Assignment 1: Open CV Introduction

Prepared by:
Adriana Bottega
Computer Engineer Student, Senior

CAD 4410 - Computer Vision
Prof. Muhammad Abid

01/27/2025

Florida Polytechnic University

CONTENTS

CONTENTS ii

LIST OF FIGURES iii

1 INTRODUCTION 1

2 MAKING THE CODE 2

2.1 THOUGHT PROCESS 2

2.2 IMPORTING LIBRARIES 2

2.3 BREAKDOWN OF APP CLASS 3

2.3.1 buildWindow() method 3

2.3.2 Functionalities 6

2.4 TESTING 7

3 PYTHON CODE 9

4 TROUBLESHOOTING 13

REFERENCES 14

LIST OF FIGURES

Figure 2.1 Interface output. 5
Figure 2.2 Different brightness and contrast values tested. 8

1 INTRODUCTION

In this project we were tasked with creating a python program in which we would be able to modify and visualize changes made to the contrast and brightness of an image. To accomplish this task, I used packages for image manipulation, plotting, and GUI creation. The packages I utilized are: OpenCV, matplotlib, tkinter. The firt step is to install the packages required for this project. To be able to use the packages we first need to install them using pip.

To install the needed packages open a new terminal or command prompt window. Next run the following commands:

```
pip install opencv-python
pip install matplotlib}
```

Running `python -m tkinter` from the command line should open a window demonstrating a simple Tk interface, letting you know that tkinter is properly installed on your system.

2 MAKING THE CODE

2.1 THOUGHT PROCESS

While I developed the given code, I started noticing patterns. To manage the pattern I created functions for said patterns. As I progressed through the project I took inspiration on advanced uses of tkinter where they create the window as a class and implement the features as methods of the class. I decided to adapt my program to a class as it fit my image of what it should look like.

The class app which creates the program has the following properties: the original image (img1), the preview image (img2), the brightness property (bri), the contrast property (con),and a list of the canvases within the window (canvases). It also contains the following methods: buildWindow(), which builds the grid inside the window and arranges the elements inside it, brightness(), function to change the brightness contrast of the image, contrast() to change the contrast of the image, update() to refresh the image on the window to mirror the changes, and save to overwrite the original image with the modified image.

2.2 IMPORTING LIBRARIES

To import libraries in the code we run import {library-name}. In this project, I import OpenCV, matplotlib, and tkinter along with sub-libraries as needed.

```
1 import cv2
2 import matplotlib as plt
3 plt.use("TkAgg")
4 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
5 from matplotlib.figure import Figure
6 import tkinter as tk
```

cv2 is an open-source library for computer vision and machine learning. matplotlib enables us to perform diffent type of graphs and grapg manipulation. matplotlib.backends.backend_tkagg is imported for embedding matplotlib plots into tkinter windows. matplotlib.figure allows us to manipulate the image inside of the canvas. Finally, tkinter facilitates making an interface for our program.

2.3 BREAKDOWN OF APP CLASS

The first thing that happens in the program is creating the app object and maintaining the program on the window with the mainloop() function.

```
129 app = App()
130 app.mainloop()
```

This call launches the initiation function `__init__()` in the app class. The App class inherits tk.Tk from tkinter to be able to build the GUI. In the initiation phase the image is loaded into the variable img1 which is then duplicated into the img2 variable and the variables bri and con are set to their default values. Then the layout of the interface is organized and the buildWindow() method is called to populate the interfce with the images and histograms

```
7  class App(tk.Tk):
8      def __init__(self, *args, **kwargs):
9
10         tk.Tk.__init__(self, *args, **kwargs)
11         tk.Tk.wm_title(self, "Assignment 1")
12
13         self.img1 = cv2.imread(r'Assignment1/dog.bmp', cv2.IMREAD_GRAYSCALE)
14         self.img2 = self.img1.copy()
15
16         self.bri = 0 #brightness deafult
17         self.con = 1 ##contrast default
18
19         #organizing the frames on the grid
20         for i in range(3):
21             self.columnconfigure(i, weight=1, minsize=100)
22             if i != 2:
23                 self.rowconfigure(i, weight=1, minsize=50)
24         #build the window
25         self.buildWindow()
```

