EE568 Digital Image Processing

Final Project

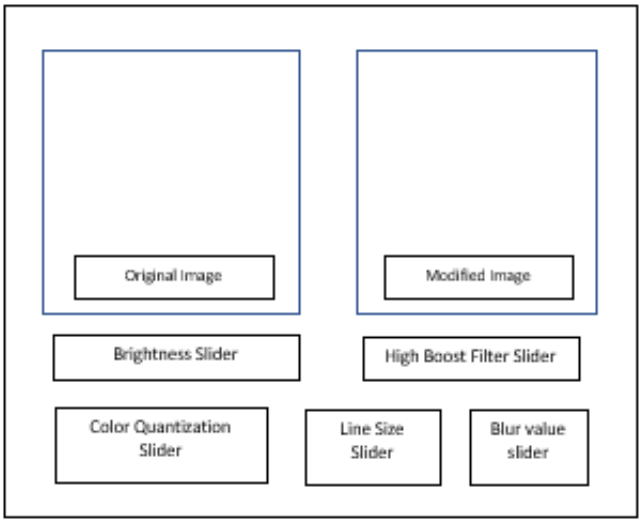
Linda Yen

Objective

The objective of this project is to create a Graphical User Interface (GUI) that allows the user to apply different filtering effects to an image. The user should be able to see the changes to the images real-time.

Background

In this project, I will be using the Python GUI package, tkinter, to create a GUI to alter images using different types of filters. The GUI layout will look like the below:



How to Run Program

In order to run this program, download “Final_Project_Linda_Yen.zip” onto your desktop.

You can add any images into the folder, preferably with a size of 500x300 to get the best result, but the images used in this project are in the file.

In order to change select the image you want to use in the program, change the *image_path* value (**“town.jpg”**) to the image file you want to use:

```
def  __init__(self, window, window_title, image_path = 'town.jpg'):
```

Once the files have been saved onto your desktop, run the program “Final_Project.py”. You will notice there will be two images. The left image will remain the same for

comparison purposes. The right image can be altered using the sliders. The Brightness Slider, High Boost Filter Slider, and the three sliders (Color Quantization, Line Size, and Blur Value) are independent of each other. Meaning if you adjusted the brightness of the image and then adjusted the High Boost Filter slider, the image will reset to the original before applying the High Boost Filter. The three sliders for the cartoon effect are dependent on each other.

Implementation

The initial skeleton of the code was taken from the class “tKinter Demo”. I kept most of the original layout of the Original and Modified Images. I added a few sliders for different filters that I will discuss in each section below.

Brightness

The first image effect I added was a brightness adjuster function. The function takes in an image and a value from the slider. The function first converts the RGB image into the HSV color space. Then it splits the image into its Hue, Saturation, Value components. Then to increase brightness of the image, the inputted value from the slider is added to the pixel value. In order to prevent an overflow situation, the pixel is saturated at 255. The H, S, V components are then merged and converted back into the RGB colors space.
Below is the code snippet for the slider:

```
# Create a SCALE that lets the user adjust the brightness of the image
self.brightness_slider = tk.Scale(self.frame2, from_=1, to=255,
orient=tk.HORIZONTAL, showvalue=1, resolution = 1,command =
self.adjust_brightness, length=300, sliderlength=20, label="Brigthness",
font="Tahoma 9")

self.brightness_slider.place(relx=0.05, rely=0.02, relwidth=0.4,
relheight=0.35)
```

Below is the code snippet for the adjust brightness function:

```
def adjust_brightness(self, value):
            value = self.brightness_slider.get()  # get value from the
corresponding scale
            hsv = cv2.cvtColor(self.NEWcv_img, cv2.COLOR_RGB2HSV)
            h, s, v = cv2.split(hsv)
```

```
            lim = 255 - value
            v[v > lim] = 255
            v[v <= lim] += value

            final_hsv = cv2.merge((h, s, v))
            NEWcv_img_modify = cv2.cvtColor(final_hsv,  cv2.COLOR_HSV2RGB)
            self.photo = PIL.ImageTk.PhotoImage(image =
PIL.Image.fromarray(NEWcv_img_modify))
            self.canvas1.create_image(MAXDIM//2, MAXDIM//2,
image=self.photo, anchor=tk.CENTER)
```

