**import cv2  
import numpy as np  
img = cv2.imread('two.tif')  
cv2.waitKey(0)  
for gamma in range(0,255):  
 gamma = gamma/255.0  
 gamma\_correction = np.array([[[pixel/255\*\*(1/gamma) for pixel in channel] for channel in cv2.split(img)] for img in [img]])  
 gamma\_correction = np.clip(gamma\_correction, 0, 1)\*255.0  
 gamma\_correction = np.uint8(np.around(gamma\_correction))  
 cv2.imshow('Original Image',img)  
 cv2.imshow('Gamma Correction', gamma\_correction)  
 cv2.waitKey(0)  
cv2.destroyAllWindows()**

**....**

import cv2  
import numpy as np  
img = cv2.imread('two.tif')  
gamma = 2  
lookUpTable = np.empty((1,256), np.uint8)  
for i in range(256):  
lookUpTable[0,i] = np.clip(pow(i / 255.0, gamma) \* 255.0, 0, 255)  
res = cv2.LUT(img, lookUpTable)  
cv2.imshow('original', img)  
cv2.imshow('result', res)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
  
#converting image to grayscale  
gray = cv2.cvtColor(res, cv2.COLOR\_BGR2GRAY)  
  
#applying thresholding  
ret,thresh1 = cv2.threshold(gray,127,255,cv2.THRESH\_BINARY)  
ret,thresh2 = cv2.threshold(gray,127,255,cv2.THRESH\_BINARY\_INV)  
  
#displaying the input and output image  
cv2.imshow('Input',img)  
cv2.imshow('Black and White',thresh1+thresh2)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**-........................**

import cv2  
import numpy as np  
  
  
def gamma\_correction(image,gamma=1.0):  
 invGamma = 1.0 / gamma  
 table = np.array([ ((i / 255.0) \*\* invGamma) \* 255  
 for i in np.arange(0,256) ]).astype("uint8")  
 return cv2.LUT(image,table)  
  
  
def contrast\_stretching(image):  
 min\_pixel = np.min(image)  
 max\_pixel = np.max(image)  
 contrast\_image = (image - min\_pixel) / (max\_pixel - min\_pixel) \* 255  
 return contrast\_image.astype(np.uint8)  
  
  
def separate\_into\_regions(image,threshold=128):  
 black\_region = np.zeros(image.shape,np.uint8)  
 white\_region = np.zeros(image.shape,np.uint8)  
 for i in range(image.shape[ 0 ]):  
 for j in range(image.shape[ 1 ]):  
 if image[ i ][ j ] < threshold:  
 black\_region[ i ][ j ] = 255  
 else:  
 white\_region[ i ][ j ] = 255  
 return black\_region,white\_region  
  
  
image = cv2.imread('two.tif')  
gamma\_corrected\_image = gamma\_correction(image,2.2)  
contrast\_stretched\_image = contrast\_stretching(gamma\_corrected\_image)  
black\_region,white\_region = separate\_into\_regions(contrast\_stretched\_image)  
  
cv2.imshow('Image',image)  
cv2.imshow('Gamma Corrected Image',gamma\_corrected\_image)  
cv2.imshow('Contrast Stretched Image',contrast\_stretched\_image)  
cv2.imshow('Black Region',black\_region)  
cv2.imshow('White Region',white\_region)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**................................................**

import numpy as np  
import cv2  
  
img = cv2.imread('two.tif')  
  
gamma = 0.5  
lookUpTable = np.empty((1,256), np.uint8)  
for i in range(256):  
 lookUpTable[0,i] = np.clip(pow(i / 255, gamma) \* 255.0, 0, 255)  
res = cv2.LUT(img, lookUpTable)  
  
cv2.imshow('Enhanced Image', res)  
cv2.imshow('Original Image', img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**.....................................**

import cv2  
import numpy as np  
  
img = cv2.imread('two.tif')  
  
#convert the input image to LAB color space  
lab = cv2.cvtColor(img, cv2.COLOR\_BGR2LAB)  
  
#split the image into L, A and B channels  
l, a, b = cv2.split(lab)  
  
#apply CLAHE to L channel  
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))  
cl = clahe.apply(l)  
  
#merge the CLAHE enhanced L channel with the original A and B channels  
limg = cv2.merge((cl,a,b))  
  
