Assignment 1 : Basic Image Editor

Rohit Vartak
Department of Electrical Engineering
190010058
190010058@iitb.ac.in

***Abstract*—We can perform various operations on our images to extract more information from them. For example, we can perform log transformations, power law transformations etc. Images are just arrays with three channels for the 3 colors namely R, G and B and hence we can use the numpy class of python to perform operations on the image. We also develop a GUI in python to stream line the process of transformation of the image.**

***Index Terms*—GUI, image, transformation,**

I. Introduction

The main objective was to perform various operations on images which were taught in class. Since images are at the end of the day just arrays one had to use the vectorization made available to us in python so as to make the operations on the image arrays fast. We perform various operations on our image, for example, we perform Equalize histogram, Gamma correct , Log transform, Blur with a mechanism to control the extent of blurring and Sharpening with a mechanism to control the extent of sharpening.
We also define the convolution function between two arrays as: We can first flatten each array and then multiply and

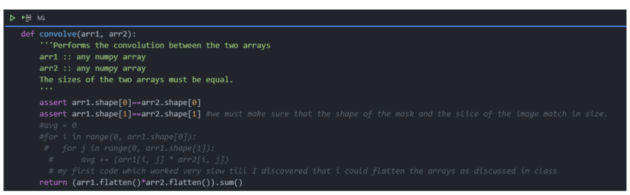

Fig. 1: Convolution

add ,which is much faster than doing it with 2 for loops and element wise multiplication.

II. GUI design

I used tkinter from python to implement the GUI features of the assignment. I have been able to implement the following features in the gui:

1) Image display area
2) Image load button that opens a file selector
3) Manipulate buttons for the following operations:
    - Equalize histogram
    - Gamma correct (ask for input gamma upon pressing the button)
    - Log transform
    - Blur with a mechanism to control the extent of blurring
    - Sharpening with a mechanism to control the extent of sharpening

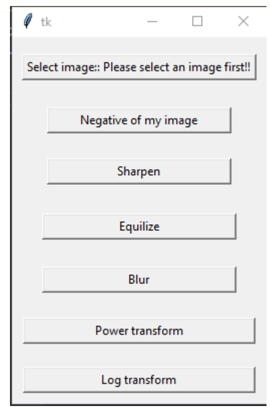

Fig. 2: GUI looks

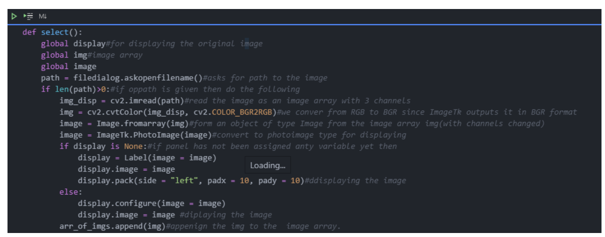

Fig. 3: Image selection block

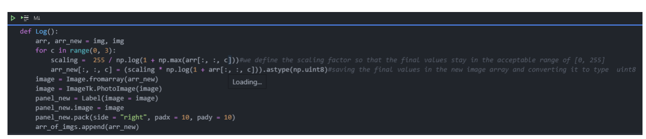

Fig. 4: Log transform block

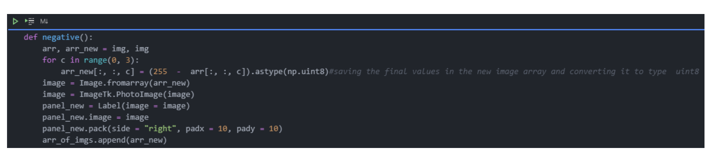

Fig. 5: Negative of an image block

```
def Power():
    c = askfloat("Please enter the value of scale", "c = ")
    g = askfloat("Please enter the value of gamma", "gamma = ")
    arr, arr_new = img, img
    for d in range(0, 3):
        scaling = 255/(c*np.max(arr[:, :, d])**g)#we define the scaling factor so that the final values stay in the acceptable range of [0, 255]
        arr_new = (scaling * c * (arr[: , :, d]**g)).astype(np.uint8)# performing the operation on the image array
    image = Image.fromarray(arr_new)
    image = ImageTk.PhotoImage(image)
    panel_new = Label(image = image)
    panel_new.image = image
    panel_new.pack(side = "right", padx = 10, pady = 10)
    arr_of_imgs.append(arr_new)
```

