch between display backend i.e., QPixmap and FeatureCanvas would be great. QPixmap shows the images without any whitespaces which looks better, was found to be buggy and sometimes fails to show any image. Even though there are multiple options implemented as mentioned above, editing the Gui at the end was too time consuming and hence it has been shelved for the moment. Hopefully, we will add these features in future iterations of the Gui Application.

References

[1] Matplotlib.org. *Embedding in Qt — Matplotlib V2.2.3 Documentation*. 2018. URL: https://matplotlib.org/gallery/user_interfaces/embedding_in_qt_sgskip.html.

[2] Stackoverflow. *Most efficient way to map function over numpy array — Stackoverflow*. 2018. URL: https://stackoverflow.com/a/35215329.

[3] Wikipedia. *Gamma correction — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-August-2018]. 2018. URL: http://en.wikipedia.org/w/index.php?title=Gamma\%20correction&oldid=837514768.

[4] Wikipedia. *Unsharp masking — Wikipedia, The Free Encyclopedia*. 2018. URL: http://en.wikipedia.org/w/index.php?title=Unsharp\%20masking&oldid=815933773.

7 Appendix

7.1 iImage.py

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import math
import scipy.fftpack as fp
from PIL import ImageQt


def conv2D(image, filt, padding='reflect', **kwargs): #1 Channel Conv2D
        pad=int((filt.shape[0]-1)/2)
        m,n=filt.shape
        i,j=image.shape
        weight=1/filt.sum()
        image=image.astype(np.float32) #for multithreading
        filt=filt.astype(np.float32)   #for multithreading. Works only with float32
        if padding is None:
            padded=image #don't pad. used for sharpenning where  padding messes things up
            c=np.asarray([[(weight*np.multiply(padded[x:x+m,y:y+n],filt)).sum()
                      for y in range(j-n+1)] for x in range(i-m+1)])
            return c.reshape((i-m+1, j-n+1))
        else:
            padded=np.pad(image, pad_width=pad, mode=padding, **kwargs)
            c=np.asarray([[(weight*np.multiply(padded[x:x+m,y:y+n],filt)).sum()
                      for y in range(j)] for x in range(i)])
            return c.reshape(image.shape)

from matplotlib.colors import rgb_to_hsv, hsv_to_rgb
class iImage(object):
    def __init__(self, image):
        """Custom Image Class with inbuilt Transforms
        Parameters
        ----------
        image: Numpy nD array for RGB Color Image, or Grayscale Image

        Returns
        ----------
        iImage Object
        """
        if len(image.shape)==2:
            image=np.dstack((image,image,image)) #Convert 1 channel image to 3 channel
   grayscale
```

```
        if image.shape[2]==3:                                   # 3 Channel Image
            if not self.is3CGrayScale(image):                       #Is 3 Channel RGB Image
                self.isRGB=True
                self.OGImage=rgb_to_hsv(image)
                self.ImageV=self.OGImage[:,:,2]         #Current HSV Image V Channel (Inatialised to Original)
            else:                                               #Is 3 Channel GrayScale
                self.isRGB=False
                self.OGImage=np.dstack((np.zeros(image.shape[:2]),
                                        np.zeros(image.shape[:2]),
                                        image[:,:,0]))              #Use First Channel as Grayscale's VChannel, H and S channel are zeros
                self.ImageV=image[:,:,0]
        else:
            self.OGImage=image
            self.ImageV=self.OGImage.copy()                 #Current GrayScale image
            self.isRGB=False

        self.history=[]           #Saves different version of the image transformed over time
        self.text_history=[]

        self.history.append(self.ImageV)
        self.text_history.append('Original Image')

    @classmethod
    def load(cls, path):
        rawImage=Image.open(path).convert('RGB')
        rawImage=np.asarray(rawImage)
        return cls(rawImage)

    def save(self, filename):  Image.fromarray(self.RGB).save(filename)

    @property
    def RGB(self): image=self.getRGB(); return ((255/image.max())*image).astype('uint8')   #Used to return RGB Image After Transforms


    @property
    def QImage(self): #Retruns QImage wrapper over current image, to be used by Qt's Qpixmap
        return ImageQt.ImageQt(Image.fromarray(self.RGB))

    @property
    def HChannel(self): return self.OGImage[:,:,0] #Return H Channel of the main iIMage object
    @property
```

```
82      def SChannel(self): return self.OGImage[:,:,1] #Return's S Channel. Both used for
    assembling HSV image before converting to RGB
83      @property
84      def VChannel(self): return self.ImageV  #Used to checkout any state in History. Returns
    VChannel
85
86      @property
87      def V(self): return self.ImageV
88
89      def getRGB(self, VChannel=None):        #Used to return RGB from custom HSV VChannel
90          if VChannel is None: VChannel=self.ImageV
91          if self.isRGB: return hsv_to_rgb(np.dstack(
92                                  (self.HChannel, self.SChannel, VChannel)))
93          else: return np.dstack((VChannel, VChannel, VChannel))
94
95      def checkout(self,index): #Checkout some image from history. Like Git
96          self.ImageV=self.history[index]
97          return self
98
99      def is3CGrayScale(self, image): #Checks if 3Channnel Image is Greyscale
100         return not(False in ((image[:,:,0]==image[:,:,1]) == (image[:,:,1]==image[:,:,2])))
101
102     def checkSave(self, transformedImage, save, save_text):
103
104             if save:
105                 self.history.append(transformedImage) #If Save=True is passed, save the
    transformed VChannel in main iIMage object
106                 self.text_history.append(save_text)
107                 self.ImageV=transformedImage
108                 return self
109             else:
110                 return iImage(self.getRGB(VChannel=transformedImage)) #Else return a new
    temporary iImage object
111
112     def fft(self): #Get fft of the main iImage's VChannel, returned a a new temporary iImage
    object that can be manupulated without touching the mail iImage
113         image=self.ImageV.astype(float)
114         image=fp.ifftshift(fp.fft2(image) ) #Calc fft and center the zero frequency
115         return iImage(image) #return a new object of main iImage's VChannel's fft. only
    VChannel of this new object will again be extracted by its .ifft() method
116
117     def ifft(self): #Get ifft of the image as a new iImage object
118         image=self.ImageV  #Take the current image's v channel (expected to be an frequency
    domain image), hence should be used after self.fft() only
119         image= (fp.ifft2(fp.ifftshift(image))).real #Shift the zero freq to edges and calc
```

