**Explaination of code:**

#Importing libraries

import keras

from keras.models import Sequential

from keras.layers import Conv2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense

from PIL import ImageFile

import numpy as np

import dicom

import os

import matplotlib.pyplot as plt

from glob import glob

from mpl\_toolkits.mplot3d.art3d import Poly3DCollection

import scipy.ndimage

from skimage import morphology

from skimage import measure

from skimage.transform import resize

from sklearn.cluster import KMeans

from keras.preprocessing import image

ImageFile.LOAD\_TRUNCATED\_IMAGES = True **#for images which are corrupted and can not load,it let the program keep running**

**#Libraries imported**

data\_path = "C:\\Users\\peace\\Desktop\\11\\HGJ\\HN-HGJ-001\\CT\\image\\" **#path to patient dicom the image folder**

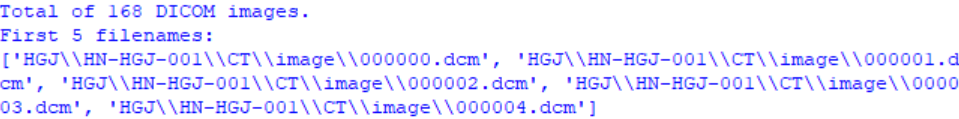
output\_path = working\_path = "output\\" #**folder where npy data will be saved**

g = glob(data\_path + '/\*.dcm') #**search the data having .dcm extension in directory**

**# Print out the first 5 file names to verify we're in the right folder.**

print ("Total of %d DICOM images.\nFirst 5 filenames:" % len(g))

print (g[:5])

Output: 

**# Loop over the image files and store everything into a list.**

def load\_scan(path):

slices = [dicom.read\_file(path + '/' + s) for s in os.listdir(path)]

slices.sort(key = lambda x: int(x.InstanceNumber))

try:

slice\_thickness = np.abs(slices[0].ImagePositionPatient[2] - slices[1].ImagePositionPatient[2])

except:

slice\_thickness = np.abs(slices[0].SliceLocation - slices[1].SliceLocation)

for s in slices:

s.SliceThickness = slice\_thickness

return slices

**#calculate LU , save data in npy flie and plot Hounsfield Units (HU) and frequency graph**

def get\_pixels\_hu(scans):

image = np.stack([s.pixel\_array for s in scans])

image = image.astype(np.int16)

# Set outside-of-scan pixels to 1

# The intercept is usually -1024, so air is approximately 0

image[image == -2000] = 0

# Convert to Hounsfield units (HU)

intercept = scans[0].RescaleIntercept

slope = scans[0].RescaleSlope

if slope != 1:

image = slope \* image.astype(np.float64)

image = image.astype(np.int16)

image += np.int16(intercept)

return np.array(image, dtype=np.int16)

id=0

patient = load\_scan(data\_path)

imgs = get\_pixels\_hu(patient)

np.save(output\_path + "fullimages\_%d.npy" % (id), imgs)

file\_used=output\_path+"fullimages\_%d.npy" % id

imgs\_to\_process = np.load(file\_used).astype(np.float64)

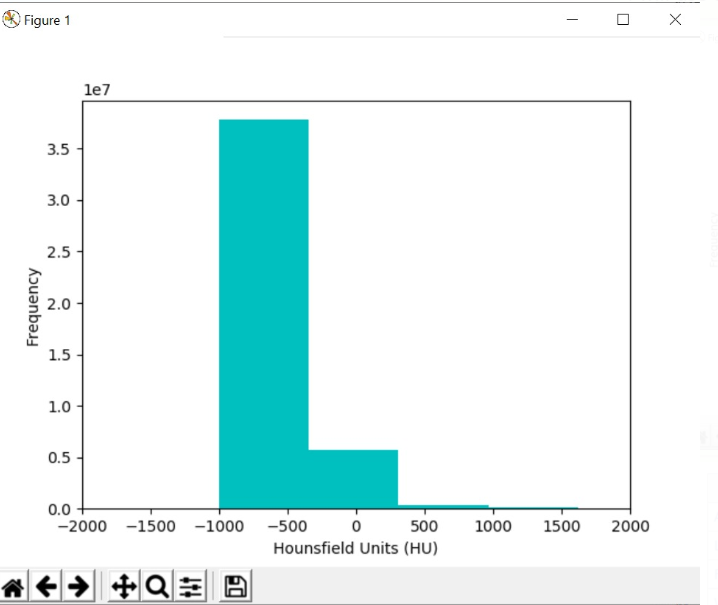
plt.hist(imgs\_to\_process.flatten(), bins=50, color='c')

