VESSEL DETECTION IN SPACEBORNE PHOTOGRAPHY USING CNN

K V Siva Prasad Reddy[1], D Lidiya[2], G V R Sreevatsava[3], K Tharun[4]

[1]Adhoc Lecturer, Dept. of CSE, JNTUACEP, Pulivendula, AP, India, E-mail: svajntuacep@gmail.com
[2]Student, Dept. of CSE, JNTUACEP, Pulivendula, AP, India, E-mail: lidiyalilly5@gmail.com
[3]Student, Dept. of CSE, JNTUACEP, Pulivendula, AP, India, E-mail: vatsava.gvr@gmail.com
[4]Student, Dept. of CSE, JNTUACEP, Pulivendula, AP, India, E-mail: tharunkommaddi@gmail.com

***Abstract*— Information extracted from aerial photographs has found applications in different areas including agriculture, meteorology, oceanography, defence and climate modelling. In many cases information extraction is still performed by human experts, making the process slow costly, and error prone. New commercial imagery providers, such as Planet, are using constellations of small satellites to capture images of the entire Earth every day.**

Monitoring human activity and analysing the patterns of sea vessels or ships is a topic of major and increasing interest. From fishing, exploration, supply analysis, drilling, transportation, tourism all these are monitored by government using the satellite imagery. Maritime Domain Awareness (MDA) is the main governing body which looks over the security, safety and economy by defining areas and boundaries.

The aim of this project is to address the difficult task of detecting the location of large ships in satellite images. We would like to do this automated process by using Convolution Neural Networks. Automating this process can be applied to many issues including monitoring port activity levels and supply chain analysis.

We obtained 4000 images of various ports from state California. 1000 images contain ships in them and the remaining 3000 images doesn't have any ships in them.

Keywords— spaceborne photography, Convolution neural networks, dataset, machine learning algorithms, classification models.

1. INTRODUCTION

SPACE BORNE PHOTOGRAPHY is one type of photographs of earth operated by government. These images are taken by a constellation of satellites. There are many other private companies which sell images by licensing them to governments and businesses such as Apple Maps and Google Maps.

This flood of new imagery is outgrowing the ability for organizations to manually look at each image that gets captured, and there is a need for machine learning and computer vision algorithms to help automate the analysis process. So in order to alleviate the burden of analyzing thousands of images we are automating the process of detecting the ships or vessels in a given satellite image. This has many applications in the areas like marine traffic, fisheries management, marine or port traffic, illegal fishing, maritime piracy, border control, etc.

Satellite imagery used to build this dataset is made available through Planet's Open California dataset, which is openly licensed. As such, this dataset is also available under the same CC-BY-SA license. Users can sign up for a free Planet account to search, view, and download their imagery and gain access to their API.

2. MOTIVATION

I used to wonder how could the government regulate and keep order of in seas. This thought used to run all over the time in my mind, as the purview of seas is vast and maintaining order is very difficult. Later I came to know the power of satellites and the role of satellite images in maritime peace. At present the current system is not automated in detecting the vessels. This really made me curious and I thought of developing a model which automatically detects the ship in a given satellite image.

3. PROPOSED WORK

Normal image detection is a very complicated process. It takes very long time and also not so effective in predicting the images correctly. So we would like to propose a system which uses Convolution Neural Networks. It is so effective and is the best algorithm for detecting images out in the market. CNNs are simple to create and are very effective. All we need to do is create a model. This model takes the images as input, and gives the probability of the ship being part in the image. As it is a neural network it forwards the image data in forward learning and updates the weights of the each node in network while back propagation. The neural network automatically updates the weights with the increase in epochs of the neural network (epoch is nothing but one cycle of forward and backward propagation).

Steps associated with this proposal

- Download the data
- Clean the data
- Visualize the data
- Split the data into training and testing sets
- Create a benchmark model
- Compile and fit the data
- Implementation of model

4. METRICS

Accuracy is a common metric for binary classifiers;

Accuracy = (images correctly classified) / (total no. of images)

With addition to accuracy we also calculate precision and recall as metrics

Precision = True positives / (True Positives + False Positives)

Recall = True positives / (True Positives + False Negatives)

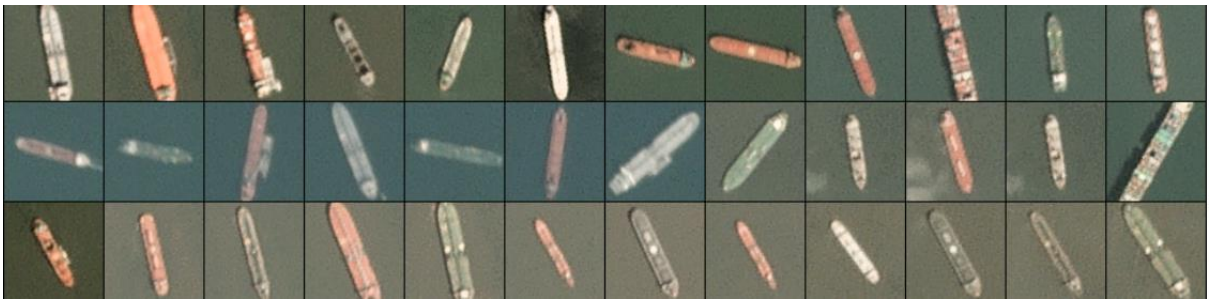
During development, a validation set was used to evaluate the model. I want to use a small set of training images as my validation images. For validation I want to use “categorical_crossentropy” as loss metric for CNN, optimizer as ‘SGD’ and ‘ADAM’.