#convert the LAB image back to RGB color space  
final = cv2.cvtColor(limg, cv2.COLOR\_LAB2BGR)  
  
#display the output image  
cv2.imshow('Enhanced Image', final)  
cv2.imshow('Original Image', img)  
cv2.waitKey(0)

**...................................................................**

import cv2  
import numpy as np  
  
  
def gamma\_correction(image,gamma=1.0):  
 invGamma = 1.0 / gamma  
 table = np.array([ ((i / 255.0) \*\* invGamma) \* 255  
 for i in np.arange(0,256) ]).astype("uint8")  
 return cv2.LUT(image,table)  
  
  
def contrast\_stretching(image):  
 min\_pixel = np.min(image)  
 max\_pixel = np.max(image)  
 contrast\_image = (image - min\_pixel) / (max\_pixel - min\_pixel) \* 255  
 return contrast\_image.astype(np.uint8)  
  
  
def separate\_into\_regions(image,threshold=128):  
 black\_region = np.zeros(image.shape,np.uint8)  
 white\_region = np.zeros(image.shape,np.uint8)  
 for i in range(image.shape[ 0 ]):  
 for j in range(image.shape[ 1 ]):  
 if image[ i ][ j ] < threshold:  
 black\_region[ i ][ j ] = 255  
 else:  
 white\_region[ i ][ j ] = 255  
 return black\_region,white\_region  
  
  
image = cv2.imread('lab07.png')  
gamma\_corrected\_image = gamma\_correction(image,2.2)  
contrast\_stretched\_image = contrast\_stretching(gamma\_corrected\_image)  
black\_region,white\_region = separate\_into\_regions(contrast\_stretched\_image)  
  
cv2.imshow('Image',image)  
cv2.imshow('Gamma Corrected Image',gamma\_corrected\_image)  
cv2.imshow('Contrast Stretched Image',contrast\_stretched\_image)  
cv2.imshow('Black Region',black\_region)  
cv2.imshow('White Region',white\_region)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**......................**

**Working**

import cv2  
import numpy as np  
  
# read input image  
img = cv2.imread('two.tif',0)  
  
# applying contrast stretching  
s = np.log(img + 1)  
s = s / s.max() \* 255  
s = s.astype(np.uint8)  
  
# applying CLAHE  
clahe = cv2.createCLAHE(clipLimit=2.0,tileGridSize=(8,8))  
cl1 = clahe.apply(img)  
  
# applying Otsu's thresholding  
ret,thresh1 = cv2.threshold(cl1,0,255,cv2.THRESH\_OTSU)  
  
# applying Canny Edge detection  
edges = cv2.Canny(thresh1,100,200)  
  
# finding contours  
contours,hierarchy = cv2.findContours(edges,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)  
  
# finding contour with maximum area and store it as best\_cnt  
max\_area = 0  
for cnt in contours:  
 area = cv2.contourArea(cnt)  
 if area > max\_area:  
 max\_area = area  
 best\_cnt = cnt  
  
# finding centroids of best\_cnt and draw a circle there  
M = cv2.moments(best\_cnt)  
cx,cy = int(M[ 'm10' ] / M[ 'm00' ]),int(M[ 'm01' ] / M[ 'm00' ])  
cv2.circle(img,(cx,cy),5,255,-1)  
  
# show input and output image  
cv2.imshow('Input Image',img)  
cv2.imshow('Contrast Stretched Image',s)  
cv2.imshow('CLAHE Image',cl1)  
cv2.imshow('Otsu\'s thresholding',thresh1)  
cv2.imshow('Canny Edge detection',edges)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**......................................................................................................**