plt.xlabel("Hounsfield Units (HU)")

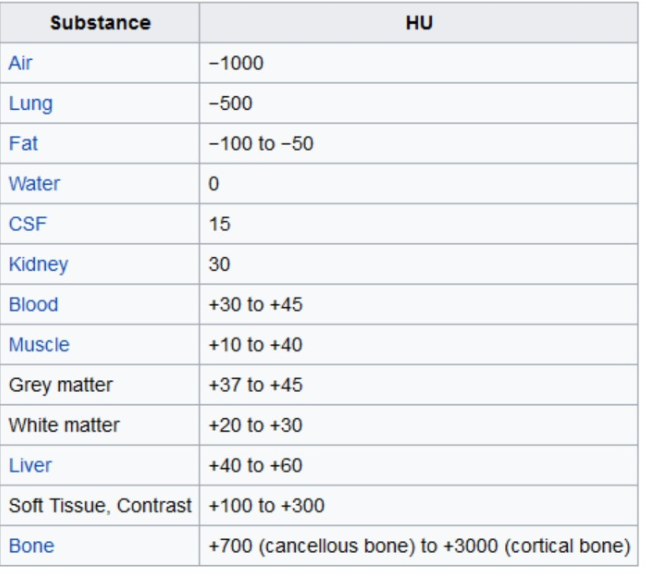
plt.xlim(-2000, 2000)

plt.ylabel("Frequency")

plt.show()

output: 

**#for refrence of HU unit**



**#Show every 3rd dicom image from counting of 10**

id = 0

imgs\_to\_process = np.load(output\_path+'fullimages\_{}.npy'.format(id))

def sample\_stack(stack, rows=6, cols=6, start\_with=10, show\_every=3):

fig,ax = plt.subplots(rows,cols,figsize=[12,12])

for i in range(rows\*cols):

ind = start\_with + i\*show\_every

ax[int(i/rows),int(i % rows)].set\_title('slice %d' % ind)

ax[int(i/rows),int(i % rows)].imshow(stack[ind],cmap='gray')

ax[int(i/rows),int(i % rows)].axis('off')

plt.show()

sample\_stack(imgs\_to\_process)

print ("Slice Thickness: %f" % patient[0].SliceThickness) #**print Slice thinkness of image**

print ("Pixel Spacing (row, col): (%f, %f) " % (patient[0].PixelSpacing[0], patient[0].PixelSpacing[1])) **#print pixel spacing**

**#Reshampling the images Determine current pixel spacing**

id = 0

imgs\_to\_process = np.load(output\_path+'fullimages\_{}.npy'.format(id))

def resample(image, scan, new\_spacing=[1,1,1]):

spacing = map(float, ([scan[0].SliceThickness] + scan[0].PixelSpacing))

spacing = np.array(list(spacing))

resize\_factor = spacing / new\_spacing

new\_real\_shape = image.shape \* resize\_factor

new\_shape = np.round(new\_real\_shape)

real\_resize\_factor = new\_shape / image.shape

new\_spacing = spacing / real\_resize\_factor

image = scipy.ndimage.interpolation.zoom(image, real\_resize\_factor)