High Boost Filter

The next image effect I added was a High Boost Filter function. A High Boost Filter is used to sharpen the high frequency components of an image while still keeping the lower frequency components of the image. The High Boost Filter function takes in an image and value A to add to the center value of the kernel. The kernel used in this implementation is:

-1	-1	-1
-1	A + 8	-1
-1	-1	-1

The A value is taken from a slider with a range between 1 and 3. The slide code snippet is below:

```
# Create a SCALE that lets the user apply a High Boost Filter to the image
            self.hbf_slider = tk.Scale(self.frame2, from_= 0, to=2,
orient=tk.HORIZONTAL, resolution=0.1, showvalue=1, command =
self.highBoostFilter, length=100, sliderlength=20, label="High Boost
Filter", font="Tahoma 9")
            self.hbf_slider.place(relx=0.5, rely=0.02, relwidth=0.2,
relheight=0.35)
```

Using cv2.filter2D(), the image is convoluted with the kernel. Below is the code snippet for the function:

```
def highBoostFilter(self, A):
            #Kernel that will slide over the image (convolution)
```

```
            A = self.hbf_slider.get()  # get value from the corresponding
scale
            kernel = np.array([[-1 , -1 , -1] , [-1 , 8+A, -1] ,[-1 , -1 ,
-1]])
            NEWcv_img_modify = cv2.filter2D(self.NEWcv_img, -1 , kernel =
kernel)
            self.photo = PIL.ImageTk.PhotoImage(image =
PIL.Image.fromarray(NEWcv_img_modify))
            self.canvas1.create_image(MAXDIM//2, MAXDIM//2,
image=self.photo, anchor=tk.CENTER)
```

Cartoon Image Filter

The last image effect I added was turning the image into a cartoon/drawing. This required a few functions.

The first function used is the edge_mask function. This function first converts the image into a gray image. Then using a median blur filter (cv2.medianBlur), noise is removed from the image since adaptive thresholding is susceptible to noise. The amount of blur can be adjusted using the Edge Blur Value Slider. Then adaptive thresholding (cv2.adaptiveThreshold) is applied to the blurred image to create an edge mask. Below is the code snippet for the function:

```
def edge_mask(img, line_size, blur_value):
                    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
                    gray_blur = cv2.medianBlur(gray, blur_value)
                    edges = cv2.adaptiveThreshold(gray_blur, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, line_size, C = 2)
                    return edges
```

The second function used is the color_quantization function. This function is used to reduce the amount of distinct colors in the image, since drawings have less color. First, the image is turned into data. The image is then segmented using the K-means clustering algorithm. In order to determine when the algorithm is terminated, a criteria, which is the maximum number of iterations in this case, is defined using cv2.TERM_CRITERIA_EPS and cv2.TERM_CRITERIA_MAX_ITER. In this project, the maximum number of iterations is set to 20 with an epsilon of 0.001. The K-Means clustering algorithm is applied using cv2.kmeans. The k value is inputted using the Color Quantization slider. By increasing the k value, the number of clusters identified by

the algorithm from the image is identified. Below is the code snippet for color quantization:

```
def color_quantization(img, k):
                    # Transform the image
                    data = np.float32(img).reshape((-1, 3))

                    # Determine criteria
                    criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 20, 0.001)

                    # Implementing K-Means
                    ret, label, center = cv2.kmeans(data, k, None, criteria,
10, cv2.KMEANS_RANDOM_CENTERS)
                    center = np.uint8(center)
                    result = center[label.flatten()]
                    result = result.reshape(img.shape)
                    return result
```