5. DATASET

The dataset consists of image pieces extracted from Planet satellite imagery collected over the San Pedro Bay and San Francisco Bay which are the large ports in California. It contains 4000 80x80 RGB images which are labelled with either a "ship" or "no-ship" classification. In order to easy classify the images these were ORTHOCENTRIFIED to a 3 meter pixel size. This data of images is in the form of a JSON file loaded with **data**, **label**, **scene_ids**, and **location** lists.

The pixel value data for each 80x80 RGB image is stored as a list of 19200 integers within the **data** list. The first 6400 entries contain the red channel values, the next 6400 the green, and the final 6400 the blue. The image is stored in row-major order, so that the first 80 entries of the array are the red channel values of the first row of the image.

The dataset is distributed as a JSON formatted text file. The loaded object contains data, label, scene_ids, and location lists.The "ship" class includes 1000 images. Images in this class are near-centered on the body of a single ship. Ships of different sizes, orientations, and atmospheric collection conditions are included. Example images from this class are shown below.



The "no-ship" class includes 3000 images. A third of these are a random sampling of different landcover features - water, vegetion, bare earth, buildings, etc. - that do not include any portion of an ship. The next third are "partial ships" that contain only a portion of an ship, but not enough to meet the full definition of the "ship" class. The last third are images that have previously been mislabeled by machine learning models, typically caused by bright pixels or strong linear features. Example images from this class are shown below.

6. RESEARCH METHODOLOGY

Algorithms and Techniques

The classifier is a Convolutional Neural Network, which is the state-of-the-art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches. The algorithm outputs an assigned probability for each class; this can be used to reduce the number of false positives using a **threshold**. (The trade-off is that this increases the number of false negatives.) The following parameters can be tuned to optimize the classifier:

- Training parameters
  - Training length (number of epochs)
  - Batch size (how many images to look at once during a single training step)
  - Solver type (what algorithm to use for learning)
  - Learning rate (how fast to learn; this can be dynamic)
  - Weight decay (prevents the model being dominated by a few “neurons”)
  - Momentum (takes the previous learning step into account when calculating the next one)
- Neural network architecture
  - Number of layers
  - Layer types (convolutional, fully-connected, or pooling)
  - Layer parameters
- Pre-processing parameters

During training, both the training and the validation sets are loaded into the RAM. After that, random batches are selected to be loaded into the GPU memory for processing. The training is done using the Adam algorithm.

Theory behind Scenes:-

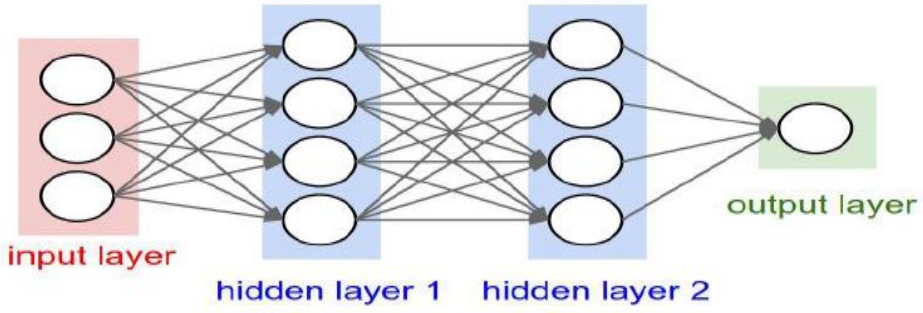
Artificial neural networks (ANNs): ANNs are computing systems vaguely inspired by the biological Neural Networks that constitute animal brains and humans. These systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

A typical brain contains something like 100 billion miniscule cells called neurons (no-one knows exactly how many there are and estimates go from about 50 billion to as many as 500 billion).

Each neuron is made up of a cell body (the central mass of the cell) with a number of connections coming off it: numerous dendrites (the cell's inputs—carrying information toward the cell body) and a single axon (the cell's output—carrying information away). Neurons are so tiny that you could pack about 100 of their cell bodies into a single millimetre. Inside a computer, the equivalent to a brain cell is a tiny switching device called a transistor. The latest, cutting-edge microprocessors (single-chip computers) contain over 2 billion transistors; even a basic microprocessor has about 50 million transistors, all packed onto an integrated circuit just 25mm square (smaller than a postage stamp)!

