**Homework:**

Implement the OTSU algorithm and compare it with the build in OTSU binarization algorithm

**Solution:**

The OTSU algorithm using the within class variance was implemented using the code below:

**Code:**

# %%

from PIL import Image

import cv2

from matplotlib import pyplot as plt

import numpy as np

import math

from matplotlib.gridspec import GridSpec

filepath = 'image1.jpg'

img = cv2.imread(filepath)

h,w,c= img.shape

#color to black and white

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

img = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

#showing grayscale image

print(type(img))

print(img.shape)

plt.title("Original grayscale image")

plt.imshow(img, cmap= 'gray')

plt.savefig('1.jpg')

plt.show()

# %%

#showing histogram

hist = cv2.calcHist([img], [0], None, [256], [0,256])

plt.title("Histogram")

plt.hist(img.ravel(),256,[0,256])

plt.savefig('2.jpg')

plt.show()

print(hist.size)

# %%

#OTSU IMPLEMENTATION

img\_data = img.flatten()

x = np.bincount(img\_data)           #index is gray level, value is frequency

T\_candidates = []

sigmasq\_w\_vals = []

offset\_begin = 10

offset\_end = 256

for T in range(offset\_begin,offset\_end):

    foreground = []

    background = []

    for i in range(0,T):

        background.append(x[i])         #loading the bg frequency

    for i in range(T,256):

        foreground.append(x[i])         #loading the fg frequency

    #OMEGA CALCULATION

    omega\_b = np.sum(background)/(h\*w)

    omega\_f = np.sum(foreground)/(h\*w)

    #MU CALCULATION

    sum = 0

    for i in range(0,T):

        sum = sum + i\*x[i]              #gray level x frequency

    mu\_b = sum/np.sum(background)

    sum = 0

    for i in range(T,256):

        sum = sum + i\*x[i]

    mu\_f = sum/np.sum(foreground)

    #SIGMA SQUARE CALCULATION

    sum =0

    for i in range(0,T):

        var1 = math.pow((i-mu\_b),2)

        var2 = x[i]

        sum = sum + (var1\*var2)

    sigmasq\_b = sum/np.sum(background)

    sum = 0

    for i in range(T,256):

        var1 = math.pow((i-mu\_f),2)

        var2 = x[i]

        sum = sum + (var1\*var2)

    sigmasq\_f = sum/np.sum(foreground)

    #WITHIN CLASS VARIANCE CALCULATION

    sigmasq\_w = (omega\_b\*sigmasq\_b) + (omega\_f\*sigmasq\_f)

    #WITHIN CLASS VARIANCES APPENDING

    sigmasq\_w\_vals.append(sigmasq\_w)

#FINDING THE FINAL THRESHOLD, IT IS THE INDEX OF MINIMUM WITHIN CLASS VARIANCE

min\_sigmasq\_w = np.min(sigmasq\_w\_vals)

min\_index = sigmasq\_w\_vals.index(min\_sigmasq\_w)

#T\_final = min\_index + offset\_begin # offset added cause loop started from offset, not 0

T\_final = min\_index # not adding offset

# %%

#BINARIZATION WITH THE FINAL THRESHOLD VALUE

for i in range (0, len(img\_data)):

    if img\_data[i]<T\_final:

        img\_data[i] = 0

    else:

        img\_data[i] = 255

#RESHAPING FROM 1D TO 2D

bin\_image\_manual = np.reshape(img\_data,(h,w))

#BINARIZATION WITH BUILT IN METHOD

T\_builtin, bin\_image\_builtin = cv2.threshold(img, 0,255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

print(T\_final)

print(T\_builtin)

# %%

#PLOTTING

fig = plt.figure(figsize=(12,12))

gs = GridSpec(1,2)

fig.add\_subplot(gs[0,0])

plt.title("Manullay implemented OTSU")

plt.imshow(bin\_image\_manual, cmap='gray')

fig.add\_subplot(gs[0,1])

plt.title("Built-in OTSU")

plt.imshow(bin\_image\_builtin, cmap='gray')

plt.savefig('3.jpg')

plt.show()

**Output:**