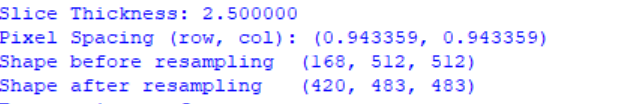
return image, new\_spacing

print ("Shape before resampling\t", imgs\_to\_process.shape) **#shape before resampling**

imgs\_after\_resamp, spacing = resample(imgs\_to\_process, patient, [1,1,1])

print ("Shape after resampling\t", imgs\_after\_resamp.shape) **#shape after resampling**

**output:**



**#making mask**

**#Standardize the pixel values**

def make\_mask(img, display=False):

row\_size= img.shape[0]

col\_size = img.shape[1]

mean = np.mean(img)

std = np.std(img)

img = img-mean

img = img/std

**# Find the average pixel value near the head and neck image to renormalize washed out images**

middle = img[int(col\_size/5):int(col\_size/5\*4),int(row\_size/5):int(row\_size/5\*4)]

mean = np.mean(middle)

max = np.max(img)

min = np.min(img)

**# To improve threshold finding, I'm moving the underflow and overflow on the pixel spectrum**

img[img==max]=mean

img[img==min]=mean

# **Using Kmeans to separate foreground (soft tissue / bone) and background (neck and head/air)**

kmeans = KMeans(n\_clusters=2).fit(np.reshape(middle,[np.prod(middle.shape),1]))

centers = sorted(kmeans.cluster\_centers\_.flatten())

threshold = np.mean(centers)

thresh\_img = np.where(img<threshold,1.0,0.0) # threshold the image

**# First erode away the finer elements, then dilate to include some of the pixels surrounding the head and neck.**

**# We don't want to accidentally clip the head and neck.**

eroded = morphology.erosion(thresh\_img,np.ones([3,3]))

dilation = morphology.dilation(eroded,np.ones([8,8]))

labels = measure.label(dilation**) # Different labels are displayed in different colors**

label\_vals = np.unique(labels)

regions = measure.regionprops(labels)

good\_labels = []

for prop in regions:

B = prop.bbox

if B[2]-B[0]<row\_size/10\*9 and B[3]-B[1]<col\_size/10\*9 and B[0]>row\_size/5 and B[2]<col\_size/5\*4:

good\_labels.append(prop.label)

mask = np.ndarray([row\_size,col\_size],dtype=np.int8)

mask[:] = 0

**# After just the head and neck are left, we do another large dilation in order to fill in and out the head and neck mask**

for N in good\_labels:

mask = mask + np.where(labels==N,1,0)

mask = morphology.dilation(mask,np.ones([10,10])) # one last dilation

if (display):

fig, ax = plt.subplots(3, 2, figsize=[12, 12])

ax[0, 0].set\_title("Original")

ax[0, 0].imshow(img, cmap='gray')

ax[0, 0].axis('off')

ax[0, 1].set\_title("Threshold")

ax[0, 1].imshow(thresh\_img, cmap='gray')

ax[0, 1].axis('off')

ax[1, 0].set\_title("After Erosion and Dilation")

ax[1, 0].imshow(dilation, cmap='gray')

ax[1, 0].axis('off')

ax[1, 1].set\_title("Color Labels")

ax[1, 1].imshow(labels)

ax[1, 1].axis('off')

ax[2, 0].set\_title("Final Mask")

ax[2, 0].imshow(mask, cmap='gray')

ax[2, 0].axis('off')

ax[2, 1].set\_title("Apply Mask on Original")

ax[2, 1].imshow(mask\*img, cmap='gray')

ax[2, 1].axis('off')

plt.show()

return mask\*img

img = imgs\_after\_resamp[260]

make\_mask(img, display=True)

masked\_image = []