```
inverse fft
        return iImage(image) #return a new temporary iImage object of this ifft image. Only .VChannel will be extracted from it by the mail iImage object.

    def apply_freq_filter(self, parameter_ratio=0.2, mode='highpass', shape="radial"): #radius_ratio is ratio of radius of filter to smallest side of image
        """Applies a mask of passed shape with parameter of shape(side of square or radius of circle)
        """

        image=self.VChannel
        parameter=parameter_ratio*min(self.ImageV.shape[:2])/2 #parameter ratio times min of shape of 2d V Channel
        x0,y0= np.asarray(image.shape)/2
        radius=parameter; side=parameter #used in two types of filter masks
        def is_inside(i,j): #Check if i,j is inside the filter radius
            if shape=='square':

                #return true if pixel (i,j) are inside the square mask of side `side`
                if ((i>x0-side and i<x0+side) and(j>y0-side and j<y0+side)): return True
                else: return False
            elif shape=='radial':

                if ((x0-i)**2+(y0-j)**2)**0.5 < radius: return True #return true if point i,j is inside the radial makk
                else: return False

        if mode=='highpass':
            def inner(i,j): return 0
            def outer(i,j): return image[i,j]
        elif mode=='lowpass':
            def inner(i,j): return image[i,j]
            def outer(i,j): return 0
        else: raise NotImplementedError #other modes of filters are not implemented

        filtered_image= np.asarray([[inner(i,j) if is_inside(i,j) else outer(i,j) for j in range(image.shape[1])]
                                                                                    for i in range(image.shape[0])])
        return iImage(filtered_image)

    def logTransform_(self, base=None): return self.logTransform(base,save=True) #Inplace (PyTorch Like)
    def logTransform(self, base=None, save=False): #Log  Transforms
        """
```

```
156         Parameters
157         base: base value for Log Transform. default Loge
158         save: Save the Transformed Image to History
159         """
160         import math
161         if base:  base = int(base)    #If base is provided, use it, else Natural Log e
162         c=255/math.log(1+self.ImageV.max())   #Scaling Constant for Gamma Transform
163         if base: func = np.vectorize(lambda x: int(c*math.log(1+x, base))) #element wise log
    transform
164         else:    func = np.vectorize(lambda x: int(c*math.log(1+x)))
165         transformedV= func(self.ImageV)
166         return self.checkSave(transformedV, save, f'LogT: Base {base}')
167 
168     def show(self, isFFT=False, axis=None, *args, **kwargs): #Can be used for debugging.
    shows the RGB image be default. can be used to see frequency spectrum
169             import matplotlib.pyplot as plt
170             if axis is None: f, axis=plt.subplots(1,1, *args, **kwargs)
171             if isFFT:       mapable=axis.imshow(20*np.log10(0.1+self.VChannel).astype(float),
     cmap=plt.cm.gray, *args, **kwargs)
172             elif self.isRGB:mapable=axis.imshow(self.RGB) #Show RGB Image
173                 #Show Grayscale Image (avoids calling self.getRGB that calls hsv_to_rgb on
    the image that was caussing image to look really bad)
174             else:           mapable=axis.imshow(self.VChannel, cmap=plt.cm.gray, *args, **
    kwargs)
175             return mapable
176 
177     def gammaTransform_(self, gamma=None): return self.gammaTransform(gamma, save=True) #
    Inplace (PyTorch Like)
178     def gammaTransform(self,gamma=None, save=False):
179         if gamma is None: self.gamma=1
180         else: self.gamma=gamma
181         from math import pow
182 
183         func = np.vectorize(lambda x: pow(x, self.gamma))
184         transformedG=func(self.ImageV/self.ImageV.max()) #Gamma of Normalized Image
185         transformedG = transformedG*(255/transformedG.max()) #Denormalizing to 8bit
186         return self.checkSave(transformedG, save, f'Gamma: {gamma}')
187 
188     def histEqualization_(self, iterations=1): return self.histEqualization(iterations, save=
    True) #Inplace (PyTorch Like)
189     def histEqualization(self, iterations=1, save=False):
190         import numpy as np
191         transformedH=np.zeros(shape=self.ImageV.shape)
192         for _ in range(iterations):
193             pdf,bins=np.histogram(self.ImageV, bins=256, density=True) #Returns PDF at
```

