**SHIP DETECTION**

**SOURCE CODE:**

import pandas as pd

import warnings

warnings.filterwarnings('ignore')

import seaborn as sns

from matplotlib import pyplot as plt

import json, sys, random

from PIL import Image, ImageDraw

from keras.models import Sequential

from keras.layers import Dense, Flatten, Activation

from keras.layers import Dropout

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.utils import np\_utils

from keras.optimizers import SGD

import keras.callbacks

f = open(r'shipsnet.json')

dataset = json.load(f)

f.close()

input\_data = np.array(dataset['data']).astype('uint8')

output\_data = np.array(dataset['labels']).astype('uint8')

input\_data.shape

n\_spectrum = 3 # color chanel (RGB)

weight = 80

height = 80

X = input\_data.reshape([-1, n\_spectrum, weight, height])

X[0].shape

pic = X[0]

rad\_spectrum = pic[0]

green\_spectrum = pic[1]

blue\_spectum = pic[2]

plt.figure(2, figsize = (5\*3, 5\*1))

plt.set\_cmap('jet')

plt.subplot(1, 3, 1)

plt.imshow(rad\_spectrum)

plt.subplot(1, 3, 2)

plt.imshow(green\_spectrum)

plt.subplot(1, 3, 3)

plt.imshow(blue\_spectum)

plt.show()

with open(r'shipsnet.json') as data\_file:

dataset = json.load(data\_file)

Shipsnet= pd.DataFrame(dataset)

print(Shipsnet.head())

print('')

x = np.array(dataset['data']).astype('uint8')

y = np.array(dataset['labels']).astype('uint8')

def describeData(a,b):

print('Total number of images: {}'.format(len(a)))

print('Number of NoShip Images: {}'.format(np.sum(b==0)))

print('Number of Ship Images: {}'.format(np.sum(b==1)))

print('Percentage of positive images: {:.2f}%'.format(100\*np.mean(b)))

print('Image shape (Width, Height, Channels): {}'.format(a[0].shape))

describeData(x,y)

from keras.utils import to\_categorical

xReshaped = x.reshape([-1, 3, 80, 80]).transpose([0,2,3,1])

yReshaped = to\_categorical(y, num\_classes=2)

print("Data Shape",x.shape)

print('Labels Shape',y.shape)

print('Reshaped Data Shape',xReshaped.shape)

print('Reshaped Labels Shape',yReshaped.shape)

def describeDataset(features,labels):

print("\n'X' shape: %s."%(features.shape,))

print("\n'y' shape: %s."%(labels.shape,))

print("\nUnique elements in y: %s"%(np.unique(y)))

describeDataset(xReshaped,yReshaped)

imgs0 = xReshaped[y==0]

imgs1 = xReshaped[y==1]

def plotOne(a,b):

"""

Plot one numpy array

"""

plt.subplot(1,2,1)

plt.title('Not A Ship')

plt.imshow(a[100])

plt.subplot(1,2,2)

plt.title('Ship')

plt.imshow(b[100])

plotOne(imgs0, imgs1)

def plotTwo(a,b):

"""

Plot a bunch of numpy arrays sorted by label

"""

for row in range(3):

plt.figure(figsize=(20, 10))

for col in range(3):

plt.subplot(1,8,col+1)

plt.title('Not A Ship')

plt.imshow(a[row+col])

plt.axis('off')

plt.subplot(1,8,col+4)

plt.title('Ship')

plt.imshow(b[row+col])

plt.axis('off')

plotTwo(imgs0, imgs1)

output\_data.shape

output\_data

sns.countplot(output\_data)

np.bincount(output\_data)

y = np\_utils.to\_categorical(output\_data, 2)

indexes = np.arange(2800)

np.random.shuffle(indexes)

X\_train = X[indexes].transpose([0,2,3,1])

y\_train = y[indexes]

X\_train = X\_train / 255

np.random.seed(42)

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same', input\_shape=(80, 80, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #40x40

model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #20x20

model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #10x10

model.add(Dropout(0.25))

model.add(Conv2D(32, (10, 10), padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #5x5

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same', input\_shape=(80, 80, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #40x40

model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #20x20

model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #10x10

model.add(Dropout(0.25))

model.add(Conv2D(32, (10, 10), padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) #5x5

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

sgd = SGD(lr=0.01, momentum=0.9, nesterov=True)

model.compile(

loss='categorical\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

model.fit(

X\_train,

y\_train,

batch\_size=32,

epochs=2,

validation\_split=0.2,

shuffle=True,

verbose=2)

image = Image.open(r'Dataset/00e90efc3.jpg')

pix = image.load()

n\_spectrum = 3

width = image.size[0]

height = image.size[1]

picture\_vector = []

for chanel in range(n\_spectrum):

for y in range(height):

for x in range(width):

picture\_vector.append(pix[x, y][chanel])

picture\_vector = np.array(picture\_vector).astype('uint8')

picture\_tensor = picture\_vector.reshape([n\_spectrum, height, width]).transpose(1, 2, 0)

plt.figure(1, figsize = (15, 30))

plt.subplot(3, 1, 1)

plt.imshow(picture\_tensor)

plt.show()

picture\_tensor = picture\_tensor.transpose(2,0,1)

def cutting(x, y):

area\_study = np.arange(3\*80\*80).reshape(3, 80, 80)

for i in range(80):

for j in range(80):

area\_study[0][i][j] = picture\_tensor[0][y+i][x+j]

area\_study[1][i][j] = picture\_tensor[1][y+i][x+j]

area\_study[2][i][j] = picture\_tensor[2][y+i][x+j]

area\_study = area\_study.reshape([-1, 3, 80, 80])

area\_study = area\_study.transpose([0,2,3,1])

area\_study = area\_study / 255

sys.stdout.write('\rX:{0} Y:{1} '.format(x, y))

return area\_study

def not\_near(x, y, s, coordinates):

result = True

for e in coordinates:

if x+s > e[0][0] and x-s < e[0][0] and y+s > e[0][1] and y-s < e[0][1]:

result = False

return result

def show\_ship(x, y, acc, thickness=5):

for i in range(80):

for ch in range(3):

for th in range(thickness):

picture\_tensor[ch][y+i][x-th] = -1

for i in range(80):

for ch in range(3):

for th in range(thickness):

picture\_tensor[ch][y+i][x+th+80] = -1

for i in range(80):

for ch in range(3):

for th in range(thickness):

picture\_tensor[ch][y-th][x+i] = -1

for i in range(80):

for ch in range(3):

for th in range(thickness):

picture\_tensor[ch][y+th+80][x+i] = -1

step = 10; coordinates = []

for y in range(int((height-(80-step))/step)):

for x in range(int((width-(80-step))/step) ):

area = cutting(x\*step, y\*step)

result = model.predict(area)

if result[0][1] > 0.90 and not\_near(x\*step,y\*step, 88, coordinates):

coordinates.append([[x\*step, y\*step], result])

print(result)

plt.imshow(area[0])

plt.show()

for e in coordinates:

show\_ship(e[0][0], e[0][1], e[1][0][1])

picture\_tensor = picture\_tensor.transpose(1,2,0)

picture\_tensor.shape

plt.figure(1, figsize = (15, 30))

plt.subplot(3,1,1)

plt.imshow(picture\_tensor)

plt.show()