Fig. 6: Gamma correction block

```
def Blur():
    kernel_size = askinteger("Please enter the value of kernel_size", "kernel_size = ")
    r = kernel_size//2 #integer ddivsionto get the width of the padding
    arr, arr_new = img, img
    mask = np.ones((kernel_size, kernel_size))/sum(sum(np.ones((kernel_size, kernel_size))))#defining the kernel and is normalised so that final values don't exceed 255
    #mask = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    for c in range(0, 3):
        #we now perform the convolution for the image
        #we first pad the image array with zeros of length/width r and then move the mask across the image
        #we use the previously defined convolve function to find the convoltion between the slice of the image array and the mask.
        pad_arr = np.pad(arr[:, :, c], r, pad_with, padder =0)#stores the podded array
        for i in range(0, pad_arr.shape[0] - kernel_size+1):
            for j in range(0, pad_arr.shape[1] - kernel_size+1):
                arr2conv = pad_arr[i:i+kernel_size, j:j+kernel_size]#choosing the slice of the image array
                arr_new[i, j, c] = convolve(arr2conv, mask)#calling the convolve function to convolve the 2
    image = Image.fromarray(arr_new.astype(np.uint8))
    image = ImageTk.PhotoImage(image)
    panel_new = Label(image = image)
    panel_new.image = image
    panel_new.pack(side = "right", padx = 10, pady = 10)
    arr_of_imgs.append(arr_new)
```

Fig. 7: Blurring block

III. Image Processing

A. Image processing operations implemented

We have implemented the following image processes-

- Equalize histogram
- Gamma correct
- Log transform
- Blur with a mechanism to control the extent of blurring
- Sharpening with a mechanism to control the extent of sharpening
- My own feature : Image negative

B. Purpose of the above transformations

- *Equalize histogram* : Helps in improving the contrast of the image.
- *Gamma correct* : It controls the overall brightness of an image.

```
def sharpen():
    kernel_size = askinteger("Please enter the value of kernel_size", "kernel_size = ")
    r = kernel_size//2 #integer ddivsionto get the width of the padding
    arr, arr_new = img, img
    mask = -np.ones((kernel_size, kernel_size))
    mask[r, r] = kernel_size**2 - 1#normal laplacian has the centre element such that the net sum is 0.
    #mask = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    for c in range(0, 3):
        #we now perform the convolution for the image
        #we first pad the image array with zeros of length/width r and then move the mask across the image
        #we use the previously defined convolve function to find the convoltion between the slice of the image array and the mask.
        pad_arr = np.pad(arr[:, :, c], r, pad_with, padder =0)
        for i in range(0, pad_arr.shape[0] - kernel_size+1):
            for j in range(0, pad_arr.shape[1] - kernel_size+1):
                arr2conv = pad_arr[i:i+kernel_ Loading... kernel_size]#choosing the slice of the image array
                arr_new[i, j, c] = convolve(arr2conv, mask)#calling the convolve function to convolve the 2
        #arr_new[:, :, c] = arr_new[: ,:, c] - np.min(arr_new[:, :, c])
        #arr_new[:, :, c] = arr_new[:, :, c]*(255/np.max(arr_new[:, :, c]))
    image = Image.fromarray(arr_new.astype(np.uint8))
    image = ImageTk.PhotoImage(image)
    panel_new = Label(image = image)
    panel_new.image = image
    panel_new.pack(side = "right", padx = 10, pady = 10)
    arr_of_imgs.append(arr_new)
```

Fig. 8: Sharpen image block

```
def Eq():
    arr_og, arr_new = img, img
    for c in range(0, 3):
        #took help from this site https://towardsdatascience.com/histogram-equalization-a-simple-way-to-improve-the-contrast-of-your-image-bcd66596d815
        arr = arr_og[:, :, c]
        hist, _ = np.histogram(arr.flatten(), 256, [0, 256]) #defining the histogram for the given array
        cdf = np.cumsum(hist)#defining the cummulative function or the cdf of the pixels
        arr_new_2d = (cdf - cdf.min())*255/(cdf.max()-cdf.min()) #finding the difference and normalising it so the maximum value is till only 255
        arr_new[:, : , c] = arr_new_2d[arr]#fetching the final image array after equilisation
    arr_of_imgs.append(arr_new)
    image = Image.fromarray(arr_new.astype(np.uint8))
    image = ImageTk.PhotoImage(image)
    panel_new = Label(image = image)
    panel_new.image = image
    panel_new.pack(side = "right", padx = 10, pady = 10)
```

Fig. 9: Historgram equilisation block

- *Log transform* : Log transformation is used for image enhancement as it expands dark pixels of the image as compared to higher pixel values.(since the function is convex in nature.)
- *Blur with a mechanism to control the extent of blur* : Helps in reducing the noise of the image and also reduce the details of the image.
- *Sharpening with a mechanism to control the extent of sharpening* : helps in finding the fine details in an image.

C. Mathematical formulae

1) *Gamma correct*
   Input value of the pixel in the image $= r$
   Output value of the pixel in the image $= s$

$$\mathsf{s} = \mathsf{T(r)} = c \times r^{\gamma} \tag{1}$$

   Note: We need to normalise the values of the above transformation to ensure that the final values remain within the acceptable limits.

