Ship Detection and Classification on Remote Sensing Images using Deep Learning

EXECUTIVE SUMMARY

Intelligent detection and classification of ships from high-resolution remote sensing images is an extraordinarily useful task in civil and military surveys. However, this job comes with a lot of hassles since various disturbances are present in the sea such as clouds, mist, islands, coastlines, ripples, and so on. Moreover, two primary setbacks in this regard are cluttered image scenes and varying ship sizes. This project utilises machine learning and neural network techniques clubbed with numerous methodologies to carry out ship detection from images with ease. The advantages and disadvantages of different detection and classification techniques of ship detection are highlighted as well. This project has been carried out under the guidance of officials from Defence Research and Development Organisation. In order to arrive at the most efficient technique to carry out the task, numerous mini-projects were also carried out in the process, the details of which have also been outlined in this report. Machine learning models such as Region Based Convolutional Neural Networks (R-CNNs), Mask R-CNNs, and You Only Look Once (YOLO) are applied on numerous datasets such as Dat-Tran's Raccoon dataset, Kaggle's Airbus dataset, and MARVEL dataset to arrive on key conclusions in this project.

1. INTRODUCTION

1.1 Objective

There is a growing need for effective ship recognition and detection for ensuring maritime security and civil management. Apart from this, there are a wide range of applications for ship detection as well right from traffic monitoring, and the defence of territory and naval battles, to harbour surveillance, fishery management, and sea pollution management [1].

Under the guidance of Defense Research and Development Organisation (DRDO), Bangalore, this project aims to develop an effective and accurate methodology using machine learning algorithms and models such as R-CNNs, Mask R-CNNs, and YOLO to meet two primary objectives -

(a) **Object Detection:** to draw a bounding box around the most feasible position of the object/ship in images.

(b) **Classification:** to classify the detected maritime object/ship into one of the classes, namely, cargo, cruise, carrier, and so on.

1.2 Motivation

In recent years, with the continuous enhancement of hardware computing power, deep learning algorithms have been rapidly developed and applied in the field of object detection. Deep learning-based methods are widely used to detect common objects in daily life and achieve extremely high performance.

Moreover, satellite and aerial remote sensing technology has developed rapidly as well, and optical remote sensing images can provide detailed information with extremely high resolution [2]. Therefore, ship detection has become a hot topic in the field of optical remote sensing. Due to the large differences between the material of the ship and sea surface in the radar images, it is easier to detect the ship target in synthetic aperture radar (SAR) images. SAR can work under all weather conditions and various climatic conditions, so ship detection is mostly completed in the SAR images. Compared to SAR images, the information provided by optical remote sensing images is more intuitive, so it is easy for humans to understand [3]. In addition, numerous satellites and unmanned aerial vehicles (UAVs) have made it possible to obtain massive high-resolution optical remote sensing images on the sea. Therefore, we can obtain more detailed information to detect the ship in the optical remote sensing images. Ship detection plays an important part in marine target monitoring.

I have been an ardent follower of the deep learning field ever since I discovered its existence during the beginning of my undergraduate years. From working on minor Kaggle projects, to carrying out a full-fledged project in collaboration with DRDO today, I have come a long way, with my upskilling process being truly fruitful at every minute level. Being able to contribute to a government agency in a field of my interest is a dream come true. This project report takes a more holistic approach while presenting key conclusions at each step of the process, since my stint at DRDO has been hugely research-oriented in nature.

1.3 Background

As a longstanding, fundamental and challenging problem in computer vision, object detection as illustrated in Fig. 1, has been an active area of research for several decades. [4]

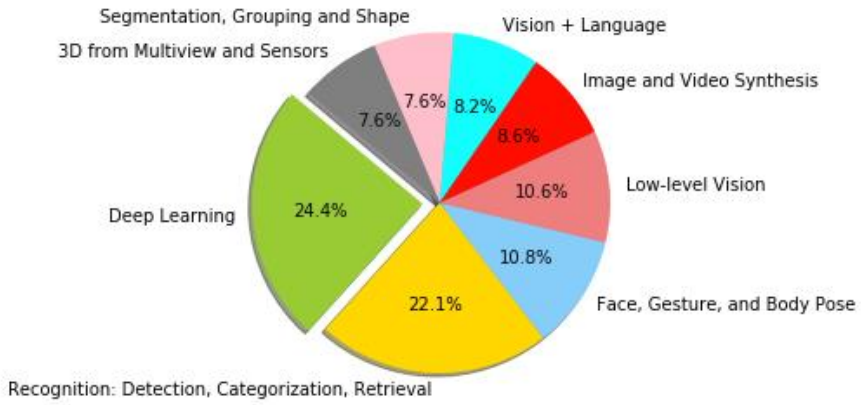


Fig. 1 – Most frequent keywords in ICCV and CVPR conference papers from 2016 to 2018. It can be seen that object detection has received significant attention in recent years.