2.3.1 buildWindow() method

The buildWindow method in the App class is were most of the labor is done. First, it creates and sorts the frames that will contain the canvas for the image/histogram figure into the main window grid and adds them to a list for future use. It also creates the frames and widgets for the scales or scroll bars and the button with their respective parameters. For example, the first parameter in br is its parent, the second and their is the range that the scroll bar has, the orient parameter in in which orientation the scroll bar is, label is the scroll bar's name, command is the action to be triggered once the value of the scroll bar changes, and finally the length is what width should

the scroll bar have. The method pack() places the scroll bar in its parent (scaleFrame) as a block with expanding to be true and for it to fill the parent on the x axis. Following this the save button is created and placed in the grid layout. This button would perform the overwriting of the original image with the preview image

```
def buildWindow(self):
        #container for the images frames
        frames = []

        #frames for histograms and pictures
        for i in range(4):
            fr = tk.Frame(self, padx=5, pady=5)
            fr.grid(row=i//2,column=i%2, pady=5, padx=5)
            frames.append(fr)

        #frame for the scrollbars
        scaleFrame = tk.Frame(self, padx=5, pady=5)
        scaleFrame.grid(row=0,column=2)

        #creating the scrollbars
        br = tk.Scale(scaleFrame, from_=-100, to=100, orient=tk.HORIZONTAL,
                label= "Brightness", command= self.brightness, length=200)
        br.pack(expand=True, fill="x")
        con = tk.Scale(scaleFrame, from_=-100, to=100, orient=tk.HORIZONTAL,
                label= "Contrast", command= self.contrast, length=200)
        con.set(-33)
        origLabel = tk.Label(self, text= "Original Image", font=("Arial", 30, "bold"), pady=10)
        previewLabel = tk.Label(self, text= "Preview Image", font=("Arial", 30, "bold"), pady= 10)
        origLabel.grid(row= 0, column=0, sticky= "N")
        previewLabel.grid(row=0, column=1, sticky= "N")

        #Creating the button save and giving it a command and placement
        save = tk.Button(self,text= "SAVE",command= lambda: self.save())
        save.grid(row=1, column=2)
```

Continuing, it creates Figure objects, subplots for the Axes of the Figure object and either displays the data as an image or as a histogram according to the i and j values. This ensures that there will be a figure and histogram for the original image and another figure and histogram for the preview. After this the canvas is created with the figure inside it and packed inside its frame. The canvas for each Axes is then appended into the canvases list. Once the program exits the j loop it pops the first value of canvases, signaling that img1 has completed processing and continues to processing img2. At this point, the layout is finished. Two images, two histograms, two scroll bars to adjust contrast and brightness, and a save button to overwrite the original image with the preview image. This signals the end of the initiation stage. Now according to what functionality is triggered, the rest of the methods will be called. The finalized interface can be seen in 2.1.

```
        #container for the image's canvases
        self.canvases = [self.img1, self.img2]

        #adding the images on the canvases
        for i in range(2):
            for j in range(2):
                fig = Figure(figsize=(3,3))
                a = fig.add_subplot(111)
                if i==0:
                    #this one for the pictures
                    a.imshow(self.canvases[0], cmap= 'gray')
                    a.axis(False)
                else:
                    #this one for the histograms
                    a.hist(self.canvases[0].ravel(), bins=256, range=[0, 256])

                #creating and embedding the images on the canvases, and
                #the canvases on the frames
                canvas = FigureCanvasTkAgg(fig, frames[(2*i)+j])
                canvas.draw()
                canvas.get_tk_widget().pack()

                #adding the canvases into the canvases list for future use
                self.canvases.append(canvas)
            #removing the images used
            self.canvases.pop(0)
```

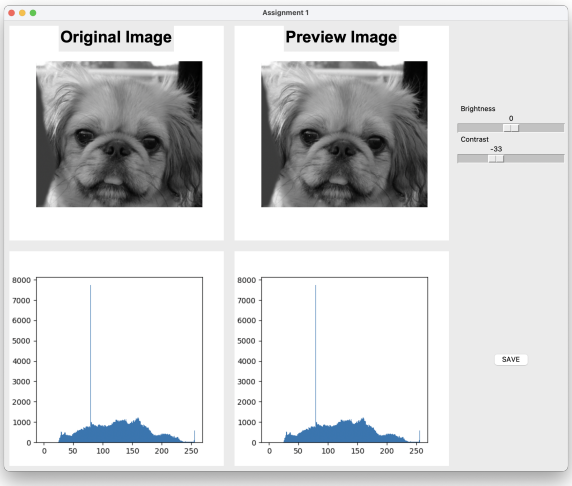


Figure 2.1. In this figure we can see the original picture and its histogram alongside the preview picture and its histogram. Beside them we have the two scroll-bars and the save button.