The next function used in the Bilateral Filter function. cv2.bilateralFilter() is used to reduce the noise in the colored image after color quantization. The code snippet for this function is below:

```
def bilateral_filter(img):
                    blurred = cv2.bilateralFilter(img, d=7,
sigmaColor=100,sigmaSpace=200)
                    return blurred
```

The last function used is the make_cartoon_img. This function is used to combine the edge mask image and the color quantized, noise reduced image. The two images are combined using cv2.bitwise_and(). Below is the code snippet for the function (I removed the other functions for clarity):

```
def make_cartoon_img(self):
.........
.........
image_edges = edge_mask(self.NEWcv_img, self.line_size, self.blur_value)
            image_reduced_color = color_quantization(self.NEWcv_img,
self.color_quantization)
            image_reduced_noise = bilateral_filter(image_reduced_color)

            cartoon = cv2.bitwise_and(image_reduced_noise,
image_reduced_noise, mask = image_edges)
```

```
            self.photo = PIL.ImageTk.PhotoImage(image =
PIL.Image.fromarray(cartoon))
            self.canvas1.create_image(MAXDIM//2, MAXDIM//2,
image=self.photo, anchor=tk.CENTER)
```

NOTE: Since line_size and blur_value must be odd values, whenever an even value is selected on the slider, one is subtracted from that value. Example, if a 4 is selected on the slider for line_size, the actual input into the function is 3.