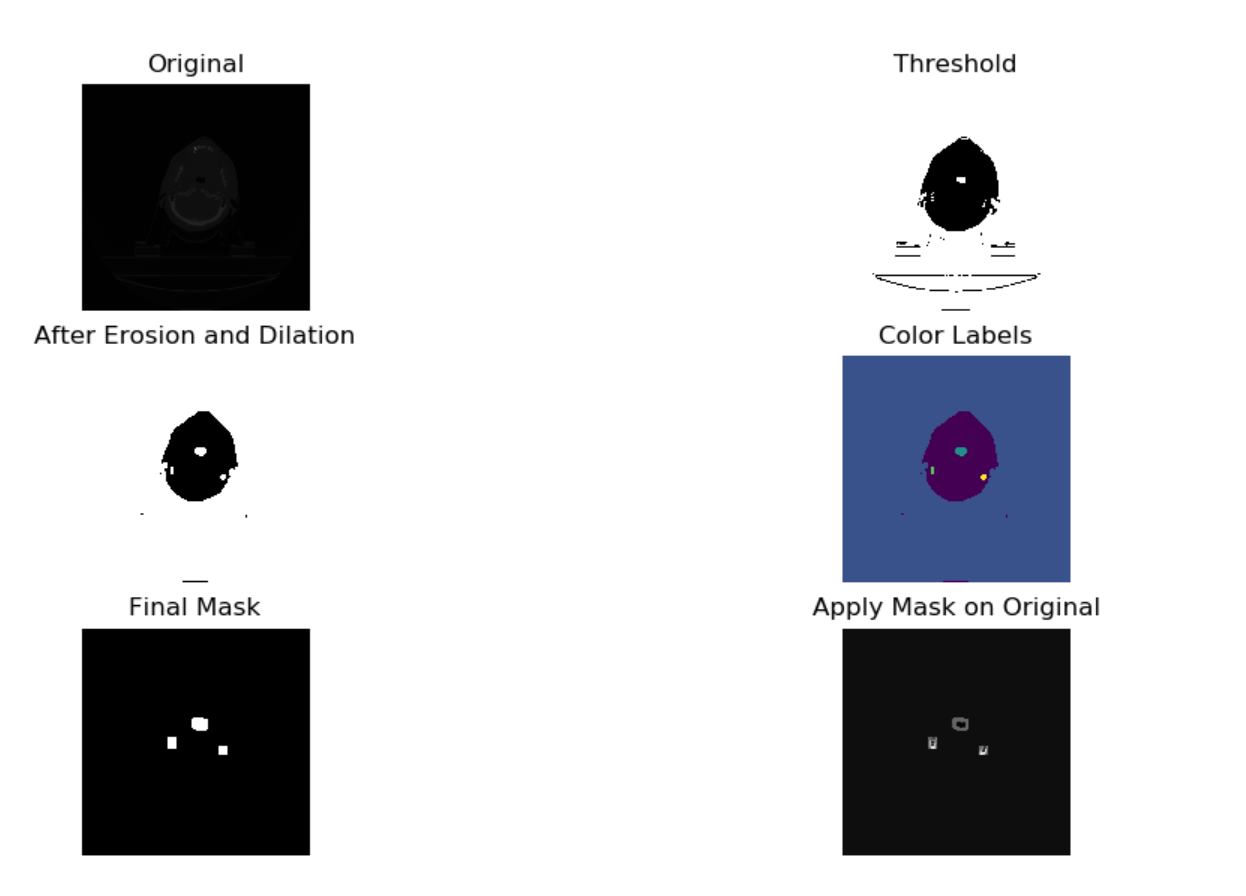
for img in imgs\_after\_resamp:

masked\_image.append(make\_mask(img))

sample\_stack(masked\_lung, show\_every=10)

np.save(output\_path + "maskedimages\_%d.npy" % (id), imgs) **#saved maked images**

**Output:**



**#Convert all DICOM images to jpg and using Labels create two folders In which one contain cancerous images and another will contain non cancerous,this has to done manually**