2.3.2 Functionalities

Once either the brightness scroll-bar or the contrast scroll-bar are used, it executes the command specified in their respective options. For the contrast is contrast() and for brightness is brightness(). Both of them have a similar process. Regarding their differences, contrast has a conditional that if the value of the contrast scroll-bar as well as the brightness scroll-bar are at their initial position then the histogram showed will be that of the original image. The reason for this will be discussed in the Troubleshooting section. Their constants bri and con are scaled from the var value using linear scaling formula. Then the modified image is sent to the function update()

```
81      #whenever we increase or decrease the brightness
82      def brightness(self, var):
83          self.bri = ((int(var)-(-100))/(100-(-100)))*(127-(-127))+(-127) #scaling to have a -127 to 127
84          modified = cv2.convertScaleAbs(self.img2, alpha=self.con, beta=self.bri) #converting the pictur
85          self.update(modified) #updating with modified value
86
87
88      #command function to increase or decrease the brightness
89      def contrast(self, var):
90          if int(var) == -33 and self.bri == 0:
91              self.con = ((int(var)-(-100))/(100-(-100)))*3
92              modified = self.img1
93          else:
94              self.con = ((int(var)-(-100))/(100-(-100)))*3 #scaling from -100 to 100 to be 0 to 3
95              print(self.con)
96              modified = cv2.convertScaleAbs(self.img2, alpha= self.con, beta= self.bri)
97          self.update(modified)
```

The next method is update. The update method receives the modified image and refreshes the canvas inside the GUI to reflect the changes made through the scroll-bar.

```
98  #updates the image with the modified image
99      def update(self, img):
100         for i in range(1,4,2):
101             fig = self.canvases[i].figure
102             fig.clear()
103             a = fig.add_subplot(111)
104
105             if i==1:
106                 a.imshow(img, cmap='gray')
107                 a.axis('off')
108             else:
109                 a.hist(img.ravel(), bins=256, range=[0, 256])
110             self.canvases[i].draw()
```

Finally, the save method generates an image with the contrast and brightness currently defined and overwrites the original image.

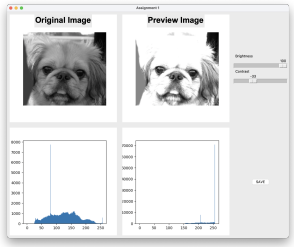
```
111     #saving the image, replacing the original one
112     def save(self):
113         modified = cv2.convertScaleAbs(self.img2, alpha=self.con, beta=self.bri)
114         cv2.imwrite(r"Assignment1/dog.bmp", modified)
```

2.4 TESTING

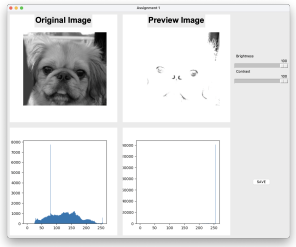
To demonstrate the utility of the scroll-bars and save button I tested different values of brightness and contrast show in Figure 2.2. When modifying the image to have high brightness, the entire image becomes lighter. Shadows, mid-tones, and highlights all shift toward white(Fig. 2.2a). When both the brightness and contrast of the image are raised, the image appears more vivid, with greater distinction between bright and dark areas (Fig. 2.2b). However, when only the contrast is raised, the image becomes sharper and more dramatic. Details in shadows and highlights are emphasized(Fig. 2.2c).

In contrast, when decreasing the brightness, the entire image becomes darker. Shadows, mid-tones, and highlights all shift toward black (Fig. 2.2d). When both brightness and contrast are low, the image becomes darker and flatter, with less distinction between light and dark areas. In my case, the image had no distinction (Fig. 2.2e). Similarly to the previous case, when decreasing only contrast, the image showed no distinction between white and dark areas (Fig. 2.2f).