ANN is a set of connected neurons organized in layers:

- **Input layer**: brings the initial data into the system for further processing by subsequent layers of artificial neurons. Like dendrites in human neuron
- **Hidden layer**: a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function. Like nucleus in human neuron.
- **Output layer**: the last layer of neurons that produces given outputs for the program. Like axon in human neuron



Typical architecture of a neural network

Working Procedure

Step 1- Model initialization: -

A random initialization of the model is a common practice. The rationale behind is that from wherever we start, if we are perseverant enough and through an iterative learning process, we can reach the pseudo-ideal model.

Step 2- Forward propagate:-

The natural step to do after initializing the model at random, is to check its performance.
We start from the input we have, we pass them through the network layer and calculate the actual output of the model straightforwardly.

Step 3- Loss function:-

At this stage, in one hand, we have the actual output of the randomly initialized neural network.
On the other hand, we have the desired output we would like the network to learn. Then we define what we call: loss function (for intuition just think loss function like absolute difference or squared difference, but it changes with model to model). Basically, it is a performance metric on how well the NN manages to reach its goal of generating outputs as close as possible to the desired values.

Step 4- Differentiation:-

We can use any optimization technique that modifies the internal weights of neural networks in order to minimize the total loss function that we previously defined.

Step 5- Back-propagation:-

Then error in loss function is propagated from output layer to input layer by using differentiation we solved in step-4

Step 6- Weight update:-

Then corresponding weights in the network are changed in order with learning rate.
New weight = old weight — Derivative Rate * learning rate

Step 7- Iterate until convergence:-

Just iterate the procedure from step2 to step6 until convergence

In here the convergence depends on different factors:

- Depends on learning rate
- On optimization method
- Different meta-parameters

The following parameters can be tuned to optimize the model:

- Number of epochs
- Batch size
- Number of layers
- Different layers like convolutional, fully-connected or pooling etc,.

7. IMPLEMENTATION

7.1 Data Pre-processing

7.1.1 Loading the required libraries

In here we import all the libraries that are required for the execution of the program.

```
#Import libraries

# for data visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import patches
import seaborn as sns
import json,sys
# Model selection and metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve, confusion_matrix, roc_auc_scor
e
from sklearn.metrics import classification_report
import random as rn
# pre-processing the data
from keras.preprocessing.image import ImageDataGenerator

# tensorflow and keras libraries
import tensorflow as tf
from keras.models import Sequential
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras.callbacks import ModelCheckpoint
from keras.layers import Dense, Flatten, Activation, GlobalAveragePooling2D
from keras.layers import Dropout
```

```
# For manipulating images
from PIL import Image
```

All these libraries are used in one step or other. Each library has its own use which are explained briefly in comments in above code.
Each library is explained in previous chapter with their usability and the process involved in the usage of each library. Even the code is written associated with the libraries.

7.1.2 Loading the data

In here the data we contain is in the form of **JSON** format. This file contains the data in the form of a table. There are 4 columns with the names

- Data
- Labels
- Locations
- Scene_ids

```
# Loading dataset from shipnet.json file
file = open ('shipsnet.json')
dataset = json.load (file)
file.close()

# viewing the dataset
df = pd.DataFrame(dataset)
display (df.head(5))
display (df.tail(5))
```

	data	labels	locations	scene_ids
0	[82, 89, 91, 87, 89, 87, 86, 86, 86, 86, 84, 8...	1	[-118.2254694333423, 33.73803725920789]	20180708_180909_0f47
1	[76, 75, 67, 62, 68, 72, 73, 73, 68, 69, 69, 6...	1	[-122.33222866289329, 37.7491755586813]	20170705_180816_103e
2	[125, 127, 129, 130, 126, 125, 129, 133, 132, ...	1	[-118.14283073363218, 33.736016066914175]	20180712_211331_0f06
3	[102, 99, 113, 106, 96, 102, 105, 105, 103, 10...	1	[-122.34784341495181, 37.76648707436548]	20170609_180756_103a
4	[78, 76, 74, 78, 79, 79, 79, 82, 86, 85, 83, 8...	1	[-122.34852408322172, 37.75878462398653]	20170515_180653_1007

First five tuples in the data

	data	labels	locations	scene_ids
3995	[126, 122, 124, 138, 165, 186, 195, 199, 203, ...	0	[-122.08693255500694, 37.77781408256089]	20170815_180821_102d
3996	[130, 134, 139, 128, 117, 126, 141, 147, 142, ...	0	[-122.10549691828378, 37.76946626247702]	20170730_191230_0f21
3997	[171, 135, 118, 140, 145, 144, 154, 165, 139, ...	0	[-122.48298739296371, 37.684929808845375]	20161116_180804_0e14
3998	[85, 90, 94, 95, 94, 92, 93, 96, 93, 94, 94, 9...	0	[-122.29028216570079, 37.71632091139081]	20170211_181116_0e16
3999	[122, 122, 126, 126, 142, 153, 174, 190, 185, ...	0	[-122.49531387721586, 37.698557210117706]	20180206_184438_1043

