import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

from PIL import Image

from pylab import \*

# Load a color image in grayscale

img = cv.imread('lena512.bmp',0)

# get dimensions of image

dimensions = img.shape

# height, width, number of channels in image

height = img.shape[0]

width = img.shape[1]

print('Image Dimension : ', dimensions)

print('Image Height : ', height)

print('Image Width : ', width)

#finding the invert/ negetive of teh image

cv.imshow("Pic", img)

img\_not = cv.bitwise\_not(img)

cv.imshow("Invert1", img\_not)

cv.waitKey(0)

cv.destroyAllWindows()

#histogram normalization:

equ = cv.equalizeHist(img)

# stacking images side-by-side

res\_for\_normalization = np.hstack((img, equ))

# show image input vs output

cv.imshow('res\_for\_normalization.png', res\_for\_normalization)

cv.waitKey(0)

cv.destroyAllWindows()

#equalization

equ = cv.equalizeHist(img)

res\_for\_equalization = np.hstack((img,equ))

#stacking images side-by-side

cv.imwrite('res\_for\_equalization.png',res\_for\_equalization)

cv.imshow('res\_for\_equalization.png', res\_for\_normalization)

cv.waitKey(0)

cv.destroyAllWindows()

#plotting histogram

hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()

cdf\_normalized = cdf \* float(hist.max()) / cdf.max()

plt.plot(cdf\_normalized, color = 'b')

plt.hist(img.flatten(),256,[0,256], color = 'r')

plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()

cdf\_m = np.ma.masked\_equal(cdf,0)

cdf\_m = (cdf\_m - cdf\_m.min())\*255/(cdf\_m.max()-cdf\_m.min())

cdf = np.ma.filled(cdf\_m,0).astype('uint8')

#part c

lst = []

for i in range(img.shape[0]):

for j in range(img.shape[1]):

lst.append(np.binary\_repr(img[i][j], width=8)) # width = no. of bits

# We have a list of strings where each string represents binary pixel value. To extract bit planes we need to iterate over the strings and store the characters corresponding to bit planes into lists.

# Multiply with 2^(n-1) and reshape to reconstruct the bit image.

eight\_bit\_img = (np.array([int(i[0]) for i in lst], dtype=np.uint8) \* 128).reshape(img.shape[0], img.shape[1])

seven\_bit\_img = (np.array([int(i[1]) for i in lst], dtype=np.uint8) \* 64).reshape(img.shape[0], img.shape[1])

six\_bit\_img = (np.array([int(i[2]) for i in lst], dtype=np.uint8) \* 32).reshape(img.shape[0], img.shape[1])

five\_bit\_img = (np.array([int(i[3]) for i in lst], dtype=np.uint8) \* 16).reshape(img.shape[0], img.shape[1])

four\_bit\_img = (np.array([int(i[4]) for i in lst], dtype=np.uint8) \* 8).reshape(img.shape[0], img.shape[1])

three\_bit\_img = (np.array([int(i[5]) for i in lst], dtype=np.uint8) \* 4).reshape(img.shape[0], img.shape[1])

two\_bit\_img = (np.array([int(i[6]) for i in lst], dtype=np.uint8) \* 2).reshape(img.shape[0], img.shape[1])

one\_bit\_img = (np.array([int(i[7]) for i in lst], dtype=np.uint8) \* 1).reshape(img.shape[0], img.shape[1])

# Concatenate these images for ease of display using cv2.hconcat()

finalr = cv.hconcat([eight\_bit\_img, seven\_bit\_img, six\_bit\_img, five\_bit\_img])

finalv = cv.hconcat([four\_bit\_img, three\_bit\_img, two\_bit\_img, one\_bit\_img])

# Vertically concatenate

final = cv.vconcat([finalr, finalv])

# Display the images

cv.imshow('a', final)

cv.waitKey(0)

# Combining 4 bit planes

new\_img = eight\_bit\_img+seven\_bit\_img+six\_bit\_img+five\_bit\_img

# Display the image

cv.imshow('a',new\_img)

cv.waitKey(0)