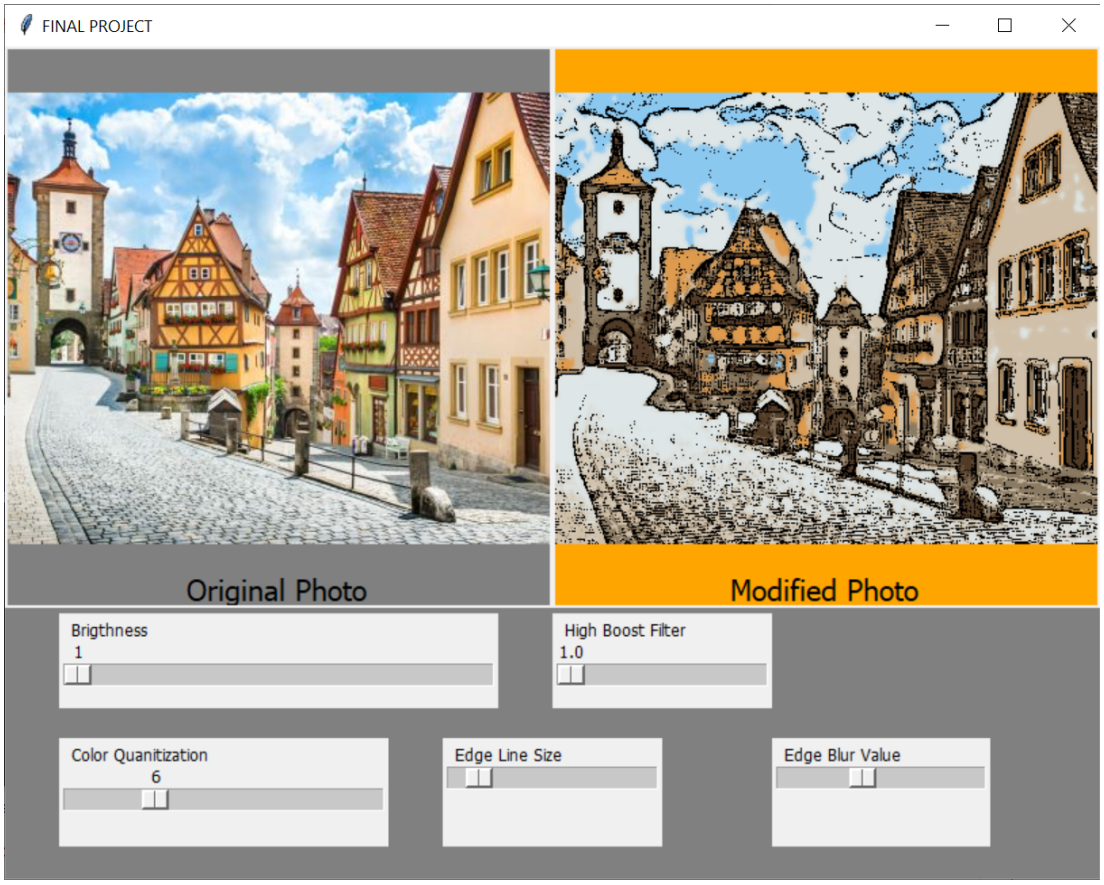
Results

Below is an image, “baseball.jpg”, that has been modified with the brightness slider (slide value set to 83):

Below is the result of the High Boost Filtering on the image, “lena.bmp” with A = 1.4:

Below is the result of the make_cartoon_img on the image “town.jpg” with Color Quantization k = 6, line size = 3, blur value = 7:

Below is the result of the overall GUI:



Lessons Learned

Overall, this project was really interesting. I first started with simple effects on the image, such as increased brightness and sharpened images. The cartoon effect was really fun to play around with as well. Although the color quantization function wasn't really necessary for the cartoon effect, it definitely augmented the look and feel. This was the first time I implemented a program to apply a mask on an image. I also learned a lot about using tkinter and how a class and method works in python. I think the GUI could have been more neatly placed together since my sliders are just spread across frame2. Another feature I would like to add in the future is a more user-friendly GUI that allows a user to upload an image from their desktop instead of having to change the image path directly in the code.

References

- K-Means Clustering for Image Classification, https://medium.com/@joel_34096/k-means-clustering-for-image-classification-a648f28bdc47
- Make your Photos Look Trippy! Build a Photo Filter From Scratch with Python, https://medium.com/@fidel.esquivelestay/make-you-photos-look-trippy-with-python-how-to-build-your-own-photo-filter-app-from-scratch-8ef4f2703282
- Understanding Brightness in an Image, https://www.geeksforgeeks.org/opencv-understanding-brightness-in-an-image/
- How to create a cool cartoon effect with OpenCV and Python, https://www.askaswiss.com/2016/01/how-to-create-cartoon-effect-opencv-python.html
- https://www.tutorialspoint.com/python/
- https://docs.opencv.org/