```
intensity(bin)
            cdf=pdf.cumsum() #calc cdf at each intensity bin
            #normalize histogram,scale by 255  and 8 bit
            transformedH=(np.interp(self.ImageV, bins[:-1], cdf)*255).astype('uint8')
        return self.checkSave(transformedH, save, f'HistogramEQ: {iterations}')

    def blur_(self,*args, **kwargs): return self.blur(*args, **kwargs, save=True) #Inplace (PyTorch Like)
    def blur(self, *args, **kwargs): return self.blur_1(*args, **kwargs)
    def blur_1(self, kernelSize=3, save=False):
        """Blur using average filter."""
        if kernelSize<1: kernelSize=1 #Handle all possible human error in kernel size
        elif kernelSize%2!=1:
            if int(kernelSize)%2==1: kernelSize=int(kernelSize)
            else: kernelSize=int(kernelSize)-1
        transformedB=conv2D(self.ImageV, np.ones(shape=(kernelSize,kernelSize)))
        #print(transformedB.shape)
        return self.checkSave(transformedB, save, f'Blur: {kernelSize}')

    def blur_2(self, param=5, save=False):
        """Blur using Frequenct Transforms
        passed param is inverse of frequency mask ratio"""
        freq_ratio=1/param #as low ratio gives more blured. Inverse was used to make it more-value = more-blur
        #Apply fft, apply a lowpass radial filter to the fft image and get inverse fft of the modified freq image and grab VChannel of that new object
        # self.fft().apply_freq_filter(freq_ratio, mode='lowpass').ifft().show()
        blured_image=self.fft().apply_freq_filter(freq_ratio, mode='lowpass').ifft().VChannel
        blured_image*(255/blured_image.max())
        return self.checkSave(blured_image, save, f'Blured: {freq_ratio}')


    def sharpen_(self, weight=0.5): return self.sharpen(weight=weight, save=True)
    def sharpen(self,*args, **kwargs): return self.sharpen_2(*args,**kwargs)
    def sharpen_1(self, weight=0.5, save=False): #Sharpen with unmask sharpening
        if weight<0:  weight = 0
        if weight >1: weight = weight
        # #print(f"Initial Dimension: {self.ImageV.shape}")
        blured=self.blur().V
        blured=blured/blured.max()  #Normalized to max 0-1
        orignal=self.ImageV/self.ImageV.max()
        transformedS = np.add(orignal*(1-weight), np.subtract(orignal, blured)*weight) # Unsharp Masking Wikipedia
        transformedS = np.add(self.ImageV, np.subtract(blured, orignal)*weight)
        # #print((np.subtract(orignal, blured)*weight).max())
```

```
234         transformedS=np.clip(transformedS, 0, 255)
235 #         transformedS= transformedS*(255/transformedS.max())
236         # #print(f"Final Dimention: {self.ImageV.shape}")
237 
238         return self.checkSave(transformedS, save, f'Sharpen: {weight}')
239 
240     def sharpen_2(self, weight=0.5, save=False): #Sharpen with convolving  a filter
241         """Sharpen using Convolution: Used in V1. Unused Now
242         """
243         weight/=5 #Scaling as QSlider has only integers
244         #kernel from https://nptel.ac.in/courses/117104069/chapter_8/8_32.html
245         kernel=0.34*np.asarray([[8 if x==y else -1 for y in range(3)] for x in range(3)])
246         image=self.ImageV/self.ImageV.max()
247         filtered=conv2D(image, kernel, padding=None, constant_values=0)
248         #Avoiding Padding makes image smaller, so adding 0 pad after extracting edges
249         filtered=np.pad(filtered, pad_width=int((kernel.shape[0]-1)/2), mode='constant',
    constant_values=0)
250         edges= np.subtract(filtered, image)
251         edges=edges*255/edges.max()
252         transformedS=np.add(self.ImageV, edges*weight)
253         transformedS=np.clip(transformedS, 0, 255)
254         return self.checkSave(transformedS, save, f'Sharpen: {weight}')
255 
256     def sharpen_3(self, freq_ratio=0.1, freq_mask_shape='radial', save=False, *args, **kwargs
    ):
257         """Sharpening Filter using FFT
258         Parameters
259         ----------
260         freq_ratio: ratio of shape parameter(side of square or radius of circle) to smallest
    side of photo
261         freq_mask_shape: ('radial', 'square', 'custom') Shape of zero mask to be used in
    frequency domain
262         Returns
263         -------
264         New iImage object with Sharpened iImage if save=False, or Same Object with Sharpened
    image"""
265         if   freq_mask_shape=='radial': low=0.005; high=0.05 #By trial and Error. Written in
    parameters.txt
266         elif freq_mask_shape=='square': low=0.01;  high=0.1
267         else: raise NotImplementedError
268         freq_ratio= low + freq_ratio*(high-low)/20 #Scaling Slider returned value between
    high and low. slider max=20
269 
270         image=self.ImageV
271         #Apply fft on current VChannel, apply a zero mask on it, take ifft of the modified
```