**import cv2**

**import numpy as np**

**# Read input image**

**img = cv2.imread('fundus.png', 0)**

**# Apply Max Filter**

**max\_filtered\_img = cv2.morphologyEx(img, cv2.MORPH\_TOPHAT, np.ones((3,3), np.uint8))**

**# Apply Otsu thresholding**

**\_, thresholded\_img = cv2.threshold(max\_filtered\_img, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)**

**# Find contours**

**\_, contours, \_ = cv2.findContours(thresholded\_img, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)**

**# Find the contour with the largest area and store it in a variable**

**max\_area\_contour = max(contours, key = lambda x: cv2.contourArea(x))**

**# Draw the contour on the input image**

**cv2.drawContours(img, [max\_area\_contour], 0, (0, 255, 0), 2)**

**# Display the input and output images**

**cv2.imshow('Input', img)**

**cv2.imshow('Output', thresholded\_img)**

**# Wait until a key is pressed**

**cv2.waitKey()**

**..............................................................................**

# import the required packages  
import cv2  
import numpy as np  
  
# read the image and convert it to grayscale  
img = cv2.imread('two.tif')  
gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
# apply the max filter to the grayscale image  
max\_filtered = cv2.morphologyEx(gray, cv2.MORPH\_TOPHAT, np.ones((3,3), np.uint8))  
  
# threshold the max filtered image to obtain the binary image  
\_, thresh = cv2.threshold(max\_filtered, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
  
# invert the binary image to obtain the final image  
final\_image = 255 - thresh  
  
# display the final image  
cv2.imshow('Final Image', final\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**......................................................................................................**

**import os**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import cv2**

**import keras**

**from keras.models import Sequential**

**from keras.layers import Dense, Dropout, Flatten**

**from keras.layers import Conv2D, MaxPooling2D**

**from keras import backend as K**

**img\_rows, img\_cols = 28, 28**

**# the data, split between train and test sets**

**(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()**

**if K.image\_data\_format() == 'channels\_first':**

**x\_train = x\_train.reshape(x\_train.shape[0], 1, img\_rows, img\_cols)**

**x\_test = x\_test.reshape(x\_test.shape[0], 1, img\_rows, img\_cols)**

**input\_shape = (1, img\_rows, img\_cols)**

**else:**

**x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1)**

**x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1)**

**input\_shape = (img\_rows, img\_cols, 1)**

**x\_train = x\_train.astype('float32')**

**x\_test = x\_test.astype('float32')**

**x\_train /= 255**

**x\_test /= 255**

**print('x\_train shape:', x\_train.shape)**

**print(x\_train.shape[0], 'train samples')**

**print(x\_test.shape[0], 'test samples')**

**# convert class vectors to binary class matrices**

**y\_train = keras.utils.to\_categorical(y\_train, num\_classes)**

**y\_test = keras.utils.to\_categorical(y\_test, num\_classes)**

**model = Sequential()**

**model.add(Conv2D(32, kernel\_size=(3, 3),**

**activation='relu',**

**input\_shape=input\_shape))**

**model.add(Conv2D(64, (3, 3), activation='relu'))**

**model.add(MaxPooling2D(pool\_size=(2, 2)))**

**model.add(Dropout(0.25))**

**model.add(Flatten())**

**model.add(Dense(128, activation='relu'))**

**model.add(Dropout(0.5))**

**model.add(Dense(num\_classes, activation='softmax'))**

**model.compile(loss=keras.losses.categorical\_crossentropy,**

**optimizer=keras.optimizers.Adadelta(),**

**metrics=['accuracy'])**

**model.fit(x\_train, y\_train,**

**batch\_size=batch\_size,**

**epochs=epochs,**

**verbose=1,**

**validation\_data=(x\_test, y\_test))**

**score = model.evaluate(x\_test, y\_test, verbose=0)**

**print('Test loss:', score[0])**

**print('Test accuracy:', score[1])**

**#save the model**

**model.save("disc\_localization.h5")**

**-------------------------------------------------------------------------------**

import cv2  
import numpy as np  
  
#Read the image  
img = cv2.imread('two.tif')  
  
#Convert the image to grayscale  
gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  
  
#Apply thresholding on the grayscale image to obtain binary image  
ret, thresh = cv2.threshold(gray,127,255,0)  
  
#Find contours in the binary image  
contours, hierarchy = cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)  
  
#Find the contour with the maximum area  
cnt = max(contours, key = lambda x: cv2.contourArea(x))  
  
#Draw the contour on the original image  
cv2.drawContours(img, [cnt], 0, (0,255,0), 3)  
  
#Show the image  
cv2.imshow('Optic Disc Detection', img)  
  
#Wait for a keypress  
cv2.waitKey(0)  
  
#Destroy all the windows  
cv2.destroyAllWindows()

**-------------------------------------------------------------------------------------------**

import cv2  
import numpy as np  
  
img = cv2.imread('two.tif')  
  