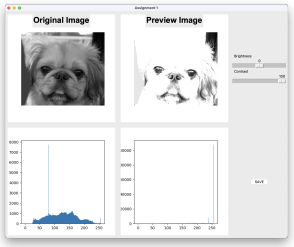
When mixing it up and decreasing brightness but lowering the contrast, the image shows stark distinctions between bright and dark areas (Fig. 2.2g). I finally tested custom values (Fig. 2.2h) to demonstrate that once the save button is pressed then the preview image overwrites the original image (Fig. 2.2i).



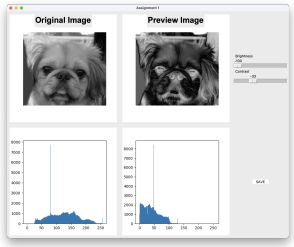
(a) The figure shows the preview of what the image would look like with high brightness.



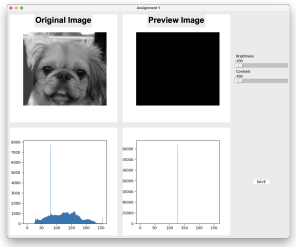
(b) The figure shows the preview of what the image would look like with high brightness and high contrast.



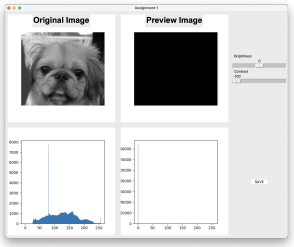
(c) The figure shows the preview of what the image would look like with high contrast.



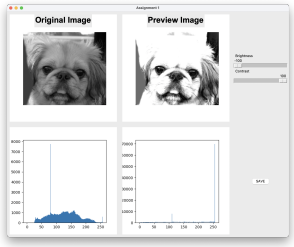
(d) The figure shows the preview of what the image would look like with low brightness.



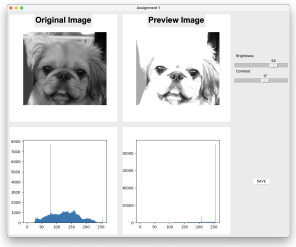
(e) The figure shows the preview of what the image would look like with low brightness and low contrast.



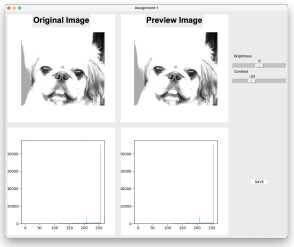
(f) The figure shows the preview of what the image would look like with low contrast.



(g) The figure shows the preview of what the image would look like with low brightness and high contrast.



(h) The figure shows the preview of what the image would look like with custom values.



(i) The figure what happens when you save an image, in this case fig. 2.2(h).

Figure 2.2. In figures a-i, I tested different values of brightness and contrast to show the output of the preview image and functionality of the scroll-bars and save button