Last five tuples in the data

7.1.3 Cleaning the data

The last two columns are not necessary so they are to be removed. And the image data is in the form of numbers which is to be converted into images. This are the things associated in cleaning of the data.

```
# The other two columns are not necessay as their values will not help.
# So I am dropping the two columns locations, scene_ids
df = df.drop(['locations','scene_ids'], axis = 1)

# Checking whether the columns were deleted or not
display(df.head())
```

	data	labels
0	[82, 89, 91, 87, 89, 87, 86, 86, 86, 86, 84, 8...	1
1	[76, 75, 67, 62, 68, 72, 73, 73, 68, 69, 69, 6...	1
2	[125, 127, 129, 130, 126, 125, 129, 133, 132, ...	1
3	[102, 99, 113, 106, 96, 102, 105, 105, 103, 10...	1
4	[78, 76, 74, 78, 79, 79, 79, 82, 86, 85, 83, 8...	1

Dataframe after dropping the unnecessary data

We are yet to convert the numerical data into image data. We are going to add all the images into a numpy array.

```
# we will convert the numerical data into image and append them into a list
images = []
for index in range( len( dataset['data'] )):
    pixel_vals = df['data'][index]
    arr = np.array(pixel_vals).astype('uint8')
    im = arr.reshape((3, 6400)).T.reshape( input_shape )
    images.append( im )
```

Normalizing the data is an essential step in image processing which helps in saving time.

```
# in order to acess the images converting them into a numpy array
images = np.array(images)
# normalizing the data by converting the values between (0,1)
images = images / 255.0
```

One-hot-coding is done in order to easy the process for the neural network. So the labels in here we have are either ship or no-ship which are yet to be changed into 1 or 0 using the one-hot-coding. This is done using the below code.

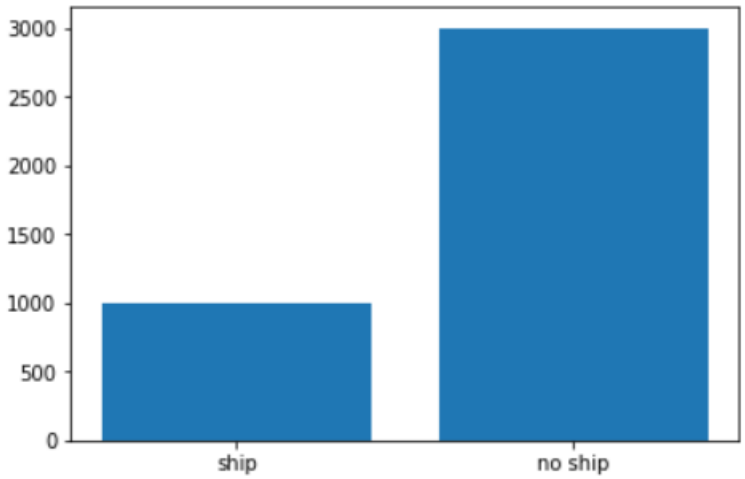
```
# creating the labels dataset and doing one-hot-coding for it
labels = dataset['labels']
labels = np_utils.to_categorical(labels)
```

7.1.4 Statistics of the data

In here we are to know the statistics of the dataset i.e., to know the number of images belong to a particular class. There by us can know whether to balance the dataset or not. This will help us analyze the dataset and so that we can take appropriate actions according to the results of the statistics.

```
# viewing the dataset in a graph
plt.bar(['ship','no ship'],[sum(df['labels'] == 1),sum(df['labels'] == 0)]);
```

We are going to plot the number of images belong to particular class “Ship” or “No-ship”.



Graph of statistics of data

7.1.5 Splitting data set

We have to split the data into training, testing and validation sets. Each set have their own working training is used to train the know data to the model testing and validation sets are unknown data used to know the performance of the model and evaluate it.

```
# splitting the data into training, testing and validation sets
x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size = 0.10, random_state = 42)

x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train, test_size = 0.10, random_state = 42)
```

```
# statistics about the data

print("There are total of %d images" % images.shape[0])

print('Total training images are ', x_train.shape[0])

print('Total testing images are ', x_test.shape[0])

print('Total validation images are ', x_validate.shape[0])
```