2) *log transform*
   Input value of the pixel in the image $= r$
   Output value of the pixel in the image $= s$

$$\mathsf{s} = \mathsf{T(r)} = c \times log(1 + r) \tag{2}$$

   Note: We need to normalise the values of the above transformation to ensure that the final values remain within the acceptable limits.
3) *Blurring*
   For this we use a kernel of all $\frac{1}{(kernel\_size)^2}$.This is done to normalise the kernel, so that the output is always less than 255. Further, we always choose the kernel_size as an odd integer and pad the image array with the appropriate number of rows and columns.
4) *Sharpening*:
   For this we use the Laplace filter. This has the value $kernel\_size^2 - 1$ as the value in the centre and the rest of the value as -1. Notice that the elements of this add up to zero and we run the risk of getting a pixel value negative and thus we need to normalise the final values after convolving by subtracting the minimum and then multiplying with $\frac{1}{Max}$.
5) *Negative*

   Input value of the pixel in the image $= r$
   Output value of the pixel in the image $= s$

$$\mathsf{s} = \mathsf{T(r)} = 255 - r \tag{3}$$

Observation and results

NOTE : *Since I was not able to implement the save functionality, I am forced to take the screenshots of the output from my GUI. I have provided the images in the .zip file for testing and verification!.*

1) *Equalize Histogram* :

Fig. 10: Before histogram equilization

Fig. 11: After histogram equilization

Refer to Figure(10) and (11).Thus, we notice that the latter image has better contrast, i.e it has pixels which are lighter and darker than the original image and this is due to histogram equilisation.

2) *Gamma correct* :
We take the following image: Refer Figures(12 and 13)

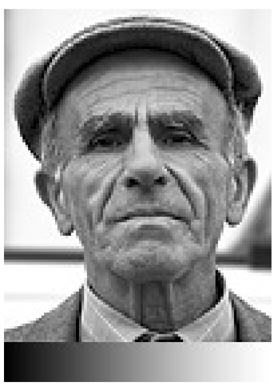

Fig. 12: Before gamma correct

Since $\gamma = 2$, we see that the lighter pixels have been mapped to the darker ones and hence we observe a much darker image for $\gamma > 1$.(referring to figures 12 and 13)

3) *Log transformation* :
We take the following image: Refer Figures(14 and 15)

Thus, since log is convex function we observe that the darker pixels are expanded out while the lighter ones are mapped over a larger range. Thus, we end up compressing the higher pixel values.

4) *Blurring* :
We take the following image:
Referring to Figures (16) and (17), we choose a 3×3 kernel size for our blurriing.

5) *The feature I have added : Negative of an Image* :
Taking the negative of the image from Figure(14), we get the following, refer to figure(18).

6) *Sharpening* :
We take the following image(Refer Figures (19 and 20):

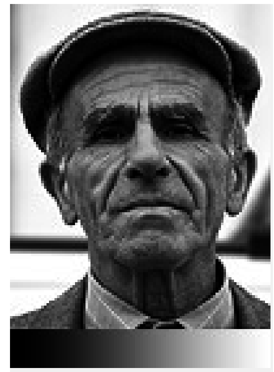

Fig. 13: After gamma correct

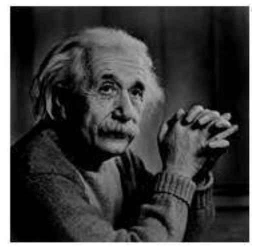

Fig. 14: Before log transformation

Conclusion and discussion

The main challenges I faced was to implement the GUI which did end up taking a lot of time since I was new to Tkinter in python. I was not able to complete many features like undo, undo-all and save in the GUI which I would like to complete. Given more time, I would have loved to play around with the images more and apply some range of parameters to the image transformations like gamma correct etc.

Fig. 15: After log transformation

Fig. 16: Before blurring

Fig. 17: After blurring

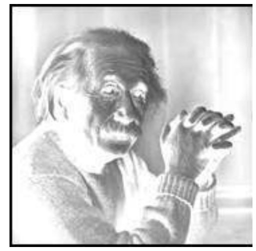

Fig. 18: After negative

Fig. 19: Before sharpening

REFERENCES

[1] https://medium.com/@kyawsawhtoon/a-tutorial-to-histogram-equalization-497600f270e2
[2] Stackoverflow(for-general-doubts)
[3] https://towardsdatascience.com/histogram-equalization-a-simple-way-to-improv
[4] #https://numpy.org/doc/stable/reference/generated/numpy.pad.html
[5] https://www.cambridgeincolour.com/tutorials/gamma-correction.htm
[6] https://www.tutorialspoint.com/dip/gray_level_transformations.htm
[7] http://www.idlcoyote.com/ip_tips/sharpen.html

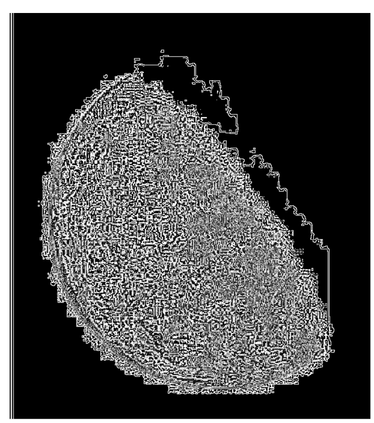

Fig. 20: After Sharpening