**#convet dicom to jpg or png**

**'''# make it True if you want in PNG format**

PNG = False

**# Specify the .dcm folder path**

folder\_path = "stage\_1\_test\_images"

**# Specify the output jpg/png folder path**

jpg\_folder\_path = "JPG\_test"

images\_path = os.listdir(folder\_path)

for n, image in enumerate(images\_path):

ds = pydi.dcmread(os.path.join(folder\_path, image))

pixel\_array\_numpy = ds.pixel\_array

if PNG == False:

image = image.replace('.dcm', '.jpg')

else:

image = image.replace('.dcm', '.png')

cv2.imwrite(os.path.join(jpg\_folder\_path, image), pixel\_array\_numpy)

if n % 50 == 0:

print('{} image converted'.format(n))

'''

#**Initialising the CNN**

classifier = Sequential()

**# Step 1 - Convolution**

classifier.add(Conv2D(32, (3, 3), input\_shape = (64, 64, 3), activation = 'relu'))

**# Step 2 - Pooling**

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

**# Adding a second convolutional layer**

classifier.add(Conv2D(32, (3, 3), activation = 'relu'))

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

**# Step 3 - Flattening**

classifier.add(Flatten())

**# Step 4 - Full connection**

classifier.add(Dense(units = 128, activation = 'relu'))

classifier.add(Dense(units = 1, activation = 'sigmoid'))

**# Compiling the CNN**

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

**# Part 2 - Fitting the CNN to the images**

from keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255,

shear\_range = 0.2,

zoom\_range = 0.2,

horizontal\_flip = True)

test\_datagen = ImageDataGenerator(rescale = 1./255)

training\_set = train\_datagen.flow\_from\_directory('C:\\Users\\peace\\Desktop\\11\\both\\',

target\_size = (64, 64),

batch\_size = 32,

class\_mode = 'binary')

test\_set = test\_datagen.flow\_from\_directory('C:\\Users\\peace\\Desktop\\11\\both\\',

target\_size = (64, 64),

batch\_size = 32,

class\_mode = 'binary')

classifier.fit\_generator(training\_set,

steps\_per\_epoch =300,

epochs = 5,

validation\_data = test\_set,

validation\_steps = 60) #there are 300 steps 5 times for training

**# Part 3 - Making new predictions //for checking that model is working or not**

test\_image = image.load\_img('C:\\Users\\peace\\Desktop\\11\\1.jpg', target\_size = (64, 64))

test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0)

result = classifier.predict(test\_image)

training\_set.class\_indices

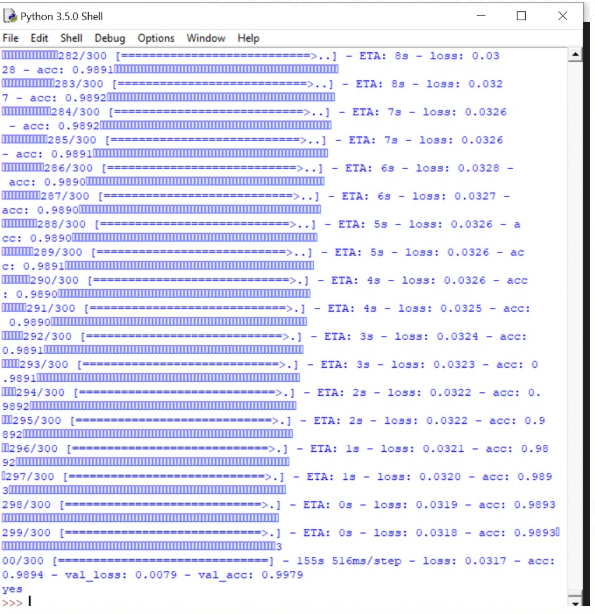
if result[0][0] == 1:

print("yes")

else:

print("no")

**Output:** output is long so only sharing the last part of the output.



**Problems:-**

**1:-Need large dataset which will be divided into test train and validation images for more accuracy.**