Appendix A

```
Below is the code snippet for the whole project:
from tkinter import filedialog
import tkinter as tk
import cv2
import PIL.Image, PIL.ImageTk
import numpy as np

MARGIN = 5
MAXDIM = 400

class App():
      #specify path of image files
      def  __init__(self, window, window_title, image_path = 'town.jpg'):
            #Initialize values for Methods
            self.color_quantization = 1
            self.line_size = 3
            self.blur_value = 3

            ################ CREATE GUI #################
            self.window = window
            self.window.title(window_title)

            # Using cv2.imread() method
            img = cv2.imread(image_path)

            #Load an image using OpenCV
            self.cv_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            self.NEWcv_img = self.cv_img.copy()

            #Get the image dimensions
            self.height, self.width, channels = self.cv_img.shape
            print('height', self.height)
            print('width', self.width)

            #Create a FRAME
            self.frame1 = tk.Frame(self.window, width=self.width,
height=self.height, bg='gray')
            self.frame1.pack(fill=tk.BOTH)

            #Create a CANVAS for original image
```

```
            self.canvas0 = tk.Canvas(self.frame1, width=MAXDIM,
height=MAXDIM+(2*MARGIN), bg='gray')
            self.canvas0.pack(side=tk.LEFT)

            #Create a CANVAS for changing image
            self.canvas1 = tk.Canvas(self.frame1, width=MAXDIM,
height=MAXDIM+(2*MARGIN), bg='orange')
            self.canvas1.pack(side=tk.RIGHT)

            # Use PIL (Pillow) to convert the NumPy ndarray to a PhotoImage
            self.photoOG =
PIL.ImageTk.PhotoImage(image=PIL.Image.fromarray(self.cv_img))
            self.photo =
PIL.ImageTk.PhotoImage(image=PIL.Image.fromarray(self.cv_img))

        # Add a PhotoImage to the Canvas (original)
            self.canvas0.create_image(MAXDIM//2, MAXDIM//2,
image=self.photoOG)

        # Add a PhotoImage to the Canvas (changing effects)
            self.canvas1.create_image(MAXDIM//2, MAXDIM//2,
image=self.photo, anchor=tk.CENTER)

        # Write labels for both images, font/size can be changed
            self.canvas0.create_text(MAXDIM//2, MAXDIM,font="Tahoma
16",text="Original Photo")
            self.canvas1.create_text(MAXDIM//2, MAXDIM,font="Tahoma
16",text="Modified Photo")

#
#####################################################################################
###################
# ################################   PARAMETER TOOLBAR
#######################################
#
#####################################################################################
###################

        # Create a FRAME that can fit the below features
            self.frame2 = tk.Frame(self.window, width=self.width,
height=200, bg='gray')
            self.frame2.pack(side=tk.BOTTOM, fill=tk.BOTH)
```

```
            # Create a SCALE that lets the user adjust the brightness of
the image
            self.brightness_slider = tk.Scale(self.frame2, from_=1, to=255,
orient=tk.HORIZONTAL, showvalue=1, resolution = 1,command =
self.adjust_brightness, length=300, sliderlength=20, label="Brigthness",
font="Tahoma 9")
            self.brightness_slider.place(relx=0.05, rely=0.02,
relwidth=0.4, relheight=0.35)

            # Create a SCALE that lets the user apply a High Boost Filter
to the image
            self.hbf_slider = tk.Scale(self.frame2, from_= 1, to=3,
orient=tk.HORIZONTAL, resolution=0.1, showvalue=1, command =
self.highBoostFilter, length=100, sliderlength=20, label="High Boost
Filter", font="Tahoma 9")
            self.hbf_slider.place(relx=0.5, rely=0.02, relwidth=0.2,
relheight=0.35)

            #Create a SCALE that lets the user change the color
qunantization amount
            self.color_quantization_slider = tk.Scale(self.frame2, from_=
1, to= 20, orient=tk.HORIZONTAL, resolution=1, showvalue=1, command =
self.change_color_quanitization, length=400, sliderlength=20, label="Color
Quanitization", font="Tahoma 9")
            self.color_quantization_slider.place(relx=0.05, rely=0.48,
relwidth=0.3, relheight=0.4)

            #Create a SCALE that lets the user change the edge line size
amount
            self.edge_line_size_slider = tk.Scale(self.frame2, from_= 3,
to= 13, orient=tk.HORIZONTAL, showvalue=0, command =
self.change_edge_line_size, length=400, sliderlength=20, label="Edge Line
Size", font="Tahoma 9")
            self.edge_line_size_slider.place(relx=0.4, rely=0.48,
relwidth=0.2, relheight=0.4)

            #Create a SCALE that lets the user change the edge mask blur
amount
            self.edge_blur_slider = tk.Scale(self.frame2, from_= 3, to= 13,
