**Student ID: kap675  
Student Name: Kundan Patel  
Course: CS GY 6643 (Computer Vision)   
Project 1**

**Instructions for executing the code:**

1. Make a new folder ‘img’ in the same directory as the file ‘otsu.py’ exists.
2. Add the input images to the img folder.
3. Open the terminal and navigate to the directory that contains the source code.
4. Enter the command ‘python3 otsu.py’.
5. The output images will be generated and added to the img folder.

**Source code:**

# importing all the required libraries

import math

import numpy as np

from matplotlib import pyplot as plt

from matplotlib import image as mpimg

from PIL import Image

# declaring a dictionary that will store all the threshold values along with the calculated variances.

threshold\_values = {}

h = [1]

# function to convert rgb to gray values

def rgb2gray(rgb):

return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

# computing the histogram using the intensity values of the pixels

def Histo(img):

row, col = img.shape

y = np.zeros(256)

for i in range(0,row):

for j in range(0,col):

y[int(round(img[i,j]))] += 1

x = np.arange(0,256)

plt.bar(x, y, color='b', width=5, align='center', alpha=0.25)

plt.show()

return y

# Function for assigning the intensity values to pixels by comparing with the thresholds and thereby computing the output image

def get\_output\_img(img, threshold):

row, col = img.shape

y = np.zeros((row, col))

for i in range(0,row):

for j in range(0,col):

if img[i,j] >= threshold[2]:

y[i,j] = 200

elif img[i,j] >= threshold[1]:

y[i,j] = 150

elif img[i,j] >= threshold[0]:

y[i,j] = 100

else:

y[i,j] = 0

return y

# function for counting the number of pixels of foreground

def countPixel(h):

count = 0

for i in range(0, len(h)):

if h[i]>0:

count += h[i]

return count

# function for calculating the weight, i.e, number of pixels belonging to foreground

def weight(s, e):

w = 0

for i in range(s, e):

w += h[i]

return w

# function for calculating mean of the intensities of the pixels

def mean(s, e):

m = 0

w = weight(s, e)

for i in range(s, e):

m += h[i] \* i

return m/float(w)

# function for calculating the variance of intensities of the pixels

def variance(s, e):

v = 0

m = mean(s, e)

w = weight(s, e)

for i in range(s, e):

v += ((i - m) \*\*2) \* h[i]

v /= w

return v

# function for generating the three thresholds using 3 for loops which calculate and stores variance for each combination of threshold values

def generate\_thresholds(h, filename):

count = countPixel(h)

for i in range(1, len(h)):

for j in range(i+1, len(h)):

for k in range(j+1, len(h)):

# variance and weight for pixels with intensities less than 'i'

v1 = variance(0, i)

w1 = weight(0, i) / float(count)

# variance and weight for pixels with intensities between 'i' and 'j'

v2 = variance(i, j)

w2 = weight(i, j) / float(count)

# variance and weight for pixels with intensities between 'j' and 'k'

v3 = variance(j, k)

w3 = weight(j, k) / float(count)

# variance and weight for pixels with intensities more than 'k'

v4 = variance(k, len(h))

w4 = weight(k, len(h)) / float(count)

# calculating the within class variance

V2w = w1 \* (v1) + w2 \* (v2) + w3 \* (v3) + w4 \* (v4)

# writing the threshold values and the calculated within class variances

fw = open(filename + ".txt", "a")

fw.write('T1='+ str(i) + "\n")

fw.write('T2='+ str(j) + "\n")

fw.write('T3='+ str(k) + "\n")

fw.write('within class variance='+ str(V2w) + "\n")

fw.write("\n")

# storing the threshold values and the corresponding in class variance in the dictionary

if not math.isnan(V2w):

threshold\_values[(i,j,k)] = V2w

# function that calculates optimal thresholds from the dictionary that contains all the thresholds and their corresponding variances

def get\_optimal\_thresholds():

min\_V2w = min(threshold\_values.values())

optimal\_threshold = [k for k, v in threshold\_values.items() if v == min\_V2w]

print('optimal threshold', optimal\_threshold[0])

return list(optimal\_threshold[0])

# work for the tiger1 image

image = mpimg.imread('img/tiger1.bmp') # reading the image

gray = rgb2gray(image) # conerting to gray level values

img = np.asarray(gray)

h = Histo(img) # plotting the histogram

generate\_thresholds(h, 'tiger1') # generating thresholds

optimal\_thresholds = get\_optimal\_thresholds() # getting optimal thresholds

res = get\_output\_img(img, optimal\_thresholds) # getting output image

plt.imshow(res) # displaying the output image

plt.savefig("img/tiger1\_out.jpg") # saving the output image

# work for the data13 image

image = mpimg.imread('img/data13.bmp') # reading the image

gray = rgb2gray(image) # conerting to gray level values

img = np.asarray(gray)

h = Histo(img) # plotting the histogram

generate\_thresholds(h, 'data13') # generating thresholds

optimal\_thresholds = get\_optimal\_thresholds() # getting optimal thresholds

res = get\_output\_img(img, optimal\_thresholds) # getting output image

plt.imshow(res) # displaying the output image

plt.savefig("img/data13\_out.jpg") # saving the output image

# work for the fruits2b image

image = mpimg.imread('img/fruits2b.bmp') # reading the image

gray = rgb2gray(image) # conerting to gray level values

img = np.asarray(gray)

h = Histo(img) # plotting the histogram

generate\_thresholds(h, 'fruits2b') # generating thresholds

optimal\_thresholds = get\_optimal\_thresholds() # getting optimal thresholds

res = get\_output\_img(img, optimal\_thresholds) # getting output image

plt.imshow(res) # displaying the output image

plt.savefig("img/fruits2b\_out.jpg") # saving the output image

**Generated Histograms:**

**Tiger1:**

Chart, histogram

Description automatically generated

**Data13:**

Chart

Description automatically generated

**Fruits2b:**

Chart, histogram

Description automatically generated

**Calculated thresholds:**

Tiger1: optimal threshold (45, 105, 175)

data13: optimal threshold (77, 155, 222)

fruits2b: optimal threshold (88, 140, 172)

**Output Images:**

**Tiger1:**

A picture containing calendar

Description automatically generated

optimal threshold (45, 105, 175)

**Data13:**

A picture containing chart

Description automatically generated

optimal threshold (77, 155, 222)

**Fruits2b:**

A picture containing chart

Description automatically generated

optimal threshold (88, 140, 172)

**Additional Information:**

The tiger1.txt, fruits2b.txt and data13.txt files contain the list of all the threshold values along with the corresponding within class variances.

The otsu.txt and otsu.py files contain the source code.

The tiger1\_out.bmp, data13\_out.bmp and fruits2b\_out.bmp are the output image files.