```
    image, and grab the VChannel of the returned object
272         edge_mask=self.fft().apply_freq_filter(freq_ratio, mode='highpass', shape=
    freq_mask_shape, *args, **kwargs).ifft().VChannel
273         image=image+edge_mask #Combine edge mask and orignal image
274         image*(255/image.max()) #Normalize for 8  bit
275         return self.checkSave(image, save, f'Sharpened: {freq_ratio}')
276
277 if __name__=="__main__":
278     image=iImage.load("/home/sufiyan/Pictures/Screenshot from 2018-08-24 14-11-32.png")
279     import matplotlib.pyplot as plt
280     plt.imshow(image.sharpen(5).show())
```

7.2 Final_GUI.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'mainwindow.ui'
#
# Created by: PyQt5 UI code generator 5.11.2
#
# WARNING! All changes made in this file will be lost!

import sys
import os
import matplotlib
import math
# Make sure that we are using QT5
matplotlib.use('Qt5Agg')
from PyQt5 import QtCore, QtWidgets, QtGui
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
from iImage import iImage
import matplotlib.pyplot as plt
from PIL import ImageQt

from PyQt5 import QtCore, QtGui, QtWidgets

use_matplotlib_backend=True

class MplCanvas(FigureCanvas):
    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig=Figure(figsize=(width,height), dpi=dpi)
        fig.tight_layout()
        self.axis=fig.add_subplot(111)
        self.axis.imshow(iImage.load("no_image.png").RGB)
        FigureCanvas.__init__(self,fig)
        self.setParent(parent)
        self.axis.axis('off')
        self.axis.get_xaxis().set_visible(False)
        self.axis.get_yaxis().set_visible(False)
        FigureCanvas.setSizePolicy(self,QtWidgets.QSizePolicy.Expanding, QtWidgets.
    QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)

class Ui_MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
```

```
43          QtWidgets.QMainWindow.__init__(self)
44          self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
45          self.setWindowTitle("application main window")
46          self.setupUi()
47          self.setupDefaults()
48          self.set_button_bindings()
49
50      def setupUi(self):
51          # MainWindow.setObjectName("MainWindow")
52          # MainWindow.resize(858, 572)
53          self.centralWidget = QtWidgets.QWidget()
54          self.centralWidget.setObjectName("centralWidget")
55          self.horizontalLayout = QtWidgets.QHBoxLayout(self.centralWidget)
56          self.horizontalLayout.setContentsMargins(11, 11, 11, 11)
57          self.horizontalLayout.setSpacing(6)
58          self.horizontalLayout.setObjectName("horizontalLayout")
59          self.splitter = QtWidgets.QSplitter(self.centralWidget)
60          sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.
    QSizePolicy.Preferred)
61          sizePolicy.setHorizontalStretch(0)
62          sizePolicy.setVerticalStretch(0)
63          sizePolicy.setHeightForWidth(self.splitter.sizePolicy().hasHeightForWidth())
64          self.splitter.setSizePolicy(sizePolicy)
65          self.splitter.setOrientation(QtCore.Qt.Horizontal)
66          self.splitter.setObjectName("splitter")
67
68          self.listWidget = QtWidgets.QListWidget(self.splitter)
69          self.listWidget.setObjectName("listWidget")
70          self.listWidget.setSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.
    Preferred)
71          # self.gridLayout_2.addWidget(self.listWidget, 8, 1, 1, 1)
72
73          self.stackedWidget = QtWidgets.QStackedWidget(self.splitter)
74          self.stackedWidget.setEnabled(True)
75          self.stackedWidget.setObjectName("stackedWidget")
76          self.page_main = QtWidgets.QWidget()
77          self.page_main.setObjectName("page_main")
78          self.gridLayout = QtWidgets.QGridLayout(self.page_main)
79          self.gridLayout.setContentsMargins(11, 11, 11, 11)
80          self.gridLayout.setSpacing(6)
81          self.gridLayout.setObjectName("gridLayout")
82          self.button_browse = QtWidgets.QPushButton(self.page_main)
83          self.button_browse.setObjectName("button_browse")
84          self.gridLayout.addWidget(self.button_browse, 1, 0, 1, 1)
85          spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Preferred, QtWidgets
```

```
    .QSizePolicy.Expanding)
86          self.gridLayout.addItem(spacerItem, 2, 0, 1, 1)
87          self.button_quit = QtWidgets.QPushButton(self.page_main)
88          self.button_quit.setObjectName("button_quit")
89          self.gridLayout.addWidget(self.button_quit, 3, 0, 1, 1)
90          self.stackedWidget.addWidget(self.page_main)
91          self.page_edit = QtWidgets.QWidget()
92          self.page_edit.setObjectName("page_edit")
93          self.gridLayout_2 = QtWidgets.QGridLayout(self.page_edit)
94          self.gridLayout_2.setContentsMargins(11, 11, 11, 11)
95          self.gridLayout_2.setSpacing(6)
96          self.gridLayout_2.setObjectName("gridLayout_2")
97          self.button_back = QtWidgets.QPushButton(self.page_edit)
98          self.button_back.setObjectName("button_back")
99          self.gridLayout_2.addWidget(self.button_back, 13, 0, 1, 1)
100         self.button_sharpen = QtWidgets.QPushButton(self.page_edit)
101         self.button_sharpen.setObjectName("button_sharpen")
102         self.gridLayout_2.addWidget(self.button_sharpen, 8, 0, 1, 1)
103         spacerItem1 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
104         self.gridLayout_2.addItem(spacerItem1, 5, 0, 1, 1)
105         self.button_hist = QtWidgets.QPushButton(self.page_edit)
106         self.button_hist.setAutoDefault(False)
107         self.button_hist.setFlat(False)
108         self.button_hist.setObjectName("button_hist")