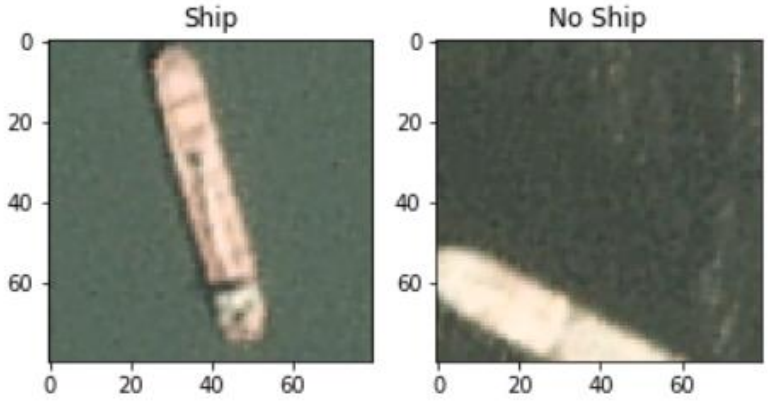
These are the results of the above code printing the statistics of the dataset.

There are total of 4000 images
Total training images are 3240
Total testing images are 400
Total validation images are 360

7.1.6 Sample images

```
# just plotting some images from the dataset
plt.figure(1)
plt.subplot(1,2,1)
plt.title('Ship')
plt.imshow(images[100])
plt.subplot(1,2,2)
plt.title('No Ship')
plt.imshow(images[2500]);
```

We would like to visualize the data using the matplotlib. This would help us validate ourselves whether the numerical data correctly converted into image data or not.



sample images

7.2 Training models

7.2.1 Creating images generators

As the dataset we contain have very less data this will make the model fitting very complicated. This is to be overcome using image generators which generated extra images using the present images by rotating and flipping and many others.

```
np.random.seed(48)
tf.set_random_seed(48)


# creating datagen
datagen = ImageDataGenerator(
      rotation_range=25,
# randomly rotate images in the range (degrees, 0 to 180)

      width_shift_range=0.1,
# randomly shift images horizontally (fraction of total width)

      height_shift_range=0.1,
# randomly shift images vertically (fraction of total height)

      horizontal_flip=True,
      vertical_flip = True)

datagen.fit(x_train)
```

This helps in creating a large dataset even though we don't have it from the data we started.

7.2.2 Building a benchmark model

In here we will create a benchmark model.
Bench mark model: Any CNN model that gives accuracy around 26% is my benchmark model. This model contains basic model of a CNN with minimum possible layers and so that this would fit for our benchmark.
My created model
Conv2D layer with 2 filters, kernel_size is equal to 3, and input shape is (80, 80, 3).
Max Pooling layer with pooling size is equal to 2.
Flatten layer and Dense layers with 2 units and activation of "SoftMax".
At compilation phase loss='categorical_crossentropy', optimizer='SGD', metrics= ['accuracy'] are used.
Check pointer is used with 'weights.best.bm_model.hdf5', as file path and save_best_only is tuned to True
Bench model is trained on fit_generator method in order to achieve the data augmentation with 32 as batch size and 110 steps per epoch and number epochs are 5.

```
# creating a benchmark model with one layer and least possible filters
bm_model = Sequential()
bm_model.add(Conv2D(2, (2,2), input_shape= input_shape))
bm_model.add(MaxPooling2D(pool_size=(2, 2))) #40x40

bm_model.add(Flatten())
bm_model.add(Dense(2))
```

7.2.2.2 Compile and fitting model

In order to validate a model first we need to compile the model

```
# compiling the model
bm_model.compile(
   loss='categorical_crossentropy',
   optimizer='sgd',
   metrics=['accuracy'])
```

Here we are using

- Loss function : categorical_crossentropy
- Optimization function : stochastic gradient descent
- Metrics : accuracy

These are the compilation metrics for the model
In order to test the model we have to fit the model

```
bench_results = bm_model.fit_generator(
   datagen.flow(x_train, y_train, batch_size=32),
   epochs=1,
   steps_per_epoch=len(x_train) / 32,
   validation_data=(x_validate,y_validate),
   callbacks=[checkpointer],
   verbose = 1)
```

- The output of the model after fitting the data is as above.
- The accuracy is about 25%
- The val_loss is about 4.07
- These are stored in weights.best.bm_model.hdf5 where the weights are stored.

7.2.3 Building Final model

As the model in the previous section is not so great and promising we are about to create a new model with more promising architecture.

```
f_model = Sequential()
f_model.add(Conv2D(32, (3, 3), padding='same', input_shape= input_shape, activation='relu'))
f_model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
f_model.add(MaxPooling2D(pool_size=(2, 2)))
f_model.add(Dropout(0.25))

f_model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
f_model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
f_model.add(MaxPooling2D(pool_size=(2, 2)))
f_model.add(Dropout(0.25))

f_model.add(Conv2D(128, (9, 9), padding='same', activation='relu'))
f_model.add(Conv2D(128, (9, 9), padding='same', activation='relu'))
f_model.add(MaxPooling2D(pool_size=(2, 2)))
f_model.add(Dropout(0.25))

# f_model.add(Conv2D(256, (2, 2), padding='same', activation='relu'))
# f_model.add(MaxPooling2D(pool_size=(2, 2)))
# f_model.add(Dropout(0.25))


f_model.add(GlobalAveragePooling2D())

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

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

f_model.summary()
```