3 PYTHON CODE

```
    import cv2
import matplotlib as plt
plt.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
from matplotlib.figure import Figure
import tkinter as tk

class App(tk.Tk):
    def __init__(self, *args, **kwargs):

        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "Assignment 1")
        self.defaultbg = self.cget('bg')

        self.img1 = cv2.imread(r'Assignment1/dog.bmp', cv2.IMREAD_GRAYSCALE)
        self.img2 = self.img1.copy()

        self.bri = 0 #brightness deafult
        self.con = 1 ##contrast default

        #organizing the frames on the grid
        for i in range(3):
            self.columnconfigure(i, weight=1, minsize=70)
            if i != 2:
                self.rowconfigure(i, weight=1, minsize=50)
        #build the window
        self.buildWindow()


    #function to create the histogram and image to embed in window
    def buildWindow(self):
        #container for the images frames
```

```
        frames = []

        #frames for histograms and pictures
        for i in range(4):
            fr = tk.Frame(self, padx=5, pady=5)
            fr.grid(row=i//2,column=i%2, pady=5, padx=5)
            frames.append(fr)

        #frame for the scrollbars
        scaleFrame = tk.Frame(self, padx=5, pady=5)
        scaleFrame.grid(row=0,column=2)

        #creating the scrollbars
        br = tk.Scale(scaleFrame, from_=-100, to=100, orient=tk.HORIZONTAL, label= "Brightness",
                      command= self.brightness, length=200)
        br.pack(expand=True, fill="x")
        con = tk.Scale(scaleFrame, from_=-100, to=100, orient=tk.HORIZONTAL, label= "Contrast",
                       command= self.contrast, length=200)
        con.pack(expand=True, fill="x")
        con.set(-33)

        origLabel = tk.Label(self, text= "Original Image", font=("Arial", 30, "bold"), pady=10)
        previewLabel = tk.Label(self, text= "Preview Image", font=("Arial", 30, "bold"), pady= 10)
        origLabel.grid(row= 0, column=0, sticky= "N")
        previewLabel.grid(row=0, column=1, sticky= "N")


        #Creating the button save and giving it a command and placement
        save = tk.Button(self,text= "SAVE",command= lambda: self.save())
        save.grid(row=1, column=2)

        #container for the image's canvases
        self.canvases = [self.img1, self.img2]

        #adding the images on the canvases
        for i in range(2):
            for j in range(2):
                fig = Figure(figsize=(4,4))
                a = fig.add_subplot(111)
                if i==0:
                    #this one for the pictures
                    a.imshow(self.canvases[0], cmap= 'gray')
                    a.axis(False)
                else:
```

```
                    #this one for the histograms
                    a.hist(self.canvases[0].ravel(), bins=256, range=[0, 256])

                #creating and embedding the images on the canvases, and the canvases on the frames
                canvas = FigureCanvasTkAgg(fig, frames[(2*i)+j])
                canvas.draw()
                canvas.get_tk_widget().pack()

                #adding the canvases into the canvases list for future use
                self.canvases.append(canvas)
            #removing the images used
            self.canvases.pop(0)

    #whenever we increase or decrease the brightness
    def brightness(self, var):
        self.bri = ((int(var)-(-100))/(100-(-100)))*(127-(-127))+(-127) #scaling to have a -127 to 127
        modified = cv2.convertScaleAbs(self.img2, alpha=self.con, beta=self.bri) #converting the pictur
        self.update(modified) #updating with modified value


    #command function to increase or decrease the brightness
    def contrast(self, var):
        if int(var) == -33 and self.bri == 0:
            self.con = ((int(var)-(-100))/(100-(-100)))*3
            modified = self.img1
        else:
            self.con = ((int(var)-(-100))/(100-(-100)))*3 #scaling from -100 to 100 to be 0 to 3
            print(self.con)
            modified = cv2.convertScaleAbs(self.img2, alpha= self.con, beta= self.bri)
        self.update(modified)

    #updates the image with the modified image
    def update(self, img):
        for i in range(1,4,2):
            fig = self.canvases[i].figure
            fig.clear()
            a = fig.add_subplot(111)

            if i==1:
                a.imshow(img, cmap='gray')
                a.axis('off')
            else:
                a.hist(img.ravel(), bins=256, range=[0, 256])
            self.canvases[i].draw()
```

```
121
122
123     #saving the image, replacing the original one
124     def save(self):
125         modified = cv2.convertScaleAbs(self.img2, alpha=self.con, beta=self.bri)
126         cv2.imwrite(r"Assignment1/dog.bmp", modified)
127
128
129 app = App()
130 app.mainloop()
```

4 TROUBLESHOOTING

During the developing of the program I had varied and different type of errors ranging from logical errors to syntax error. The first step at solving them would be analyze the line of code the error pointed at. From then on I would do trouble shooting and decide if it was a logic error and how to fix the error or if it was a bad use of a function, etc.

One of the first errors I encountered was manipulating an image that wasn't in grayscale. After realizing that it was easier to manipulate the image in grayscale, I began working on the GUI. This was followed by a lot of error due to misuse of the library by my part. Luckily, most if not all of the error had an easy fix due to tkinter documentation.

The most persistent error caused the preview's histogram to be different than the original even when both were practically the same file. It took me a while to realize that it was because the value calculated by the linear scaling equation never equated to the actual contrast value of the original image. Which is why I added a conditional inside of the function that modifies the images contrast to send the original image as the modified image when the scroll-bar's value was the initial one.

REFERENCES

(2023). "Changing the contrast and brightness of an image using python – opencv." *GeeksForGeeks*.

E-Paine (2021). "tkinter-docs." *tkinter-docs*.

Gee (2023) E-Paine (2021)