109         self.gridLayout_2.addWidget(self.button_hist, 0, 0, 1, 1)
110         spacerItem2 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
111         self.gridLayout_2.addItem(spacerItem2, 11, 0, 1, 1)
112         spacerItem3 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
113         self.gridLayout_2.addItem(spacerItem3, 3, 0, 1, 1)
114         spacerItem4 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
115         self.gridLayout_2.addItem(spacerItem4, 9, 0, 1, 1)
116         spacerItem5 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
117         self.gridLayout_2.addItem(spacerItem5, 1, 0, 1, 1)
118         self.button_blur = QtWidgets.QPushButton(self.page_edit)
119         self.button_blur.setObjectName("button_blur")
120         self.gridLayout_2.addWidget(self.button_blur, 6, 0, 1, 1)
121         spacerItem6 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
122         self.gridLayout_2.addItem(spacerItem6, 7, 0, 1, 1)
123         self.button_save = QtWidgets.QPushButton(self.page_edit)
```

```
124         self.button_save.setObjectName("button_save")
125         self.gridLayout_2.addWidget(self.button_save, 12, 0, 1, 1)
126         self.button_gamma = QtWidgets.QPushButton(self.page_edit)
127         self.button_gamma.setObjectName("button_gamma")
128         self.gridLayout_2.addWidget(self.button_gamma, 4, 0, 1, 1)
129         self.button_log = QtWidgets.QPushButton(self.page_edit)
130         self.button_log.setObjectName("button_log")
131         self.gridLayout_2.addWidget(self.button_log, 2, 0, 1, 1)
132         spacerItem7 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
133         self.gridLayout_2.addItem(spacerItem7, 10, 0, 1, 1)
134 
135         self.stackedWidget.addWidget(self.page_edit)
136         self.page_hist = QtWidgets.QWidget()
137         self.page_hist.setObjectName("page_hist")
138         self.gridLayout_3 = QtWidgets.QGridLayout(self.page_hist)
139         self.gridLayout_3.setContentsMargins(11, 11, 11, 11)
140         self.gridLayout_3.setSpacing(6)
141         self.gridLayout_3.setObjectName("gridLayout_3")
142         self.button_back_to_edit_1=QtWidgets.QPushButton(self.page_hist)
143         self.button_back_to_edit_1.setObjectName("button_back_to_edit_1")
144         self.gridLayout_3.addWidget(self.button_back_to_edit_1, 6,0,1,1)
145         self.apply_hist = QtWidgets.QPushButton(self.page_hist)
146         self.apply_hist.setObjectName("apply_hist")
147         self.gridLayout_3.addWidget(self.apply_hist, 5, 0, 1, 1)
148         spacerItem8 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
149         self.gridLayout_3.addItem(spacerItem8, 4, 0, 1, 1)
150         spacerItem9 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.
    QSizePolicy.Expanding)
151         self.gridLayout_3.addItem(spacerItem9, 3, 0, 1, 1)
152         self.label = QtWidgets.QLabel(self.page_hist)
153         self.label.setObjectName("label")
154         self.gridLayout_3.addWidget(self.label, 1, 0, 1, 1)
155         self.slider_hist = QtWidgets.QSlider(self.page_hist)
156         self.slider_hist.setMinimum(1)
157         self.slider_hist.setMaximum(5)
158         self.slider_hist.setOrientation(QtCore.Qt.Horizontal)
159         self.slider_hist.setTickPosition(QtWidgets.QSlider.TicksAbove)
160         self.slider_hist.setObjectName("slider_hist")
161         self.gridLayout_3.addWidget(self.slider_hist, 2, 0, 1, 1)
162         spacerItem10 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
163         self.gridLayout_3.addItem(spacerItem10, 0, 0, 1, 1)
164         self.stackedWidget.addWidget(self.page_hist)
```

```
165         self.page_log = QtWidgets.QWidget()
166         self.page_log.setObjectName("page_log")
167         self.gridLayout_4 = QtWidgets.QGridLayout(self.page_log)
168         self.gridLayout_4.setContentsMargins(11, 11, 11, 11)
169         self.gridLayout_4.setSpacing(6)
170         self.gridLayout_4.setObjectName("gridLayout_4")
171         spacerItem11 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
172         self.gridLayout_4.addItem(spacerItem11, 3, 0, 1, 1)
173         self.label_2 = QtWidgets.QLabel(self.page_log)
174         self.label_2.setObjectName("label_2")
175         self.gridLayout_4.addWidget(self.label_2, 1, 0, 1, 1)
176         spacerItem12 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
177         self.gridLayout_4.addItem(spacerItem12, 4, 0, 1, 1)
178         self.button_back_to_edit_2=QtWidgets.QPushButton(self.page_log)
179         self.button_back_to_edit_2.setObjectName("button_back_to_edit_2")
180         self.gridLayout_4.addWidget(self.button_back_to_edit_2, 6,0,1,1)
181         self.apply_log = QtWidgets.QPushButton(self.page_log)
182         self.apply_log.setObjectName("apply_log")
183         self.gridLayout_4.addWidget(self.apply_log, 5, 0, 1, 1)
184         self.formLayout_2 = QtWidgets.QFormLayout()
185         self.formLayout_2.setSpacing(6)
186         self.formLayout_2.setObjectName("formLayout_2")
187         self.label_3 = QtWidgets.QLabel(self.page_log)
188         self.label_3.setObjectName("label_3")
189         self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_3)
190         self.text_log = QtWidgets.QLineEdit(self.page_log)
191         self.text_log.setObjectName("text_log")
192         self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.text_log)
193         self.gridLayout_4.addLayout(self.formLayout_2, 2, 0, 1, 1)
194         spacerItem13 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
195         self.gridLayout_4.addItem(spacerItem13, 0, 0, 1, 1)
196         self.stackedWidget.addWidget(self.page_log)
197         self.page_gamma = QtWidgets.QWidget()