7.2.3.2 Compiling and Fitting the final model

```
f_model.compile(
   loss='categorical_crossentropy',
   optimizer='adam',
   metrics=['accuracy'])
```

The above code is to compile the final model
Here we use

- Loss function : categorical_crossentropy
- Optimization function : adam function
- Metrics : accuracy

This code is to fit the model

```
f_model_results = f_model.fit_generator(
   datagen.flow(x_train, y_train, batch_size=32),
   steps_per_epoch=len(x_train) / 32,
   epochs = 20,
   validation_data = (x_validate,y_validate),
   callbacks=[checkpointer],
   verbose = 1)
```

After fitting the model this is the output we got.

- The accuracy kept on increasing with increase in epochs and reached 97.8%
- The loss function is very less and about 0.06.
- These are saved into weights.best.f_model.hdf5

7.3 Implementation of model

So far we have created two models

- Benchmark model
- Final model

The accuracy of the final model is so good and even the loss function is very less. These values looks so promising in finalising the model. So we use this final model in order to detect the vessels in the given satellite images.

The steps to be followed

- First we define some functions required to plot the marker in the image.
- We use the model to detect the vessels by parsing each and every piece of the image by traversing along the image and validating it.
- Finally we mark the pieces which have the probability of being a ship above 90%

7.3.1 Defining the required functions

These are the functions required to mark the images.

```
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 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
```

Cutting :

This function is to cut the total given image into pieces of 80x80 size. So as our model takes only the images of size 80x80 we have to reshape the image and then feed the image data into model.

Show_ship:

This method is the crucial method. This actually reads the co-ordinates and using tensors traverses along the image. If the image piece is found to be ship then the show ship method marks the image in red borders.