orient=tk.HORIZONTAL, showvalue=0, command = self.change_edge_blur_value,
length=400, sliderlength=20, label="Edge Blur Value", font="Tahoma 9")
            self.edge_blur_slider.place(relx=0.7, rely=0.48, relwidth=0.2,
relheight=0.4)
```

```
            self.window.mainloop()

##########################################################################
####################
##################################  CALLBACK FUNCTIONS
#######################################
##########################################################################
####################

#'''################################  OTHER FUNCTIONS
################################'''#
    # define other callback functions here
    ############## ADJUST BRIGHTNESS ##############
      def adjust_brightness(self, value):
            value = self.brightness_slider.get()  # get value from the
corresponding scale
            hsv = cv2.cvtColor(self.NEWcv_img, cv2.COLOR_RGB2HSV)
            h, s, v = cv2.split(hsv)

            lim = 255 - value
            v[v > lim] = 255
            v[v <= lim] += value

            final_hsv = cv2.merge((h, s, v))
            NEWcv_img_modify = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2RGB)
            self.photo = PIL.ImageTk.PhotoImage(image =
PIL.Image.fromarray(NEWcv_img_modify))
            self.canvas1.create_image(MAXDIM//2, MAXDIM//2,
image=self.photo, anchor=tk.CENTER)

      ############## APPLY HIGH BOOST FILTERING ##############
      def highBoostFilter(self, A):
            #Kernel that will slide over the image (convolution)
            A = self.hbf_slider.get()  # get value from the corresponding
scale
            kernel = np.array([[-1 , -1 , -1] , [-1 , 8+A, -1] ,[-1 , -1 ,
-1]])
            NEWcv_img_modify = cv2.filter2D(self.NEWcv_img, -1 , kernel =
kernel)
            self.photo = PIL.ImageTk.PhotoImage(image =
PIL.Image.fromarray(NEWcv_img_modify))
            self.canvas1.create_image(MAXDIM//2, MAXDIM//2,
```

```
image=self.photo, anchor=tk.CENTER)

      ############## METHODS ##############
      ### Below methods are used to change inputs for the diffferent
functions in the make_cartoon_img function
      def change_color_quanitization(self, k):
            self.color_quantization = int(k)
            #print('color Quanitization value ', self.color_quantization)
            self.make_cartoon_img()

      def change_edge_line_size(self, k):
            if (int(k) % 2) == 0:
                  k = int(k) - 1
            self.line_size = int(k)
            #print('line size ', self.line_size)
            self.make_cartoon_img()

      def change_edge_blur_value(self, k):
            if (int(k) % 2) == 0:
                  k = int(k) - 1
            self.blur_value = int(k)
            #print('blur value ', self.blur_value)
            self.make_cartoon_img()

      ############## APPLY CARTOON IMAGE ##############
      def make_cartoon_img(self):
            def edge_mask(img, line_size, blur_value):
                  gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
                  gray_blur = cv2.medianBlur(gray, blur_value)
                  edges = cv2.adaptiveThreshold(gray_blur, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, line_size, C = 2)
                  return edges


            def color_quantization(img, k):
                  # Transform the image
                  data = np.float32(img).reshape((-1, 3))

                  # Determine criteria
                  criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 20, 0.001)

                  # Implementing K-Means
```

```
                ret, label, center = cv2.kmeans(data, k, None, criteria,
10, cv2.KMEANS_RANDOM_CENTERS)
                center = np.uint8(center)
                result = center[label.flatten()]
                result = result.reshape(img.shape)
                return result

            def bilateral_filter(img):
                blurred = cv2.bilateralFilter(img, d=7,
sigmaColor=100,sigmaSpace=200)
                return blurred

            image_edges = edge_mask(self.NEWcv_img, self.line_size,
self.blur_value)
            image_reduced_color = color_quantization(self.NEWcv_img,
self.color_quantization)
            image_reduced_noise = bilateral_filter(image_reduced_color)

            cartoon = cv2.bitwise_and(image_reduced_noise,
image_reduced_noise, mask = image_edges)
            self.photo = PIL.ImageTk.PhotoImage(image =
PIL.Image.fromarray(cartoon))
            self.canvas1.create_image(MAXDIM//2, MAXDIM//2,
image=self.photo, anchor=tk.CENTER)
################################################################################
###################
# Create a window and pass it to the Application object

App(tk.Tk(), "FINAL PROJECT")
```