# Convert to grayscale  
gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
# Apply histogram equalization  
eq = cv2.equalizeHist(gray)  
  
# Stack the images to get out output  
res = np.hstack((gray,eq)) #stacking images side-by-side  
  
# Save image  
# cv2.imshow(,res)  
cv2.imshow('res.png',res)  
#Wait for a keypress  
cv2.waitKey(0)  
  
#Destroy all the windows  
cv2.destroyAllWindows()

**-----------------------------------------------------------------**

# Detect and label the brightest round region in a given high contrast fundus image  
  
import matplotlib.pyplot as plt  
import numpy as np  
from skimage.feature import peak\_local\_max  
from skimage.morphology import watershed  
from scipy import ndimage  
  
# Generate an artificial high contrast image with several bright  
# round regions  
  
x, y = np.indices((80, 80))  
x1, y1, x2, y2 = 28, 28, 44, 52  
r1, r2 = 16, 20  
mask\_circle1 = (x - x1)\*\*2 + (y - y1)\*\*2 < r1\*\*2  
mask\_circle2 = (x - x2)\*\*2 + (y - y2)\*\*2 < r2\*\*2  
image = np.logical\_or(mask\_circle1, mask\_circle2)  
  
# Now we want to separate the two objects in image  
# Generate the markers as local maxima of the distance to the background  
distance = ndimage.distance\_transform\_edt(image)  
local\_maxi = peak\_local\_max(distance, indices=False, footprint=np.ones((3, 3)),  
 labels=image)  
markers = ndimage.label(local\_maxi)[0]  
labels = watershed(-distance, markers, mask=image)  
  
fig, axes = plt.subplots(1, 2, figsize=(8, 3), sharex=True, sharey=True)  
ax = axes.ravel()  
ax[0].imshow(image, cmap=plt.cm.gray, interpolation='nearest')  
ax[0].set\_title('Overlapping objects')  
ax[1].imshow(-distance, cmap=plt.cm.gray, interpolation='nearest')  
ax[1].set\_title('Distances')  
for a in ax:  
 a.set\_axis\_off()  
  
fig, ax = plt.subplots(figsize=(4, 3))  
ax.imshow(labels, cmap=plt.cm.spectral, interpolation='nearest')  
ax.set\_title('Segmented objects')  
ax.set\_axis\_off()  
  
plt.show()

**-----------------------------------------------------------**

**from PIL import Image**

**import numpy as np**

**import matplotlib.pyplot as plt**

**# read image**

**img = Image.open("Output.tif").convert("L")**

**# find max intensity**

**w, h = img.size**

**intensity = []**

**for i in range(w):**

**for j in range(h):**

**pixel = img.getpixel((i,j))**

**intensity.append(pixel)**

**max\_intensity = max(intensity)**

**# find coordinates of max intensity**

**coords = []**

**for i in range(w):**

**for j in range(h):**

**pixel = img.getpixel((i,j))**

**if pixel == max\_intensity:**

**coords.append((i,j))**

**# plot max intensity region**

**x, y = zip(\*coords)**

**plt.plot(x, y, "ro")**

**# save and show plot**

**plt.savefig("output.png")**

**plt.show()**

**----------------------------------------------------------------------------------------------------------------**

**WORKING**

import cv2  
import numpy as np  
  
# read image  
img = cv2.imread( "Output.tif" , 0 )  
  
# apply threshold  
\_ , thresh = cv2.threshold( img , 200 , 255 , cv2.THRESH\_BINARY )  
  
# find contours  
contours , \_ = cv2.findContours( thresh , cv2.RETR\_TREE , cv2.CHAIN\_APPROX\_SIMPLE )  
  
# sort contours  
sorted\_contours = sorted( contours , key=lambda x : cv2.contourArea( x ) , reverse=True )  
  
# extract the largest contour  
largest\_contour = sorted\_contours[ 0 ]  
  
# draw the largest contour on the original image  
cv2.drawContours( img , [ largest\_contour ] , 0 , (0 , 255 , 0) , 3 )  
  
# display the result  
cv2.imshow( "FUCK YOU" , img )  
cv2.imwrite("RETINAL OPTIC DISC.tif", img)  
cv2.waitKey( 0 )  
cv2.destroyAllWindows()

**----------------------------------------------------------------**