198         self.page_gamma.setObjectName("page_gamma")
199         self.gridLayout_5 = QtWidgets.QGridLayout(self.page_gamma)
200         self.gridLayout_5.setContentsMargins(11, 11, 11, 11)
201         self.gridLayout_5.setSpacing(6)
202         self.gridLayout_5.setObjectName("gridLayout_5")
203         spacerItem14 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
204         self.gridLayout_5.addItem(spacerItem14, 5, 0, 1, 1)
205         spacerItem15 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
```

```
    .QSizePolicy.Expanding)
206         self.gridLayout_5.addItem(spacerItem15, 4, 0, 1, 1)
207         self.formLayout_3 = QtWidgets.QFormLayout()
208         self.formLayout_3.setSpacing(6)
209         self.formLayout_3.setObjectName("formLayout_3")
210         self.label_4 = QtWidgets.QLabel(self.page_gamma)
211         self.label_4.setObjectName("label_4")
212         self.formLayout_3.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_4)
213         self.text_gamma = QtWidgets.QLineEdit(self.page_gamma)
214         self.text_gamma.setObjectName("text_gamma")
215         self.formLayout_3.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.text_gamma)
216         self.gridLayout_5.addLayout(self.formLayout_3, 3, 0, 1, 1)
217         self.button_back_to_edit_3=QtWidgets.QPushButton(self.page_gamma)
218         self.button_back_to_edit_3.setObjectName("button_back_to_edit_3")
219         self.gridLayout_5.addWidget(self.button_back_to_edit_3, 7,0,1,1)
220         self.apply_gamma = QtWidgets.QPushButton(self.page_gamma)
221         self.apply_gamma.setObjectName("apply_gamma")
222         self.gridLayout_5.addWidget(self.apply_gamma, 6, 0, 1, 1)
223         self.label_5 = QtWidgets.QLabel(self.page_gamma)
224         self.label_5.setObjectName("label_5")
225         self.gridLayout_5.addWidget(self.label_5, 1, 0, 1, 1)
226         spacerItem16 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
227         self.gridLayout_5.addItem(spacerItem16, 0, 0, 1, 1)
228         self.stackedWidget.addWidget(self.page_gamma)
229         self.page_blur = QtWidgets.QWidget()
230         self.page_blur.setObjectName("page_blur")
231         self.gridLayout_6 = QtWidgets.QGridLayout(self.page_blur)
232         self.gridLayout_6.setContentsMargins(11, 11, 11, 11)
233         self.gridLayout_6.setSpacing(6)
234         self.gridLayout_6.setObjectName("gridLayout_6")
235         self.button_back_to_edit_4=QtWidgets.QPushButton(self.page_blur)
236         self.button_back_to_edit_4.setObjectName("button_back_to_edit_4")
237         self.gridLayout_6.addWidget(self.button_back_to_edit_4, 5,0,1,1)
238         self.apply_blur = QtWidgets.QPushButton(self.page_blur)
239         self.apply_blur.setObjectName("apply_blur")
240         self.gridLayout_6.addWidget(self.apply_blur, 4, 0, 1, 1)
241         self.formLayout_4 = QtWidgets.QFormLayout()
242         self.formLayout_4.setSpacing(6)
243         self.formLayout_4.setObjectName("formLayout_4")
244         self.label_6 = QtWidgets.QLabel(self.page_blur)
245         self.label_6.setObjectName("label_6")
246         self.formLayout_4.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_6)
247         self.slider_blur = QtWidgets.QSlider(self.page_blur)
248         self.slider_blur.setMinimum(5)
```

```
249         self.slider_blur.setMaximum(20)
250         self.slider_blur.setSingleStep(2)
251         self.slider_blur.setOrientation(QtCore.Qt.Horizontal)
252         self.slider_blur.setTickPosition(QtWidgets.QSlider.TicksBelow)
253         self.slider_blur.setObjectName("slider_blur")
254         self.formLayout_4.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.slider_blur)
255         self.gridLayout_6.addLayout(self.formLayout_4, 2, 0, 1, 1)
256         self.label_7 = QtWidgets.QLabel(self.page_blur)
257         self.label_7.setObjectName("label_7")
258         self.gridLayout_6.addWidget(self.label_7, 1, 0, 1, 1)
259         spacerItem17 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
260         self.gridLayout_6.addItem(spacerItem17, 0, 0, 1, 1)
261         spacerItem18 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
262         self.gridLayout_6.addItem(spacerItem18, 3, 0, 1, 1)
263         self.stackedWidget.addWidget(self.page_blur)
264         self.page_sharpen = QtWidgets.QWidget()
265         self.page_sharpen.setObjectName("page_sharpen")
266         self.gridLayout_7 = QtWidgets.QGridLayout(self.page_sharpen)
267         self.gridLayout_7.setContentsMargins(11, 11, 11, 11)
268         self.gridLayout_7.setSpacing(6)
269         self.gridLayout_7.setObjectName("gridLayout_7")
270         spacerItem19 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
271         self.gridLayout_7.addItem(spacerItem19, 6, 0, 1, 1)
272         spacerItem20 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
273         self.gridLayout_7.addItem(spacerItem20, 0, 0, 1, 1)
274         self.button_back_to_edit_5=QtWidgets.QPushButton(self.page_sharpen)
275         self.button_back_to_edit_5.setObjectName("button_back_to_edit_5")
276         self.gridLayout_7.addWidget(self.button_back_to_edit_5, 8,0,1,1)
277         self.apply_sharpen = QtWidgets.QPushButton(self.page_sharpen)
278         self.apply_sharpen.setObjectName("apply_sharpen")
279         self.gridLayout_7.addWidget(self.apply_sharpen, 7, 0, 1, 1)
280         self.formLayout_5 = QtWidgets.QFormLayout()
281         self.formLayout_5.setSpacing(6)
282         self.formLayout_5.setObjectName("formLayout_5")
283         ##Adding Radio Button for Sharpen backend here manualy
284         # self.radio_sharpen_1=QtWidgets.QRadioButton(self.page_sharpen)
285         # self.radio_sharpen_1.setText("Use Filter (stable)")
