**Pre processing Code:**

import matplotlib.style

from scipy import signal

import scipy

import numpy as np

from numpy import asarray

import aifc

import matplotlib.pyplot as plt

from matplotlib import mlab

import matplotlib

import pandas as pd

import glob

import cv2

from PIL import Image

path\_data = ''

file\_path=''

# ReadAIFF function

def ReadAIFF(file):

s = aifc.open(file,'r')

nFrames = s.getnframes()

strSig = s.readframes(nFrames)

return np.fromstring(strSig, np.short).byteswap()

# Plot spectrogram function

def plot\_spectrogram(filename):

sound = ReadAIFF(filename)

Pxx, freqs, bins, im = plt.specgram(sound, NFFT=256, Fs=2000, noverlap=160, cmap=plt.cm.gist\_heat)

mask = ((357.56 >= freqs) & (39.06 <= freqs))

denominator=0

#40x40 matrix

Pxx = Pxx[[mask]]

squared=np.square(Pxx)

for r in np.nditer(squared):

denominator += r

PxxStar=Pxx/denominator

print("yes")

matplotlib.image.imsave('images/' + filename + '.png', PxxStar)

#plot\_spectrogram(path\_data + 'whales/train6.aiff')

for filepath in glob.iglob('whales/\*.aiff'):

plot\_spectrogram(filepath)

**Create numpy arays to feed Keras Model**

data = []

labels = []

# grab the image paths and randomly shuffle them

imagePaths = sorted(list(paths.list\_images(args["dataset"])))

random.seed(42)

random.shuffle(imagePaths)

# loop over the input images

for imagePath in imagePaths:

# load the image, pre-process it, and store it in the data list

image = cv2.imread(imagePath)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

image = cv2.resize(image, (40, 40))

image = img\_to\_array(image)

data.append(image)

# extract the class label from the image path and update the

# labels list

label = imagePath.split(os.path.sep)[-2]

label = 1 if label == "whales" else 0

labels.append(label)

data = np.array(data, dtype="float")

labels = np.array(labels)

# partition the data into training and testing splits using 75% of

# the data for training and the remaining 25% for testing

(trainX, testX, trainY, testY) = train\_test\_split(data,

labels, test\_size=0.25, random\_state=42)

# convert the labels from integers to vectors

trainY = to\_categorical(trainY, num\_classes=2)

testY = to\_categorical(testY, num\_classes=2)

**LeNetModel:**

class LeNet:

@staticmethod

def build(width, height, depth, classes):

# initialize the model

model = Sequential()

inputShape = (height, width, depth)

# if we are using "channels first", update the input shape

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

inputShape = (depth, height, width)

# first set

model.add(Conv2D(32, (5, 5), padding="same",

input\_shape=inputShape))

model.add(Activation("tanh"))

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

# second set

model.add(Conv2D(64, (5, 5), padding="same"))

model.add(Activation("tanh"))

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

model.add(Flatten())

model.add(Dense(3136))

model.add(Activation("tanh"))

model.add(Dense(1000))

model.add(Activation("tanh"))

# softmax classifier

model.add(Dense(classes))

model.add(Activation("softmax"))

# return the constructed network architecture

return model

**Compiling model**

model = LeNet.build(width=40, height=40, depth=1, classes=2)

opt = SGD(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

model.compile(loss="binary\_crossentropy", optimizer=opt,

metrics=["accuracy"])