In [1]:

```
import json, sys, random
import numpy as np
```

In [2]:

```
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
```

```
C:\Users\sanat\Anaconda3\envs\tensorflow\lib\site-packages\h5py\__init__.py:
36: FutureWarning: Conversion of the second argument of issubdtype from `flo
at` to `np.floating` is deprecated. In future, it will be treated as `np.flo
at64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [3]:

```
from PIL import Image, ImageDraw
```

In [4]:

```
from matplotlib import pyplot as plt
```

## Download and study the dataset

In [5]:

```
# download dataset from json object
f = open('D:/Projects/New folder/shipsnet.json')
dataset = json.load(f)
f.close()
```

In [6]:

```
input_data = np.array(dataset['data']).astype('uint8')
output_data = np.array(dataset['labels']).astype('uint8')
```

INPUT

The dataset contains 2800 images. One image is represented as a vector of length 19200 elements.

In [7]:

```
input_data.shape
```

Out[7]:

```
(3600, 19200)
```

In [8]:

```
n_spectrum = 3 # color chanel (RGB)
weight = 80
height = 80
X = input_data.reshape([-1, n_spectrum, weight, height])
X[0].shape
```

Out[8]:

```
(3, 80, 80)
```

In [9]:

```
# get one chanel
pic = X[0]

rad_spectrum = pic[0]
green_spectrum = pic[1]
blue_spectum = pic[2]
```

In [10]:

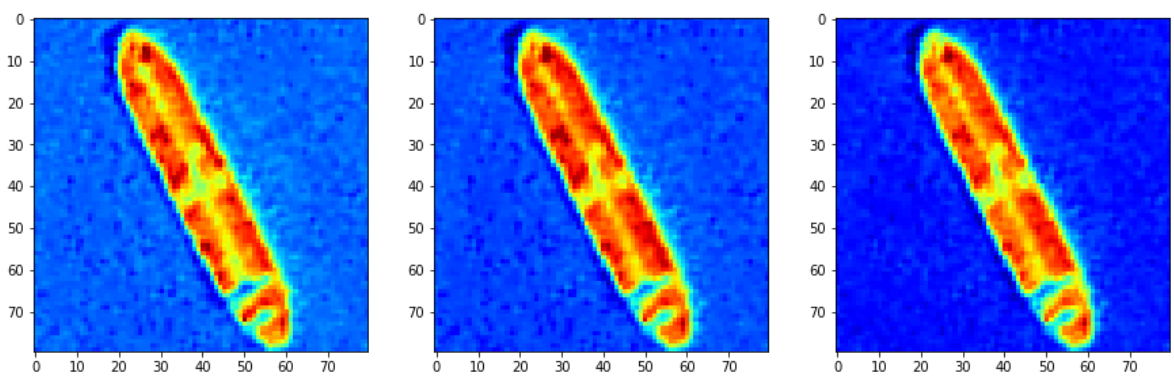
```
plt.figure(2, figsize = (5*3, 5*1))
plt.set_cmap('jet')

# show each channel
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()
```




OUTPUT

The output is a vector of lenght 2800 elements.

In [11]:

```
output_data.shape
```

Out[11]:

```
(3600,)
```

The vector contains int 0 and 1

In [12]:

```
output_data
```

Out[12]:

```
array([1, 1, 1, ..., 0, 0, 0], dtype=uint8)
```

Vector contains of 2100 zeros and 700 units. This means that in a dataset of 700 images tagged with "ship" and 2100 images marked as "not ship".

In [13]:

```
np.bincount(output_data)
```

Out[13]:

```
array([2700,  900], dtype=int64)
```

## Preparing data

In [14]:

```
# output encoding
y = np_utils.to_categorical(output_data, 2)
```

In [15]:

```
# shuffle all indexes
indexes = np.arange(2800)
np.random.shuffle(indexes)
```

In [16]:

```
X_train = X[indexes].transpose([0,2,3,1])
y_train = y[indexes]
```

In [17]:

```
# normalization
X_train = X_train / 255
```

## Traing network

In [18]:

```
np.random.seed(42)
```

In [19]:

```
# network design
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'))
```

In [20]:

```
# optimization setup
sgd = SGD(lr=0.01, momentum=0.9, nesterov=True)
model.compile(
    loss='categorical_crossentropy',
    optimizer=sgd,
    metrics=['accuracy'])

# training
model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=10,
    validation_split=0.2,
    shuffle=True,
    verbose=2)
```