286         # self.radio_sharpen_2=QtWidgets.QRadioButton(self.page_sharpen)
287         # self.radio_sharpen_2.setText("Use FFT (Buggy)")
288         # self.gridLayout_7.addWidget(self.radio_sharpen_1, 1,0,1,1)
289         # self.gridLayout_7.addWidget(self.radio_sharpen_2, 2,0,1,1)
```

```
290
291
292         self.label_8 = QtWidgets.QLabel(self.page_sharpen)
293         self.label_8.setObjectName("label_8")
294         self.formLayout_5.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_8)
295         self.slider_sharpen = QtWidgets.QSlider(self.page_sharpen)
296         self.slider_sharpen.setMaximum(20)
297         self.slider_sharpen.setOrientation(QtCore.Qt.Horizontal)
298         self.slider_sharpen.setObjectName("slider_sharpen")
299         self.formLayout_5.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.slider_sharpen)
300         self.gridLayout_7.addLayout(self.formLayout_5, 4, 0, 1, 1)
301         self.label_9 = QtWidgets.QLabel(self.page_sharpen)
302         self.label_9.setObjectName("label_9")
303         self.gridLayout_7.addWidget(self.label_9, 3, 0, 1, 1)
304         spacerItem21 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets
    .QSizePolicy.Expanding)
305         self.gridLayout_7.addItem(spacerItem21, 5, 0, 1, 1)
306         self.stackedWidget.addWidget(self.page_sharpen)
307         self.display_widget = QtWidgets.QWidget(self.splitter)
308         self.display_widget.setObjectName("display_widget")
309         self.display_layout = QtWidgets.QVBoxLayout(self.display_widget)
310
311         #Manually adding here
312
313
314         if use_matplotlib_backend:
315             #MAtplotlib Backend Here
316             self.display_layout.setContentsMargins(0,0,0,0)
317             self.canvas=MplCanvas(self.display_widget)
318             self.display_layout.addWidget(self.canvas)
319             self.canvas.axis.imshow(iImage.load('no_image.png').RGB)
320             self.canvas.draw()
321             # raise NotImplementedError
322
323         else:
324             self.pixmap= QtGui.QPixmap.fromImage(iImage.load('no_image.png').QImage)
325             self.pixlabel=QtWidgets.QLabel()
326             h,w=self.pixlabel.height(),self.pixlabel.width()
327             self.pixlabel.setScaledContents(True)
328             self.pixmap.scaled(w, h, QtCore.Qt.KeepAspectRatio)
329             self.pixlabel.setSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.
    QSizePolicy.Expanding)
330             self.pixlabel.setPixmap(self.pixmap.scaled(w, h, QtCore.Qt.KeepAspectRatio))
331             self.display_layout.addWidget(self.pixlabel)
332             self.pixlabel.show()
```

```
333
334         #Till here
335         # self.display_layout.addWidget(self.display)
336         self.horizontalLayout.addWidget(self.splitter)
337         self.setCentralWidget(self.centralWidget)
338         self.menuBar = QtWidgets.QMenuBar(self)
339         self.menuBar.setGeometry(QtCore.QRect(0, 0, 858, 22))
340         self.menuBar.setObjectName("menuBar")
341         self.setMenuBar(self.menuBar)
342         self.mainToolBar = QtWidgets.QToolBar(self)
343         self.mainToolBar.setObjectName("mainToolBar")
344         self.addToolBar(QtCore.Qt.TopToolBarArea, self.mainToolBar)
345         self.statusBar = QtWidgets.QStatusBar(self)
346         self.statusBar.setObjectName("statusBar")
347         self.setStatusBar(self.statusBar)
348
349         self.retranslateUi()
350         self.stackedWidget.setCurrentIndex(0)
351         QtCore.QMetaObject.connectSlotsByName(self)
352
353     def retranslateUi(self):
354         #Adding Text to Buttons
355         _translate = QtCore.QCoreApplication.translate
356         self.setWindowTitle(_translate("MainWindow", "MainWindow"))
357         self.button_browse.setText(_translate("MainWindow", "Browse..."))
358         self.button_quit.setText(_translate("MainWindow", "Quit"))
359         self.button_back.setText(_translate("MainWindow", "Back"))
360         self.button_sharpen.setText(_translate("MainWindow", "Sharpen"))
361         self.button_hist.setText(_translate("MainWindow", "Histogram Equalization"))
362         self.button_blur.setText(_translate("MainWindow", "Blur"))
363         self.button_save.setText(_translate("MainWindow", "Save"))
364         self.button_gamma.setText(_translate("MainWindow", "Gamma Correction"))
365         self.button_log.setText(_translate("MainWindow", "Log Transform"))
366         self.apply_hist.setText(_translate("MainWindow", "Apply"))
367         self.label.setText(_translate("MainWindow", "Histogram Equalization"))
368         self.label_2.setText(_translate("MainWindow", "Log Transform"))
369         self.apply_log.setText(_translate("MainWindow", "Apply"))
370         self.label_3.setText(_translate("MainWindow", "Log Base"))
371         self.text_log.setText(_translate("MainWindow", "e"))
372         self.label_4.setText(_translate("MainWindow", "Gamma Value"))
373         self.text_gamma.setText(_translate("MainWindow", "1"))
374         self.apply_gamma.setText(_translate("MainWindow", "Apply"))
375         self.label_5.setText(_translate("MainWindow", "Gamma Correction"))
376         self.apply_blur.setText(_translate("MainWindow", "Apply"))
377         self.label_6.setText(_translate("MainWindow", "Filter Size"))
```

```
        self.label_7.setText(_translate("MainWindow", "Blur"))
        self.apply_sharpen.setText(_translate("MainWindow", "Apply"))
        self.label_8.setText(_translate("MainWindow", "Sharpness"))
        self.label_9.setText(_translate("MainWindow", "Adjust Sharpness"))
        self.button_back_to_edit_1.setText(_translate("MainWindow", "Back"))