7.3.2 Detecting the ships

Finally we traverse the given image and move across it piece by piece of size 80x80. During each phase of checking the validity of piece of being the vessel we use the model we created before. This way we get the probability of the piece being a vessel. We can then check the probability, we set the minimum probability percentage as 90%. So if the output probability is above 90% then the piece we are checking is above par and is a vessel for sure.

```
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 = f_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 this code to execute finally

- First we have to load the image
- The image of the port in here it is port of San Francisco
- Then we have to split the data of image into small pieces of size 80x80
- Then we have to feed the each image piece into the model
- The model will calculate the probability of the image being a ship or vessel
- If it is any vessel it will have the probability greater than 0.9
- Once the vessels are found then they are encompassed by rectangular box
- Finally we get the image of the port with the ships marked in it

8. RESULTS

The final model is evaluated and fitted. Its values of fitting the data is very keen. It is about 98% of the data. It is very good and precise. Whereas the benchmark model is not so good at predicting the data. It is about 25% which is very less and can be declared as not so good model. In here we will summarize the data models.

8.1 Models Summary

The summary of the performance of the model is

Model name	Accuracy
Bench mark model	26.75%
Final model	98.75%

Since, the bench mark model didn't take more epochs. It is just 1 epoch used to training and testing the data. This made the model have less resources to update the weights of the nodes in the architecture. This made the first model not so good at predicting the right choices. On the other hand, the second model had 20 epochs which gave it quite a time to update the weights of the nodes in the architecture. This way it reached an optimum positon where it takes quite a long time in updating further. That is the reason behind the 99% accuracy.

8.2 Model Evaluation and Validation

	Precision	recall	f1-score	support
No Ship	0.99	1.00	0.99	293
Ship	0.99	0.97	0.98	107
Micro avg	0.99	0.99	0.99	400
Macro avg	0.99	0.98	0.99	400
Weighted avg	0.99	0.99	0.99	400

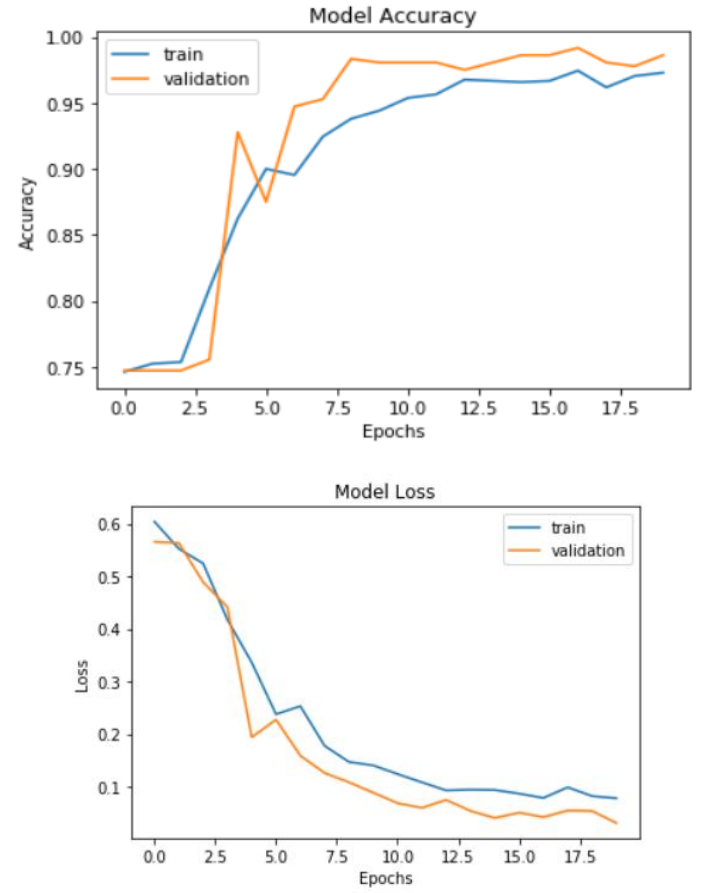
Classification report for model to evaluate results:-

As the classification is binary we got very good results almost 1. The model we have completely fits the data which is a little bit troublesome but as we have less amount of images this situation is unavoidable.
But this model is very good at detecting ships no matter what.

This model exceeds our benchmark model and also outperforms it. We can still improve the model so that it doesn't over fit.

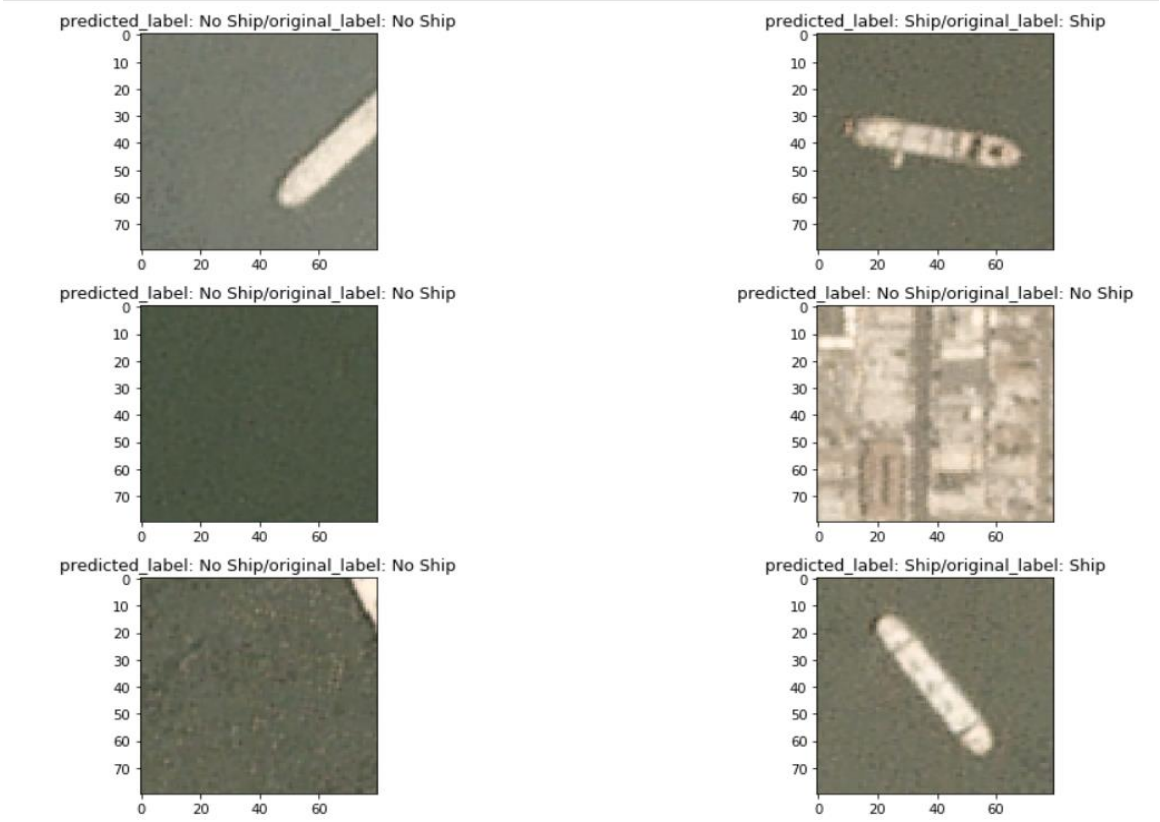
8.3 Free-Form Visualization

Loss and accuracy graphs of the model