```
Epoch 2/10
 - 30s - loss: 0.1728 - acc: 0.9308 - val_loss: 0.1196 - val_acc: 0.9518
Epoch 3/10
 - 31s - loss: 0.1307 - acc: 0.9500 - val_loss: 0.0942 - val_acc: 0.9589
Epoch 4/10
 - 29s - loss: 0.1163 - acc: 0.9554 - val_loss: 0.0728 - val_acc: 0.9643
Epoch 5/10
 - 30s - loss: 0.0950 - acc: 0.9670 - val_loss: 0.0893 - val_acc: 0.9571
Epoch 6/10
 - 32s - loss: 0.0886 - acc: 0.9643 - val_loss: 0.0620 - val_acc: 0.9732
Epoch 7/10
 - 29s - loss: 0.0871 - acc: 0.9688 - val_loss: 0.0485 - val_acc: 0.9857
Epoch 8/10
 - 28s - loss: 0.0681 - acc: 0.9772 - val_loss: 0.0599 - val_acc: 0.9714
Epoch 9/10
 - 28s - loss: 0.0793 - acc: 0.9728 - val_loss: 0.0939 - val_acc: 0.9589
Epoch 10/10
 - 30s - loss: 0.0726 - acc: 0.9768 - val_loss: 0.0573 - val_acc: 0.9750
```

Out[20]:

# Using network

## download image

In [21]:

```
image = Image.open('D:/Projects/New folder/sfbay_1.png')
pix = image.load()
```

In [22]:

```
n_spectrum = 3
width = image.size[0]
height = image.size[1]
```

In [23]:

```
# create vector
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])
```

In [24]:

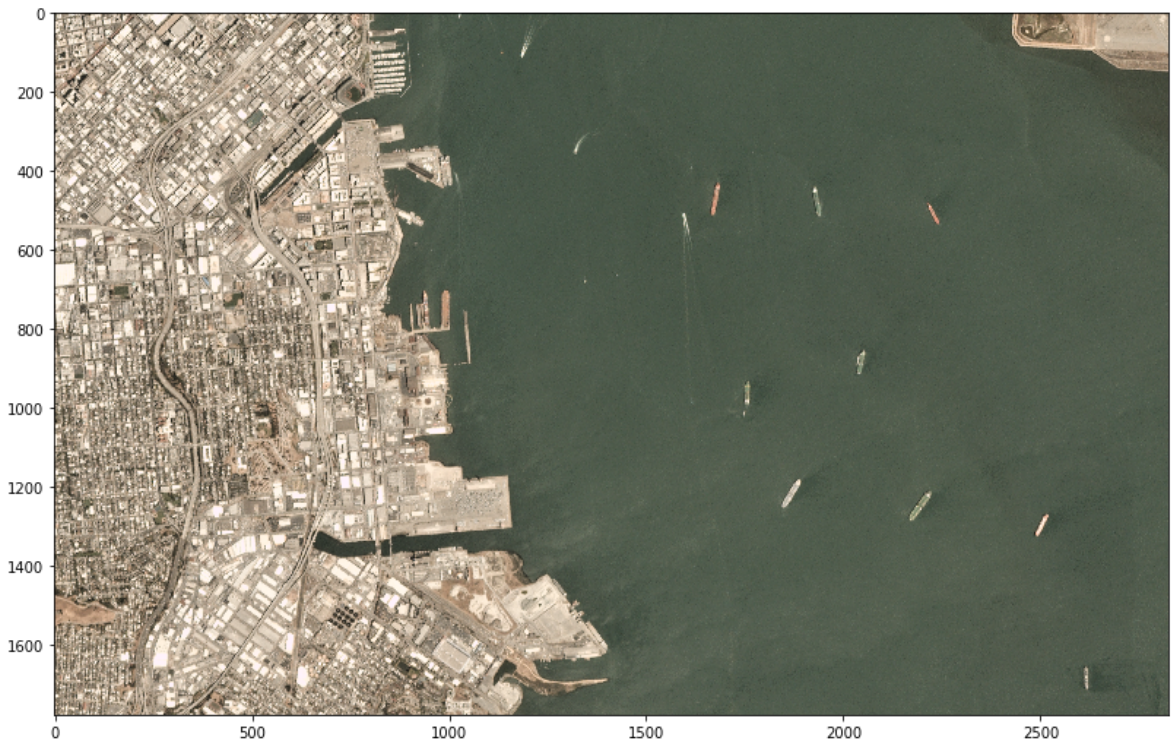
```
picture_vector = np.array(picture_vector).astype('uint8')
picture_tensor = picture_vector.reshape([n_spectrum, height, width]).transpose(1, 2, 0)
```

In [25]:

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

plt.subplot(3, 1, 1)
plt.imshow(picture_tensor)

plt.show()
```




In [26]:

```
picture_tensor = picture_tensor.transpose(2,0,1)
```

## Search on the image

In [27]:

```
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
```

In [28]:

```
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
```

In [29]:

```
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
```

In [30]:

```
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()
```


In [31]:

```
for e in coordinates:
    show_ship(e[0][0], e[0][1], e[1][0][1])
```

In [32]:

```
#picture_tensor = picture_tensor.transpose(2,0,1)
picture_tensor = picture_tensor.transpose(1,2,0)
picture_tensor.shape
```

Out[32]:

```
(1777, 2825, 3)
```

In [33]:

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

plt.subplot(3,1,1)
plt.imshow(picture_tensor)

plt.show()
```