        self.button_back_to_edit_2.setText(_translate("MainWindow", "Back"))
        self.button_back_to_edit_3.setText(_translate("MainWindow", "Back"))
        self.button_back_to_edit_4.setText(_translate("MainWindow", "Back"))
        self.button_back_to_edit_5.setText(_translate("MainWindow", "Back"))

    @property
    def gamma(self):
        try:  return float(self.text_gamma.text())
        except:
            self.text_gamma.setText('') #Clear if an Invalid Number
            return self.lastGamma

    @property
    def log(self):
        try: return float(self.text_log.text())
        except:
            self.text_log.setText('') #Clear entry if not a number
            return self.lastLog

    @property
    def sharpness(self): return self.slider_sharpen.value()

    @property
    def blur(self):      return self.slider_blur.value()

    @property
    def hist(self):      return self.slider_hist.value()


    def setupDefaults(self):
        self.lastGamma=1    #deafult Value of Gamma
        self.lastLog = None #Default uses log base e



    def imshow_(self, image):

        if use_matplotlib_backend:
            # raise NotImplementedError
            self.canvas.axis.imshow(image.RGB)
```

```
423             self.canvas.draw()
424         else:
425             self.pixlabel.setPixmap(QtGui.QPixmap.fromImage(image.QImage)) #get the QImage
    object of the iImage class
426             self.pixlabel.show()
427             self.show()
428 
429     def get_file(self): #Get file Name when Browse is clicked
430         fileName, _ = QtWidgets.QFileDialog.getOpenFileName(self, 'Select Image')
431         #print(type(fileName), fileName=='')
432         try:
433             self.image = iImage.load(fileName)
434             self.imshow_(self.image)
435             self.stackedWidget.setCurrentWidget(self.page_edit)
436             self.update_history()
437         except:
438             print(f'Not an Image: {fileName}')
439 
440     def save_image(self):
441         filename,_=QtWidgets.QFileDialog.getSaveFileName(self, "Enter File Name", 'edited.png
    ')
442         if not filename=='': #received a valid non blank file name
443             self.image.save(filename)
444 
445     def update_history(self):
446         self.listWidget.clear()
447         self.listWidget.addItems(self.image.text_history)
448         count=self.listWidget.count()
449         self.listWidget.setCurrentRow(count-1)
450         self.goto_edit()
451         self.show()
452 
453     def goto_edit(self): self.stackedWidget.setCurrentWidget(self.page_edit); self.imshow_(
    self.image)
454 
455     def applyHistToImage(self):     self.imshow_(self.image.histEqualization_(self.hist));
    self.update_history()
456     def applyGammaToImage(self):    self.imshow_(self.image.gammaTransform_(self.gamma)); self
    .update_history()
457     def applyLogToImage(self):      self.imshow_(self.image.logTransform_(self.log)); self.
    update_history()
458     def applyBlurToImage(self):     self.imshow_(self.image.blur_(self.blur)); self.
    update_history()
459     def applySharpenToImage(self): self.imshow_(self.image.sharpen_(self.sharpness)); self.
    update_history()
```

```
460
461     def checkout(self):
462         index=self.listWidget.currentRow()
463         self.imshow_(self.image.checkout(index))
464         self.goto_edit()
465
466     def set_button_bindings(self):
467         self.button_browse.clicked.connect(self.get_file)
468         self.button_quit.clicked.connect(self.close)
469         self.button_hist.clicked.connect(lambda: self.stackedWidget.setCurrentWidget(self.
    page_hist))
470         self.button_log.clicked.connect(lambda: self.stackedWidget.setCurrentWidget(self.
    page_log))
471         self.button_gamma.clicked.connect(lambda: self.stackedWidget.setCurrentWidget(self.
    page_gamma))
472         self.button_blur.clicked.connect(lambda: self.stackedWidget.setCurrentWidget(self.
    page_blur))
473         self.button_sharpen.clicked.connect(lambda: self.stackedWidget.setCurrentWidget(self.
    page_sharpen))
474         self.button_back.clicked.connect(lambda: self.stackedWidget.setCurrentWidget(self.
    page_main))
475         self.button_save.clicked.connect(self.save_image)
476
477         self.apply_hist.clicked.connect(self.applyHistToImage)
478         self.apply_gamma.clicked.connect(self.applyGammaToImage)
479         self.apply_log.clicked.connect(self.applyLogToImage)
480         self.apply_blur.clicked.connect(self.applyBlurToImage)
481         self.apply_sharpen.clicked.connect(self.applySharpenToImage)
482
483         self.slider_blur.sliderReleased.connect(lambda: self.imshow_(self.image.blur(self.
    blur)))
484         self.slider_hist.sliderReleased.connect(lambda: self.imshow_(self.image.
    histEqualization(self.hist)))
485         self.slider_sharpen.sliderReleased.connect(lambda: self.imshow_(self.image.sharpen(
    self.sharpness)))
486         self.text_log.returnPressed.connect(lambda: self.imshow_(self.image.logTransform(self
    .log)))
487         self.text_gamma.returnPressed.connect(lambda: self.imshow_(self.image.gammaTransform(
    self.gamma)))
488
489         self.listWidget.currentRowChanged.connect(self.checkout)
490
491         self.button_back_to_edit_1.clicked.connect(self.goto_edit)
492         self.button_back_to_edit_2.clicked.connect(self.goto_edit)
493         self.button_back_to_edit_3.clicked.connect(self.goto_edit)
```

```
        self.button_back_to_edit_4.clicked.connect(self.goto_edit)
        self.button_back_to_edit_5.clicked.connect(self.goto_edit)



if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.show()
    sys.exit(app.exec_())
```

Thank You