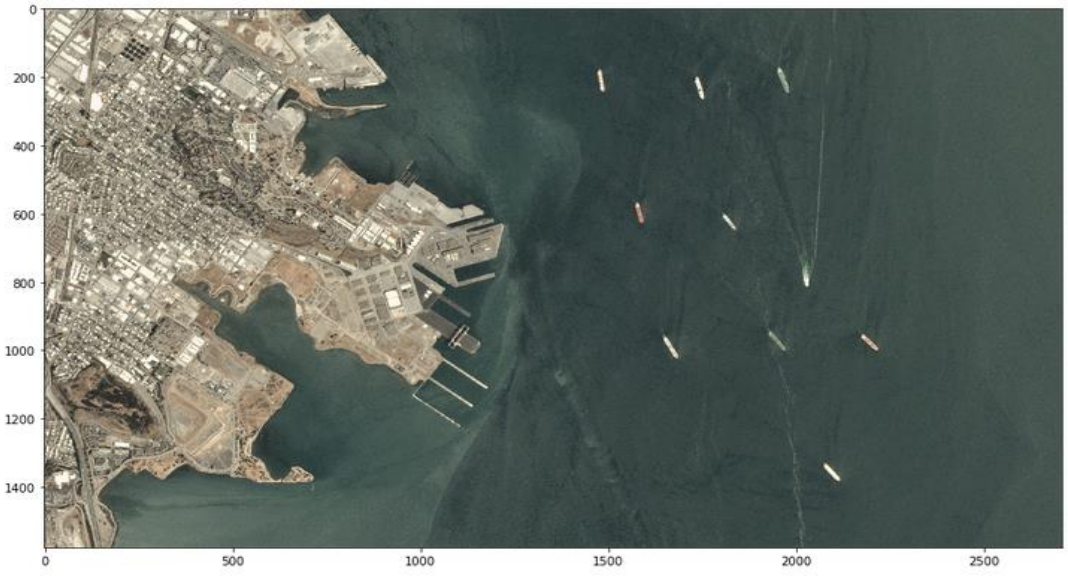
From the above figure we can see that the accuracy of the model constantly increasing with increase in epochs. And also we can see the model loss is gradually decreasing and almost reaches zero.

The below figure shows the predicted and original labels of the images by the model.



From the above figure we can clearly see that our model is very good at predicting the data.

The below image is the real image we would like to feed to the model



The below figure shows the predicted images of ships by the model

The model correctly identifies all the ships. It misclassifies only one plot as ship. It is good with remaining plots.

9. CONCLUSION

From the above working of the two models we can clearly see that the final model is outperforming the benchmark model. It has an accuracy score of 98%. For a model of this size 98% is very high, and also the time taken by the model to validate an image fed to it is almost 2 sec. This model is so accurate and very fast.

When it comes to implementation it is very good at it. It literally identifies every vessel in the scene. The recall is very high. The precision of the model is also high, we can see that at the output where it misclassified a 'no-ship' to a 'ship' class. This is the best model as recall is the key factor required for us to conclude as the best model.

10. REFERENCES

1. Abileah, 2009 R. Abileah
Surveying coastal ship traffic with LANDSAT
OCEANS 2009 (2009), pp. 1-6, 10.23919/OCEANS.2009.5422109

2. Antelo et al., 2009 J. Antelo, G. Ambrosio, J. Gonzalez, C. Galindo
Ship detection and recognition in high-resolution satellite images
IEEE International Geoscience and Remote Sensing Symposium (IGARSS) (2009), pp. 514-517, 10.1109/IGARSS.2009.5417426

3. Arshad et al., 2010 N. Arshad, K.-S. Moon, J.-N. Kim
Multiple ship detection and tracking using background registration and morphological operations
T. Kim, S.K. Pal, W.I. Grosky, N. Pissinou, T.K. Shih, D. Ślęzak (Eds.), Signal Processing and Multimedia, Communications in Computer and Information Science (2010), pp. 121-126

4. Beşbinar and Alatan, 2015 B. Beşbinar, A.A. Alatan
Inshore ship detection in high-resolution satellite images: approximation of harbors using sea-land segmentation
L. Bruzzone (Ed.), Image and Signal Processing for Remote Sensing XXI (2015), p. 12, 10.1117/12.2194928

5. Huang et al., 2011 G. Huang, Y. Wang, Y. Zhang, Y. Tian
Ship detection using texture statistics from optical satellite images
International Conference on Digital Image Computing Techniques and Applications (DICTA) (2011), pp. 507-512, 10.1109/DICTA.2011.91

6. Pan et al., 2015 B. Pan, Z. Jiang, J. Wu, H. Zhang, P. Luo
Ship Recognition Based on Active Learning and Composite Kernel SVM
T. Tan, Q. Ruan, S. Wang, H. Ma, K. Di (Eds.), Advances in Image and Graphics Technologies. IGTA 2015. Communications in Computer and Information Sciences, Springer, Berlin, Heidelberg (2015), pp. 198-207, 10.1007/978-3-662-47791-5_23