The goal of object detection is to determine whether there are any instances of objects from given categories (such as humans, cars, bicycles, dogs or cats) in an image and, if present, to return the spatial location and extent of each object instance (e.g., via a bounding box [5]; [6]). As the cornerstone of image understanding and computer vision, object detection forms the basis for solving complex or high-level vision tasks such as segmentation, scene understanding, object tracking, image captioning, event detection, and activity recognition. Object detection supports a wide range of applications, including robot vision, consumer electronics, security, autonomous driving, human computer interaction, content-based image retrieval, intelligent video surveillance, and augmented reality. Recently, deep learning techniques ([7]; [8]) have emerged as powerful methods for learning feature representations automatically from data. In particular, these techniques have provided major improvements in object detection, as illustrated in Fig. 2.

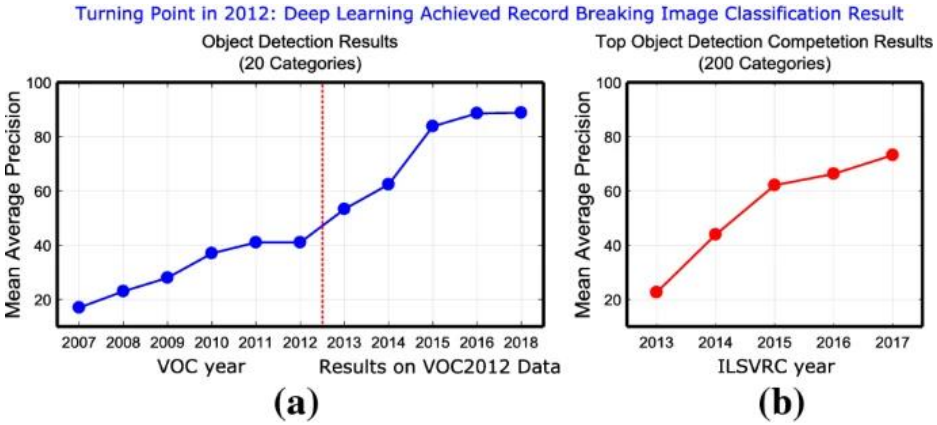


Fig. 2 - An overview of recent object detection performance: we can observe a significant improvement in performance (measured as mean average precision) since the arrival of deep learning in 2012. (**a)** Detection results of winning entries in the VOC2007-2012 competitions, and (**b)** top object detection competition results in ILSVRC2013-2017 (results in both panels use only the provided training data)

As illustrated in Fig. 3, object detection can be grouped into one of two types ([9]; [10]): detection of specific instances versus the detection of broad categories. The first type aims to detect instances of a particular object (such as Donald Trump's face, the Eiffel Tower, or a neighbour's dog), essentially a matching problem. The goal of the second type is to detect (usually previously unseen) instances of some predefined object categories (for example humans, cars, bicycles, and dogs). Historically, much of the effort in the field of object detection has focused on the detection of a single category (typically faces and pedestrians) or a few specific categories. In contrast, over the past several years, the research community has started moving towards the more challenging goal of building general purpose object detection systems where the breadth of object detection ability rivals that of humans.

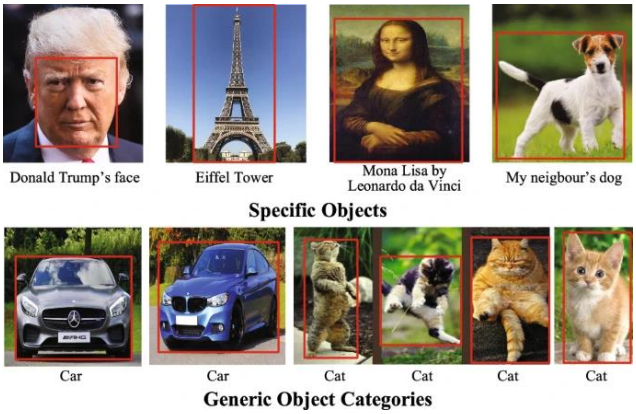


Fig. 3 - Object detection includes localizing instances of a *particular* object (top), as well as generalizing to detecting object *categories* in general (bottom).

A brief insight into the evolution of deep learning mechanisms and techniques to carry out object detection since the beginning of 2000s, has been summarised in Table 1 below. Every paper has been linked in the References section as well.

Table 1 - Summary of related object detection surveys since 2000

Year	Paper	Content
2009	Monocular pedestrian detection: survey and experiments [11]	An evaluation of three pedestrian detectors
2010	Survey of pedestrian detection for advanced driver assistance systems [12]	A survey of pedestrian detection for advanced driver assistance systems

2012	Pedestrian detection: an evaluation of the state of the art [13]	A thorough and detailed evaluation of detectors in monocular images
2002	Detecting faces in images: a survey [14]	First survey of face detection from a single image
2015	A survey on face detection in the wild: past, present and future [15]	A survey of face detection in the wild since 2000
2006	On road vehicle detection: a review [16]	A review of vision based on-road vehicle detection systems
2015	Text detection and recognition in imagery: a survey [17]	A survey of text detection and recognition in colour imagery
2007	Toward category level object recognition [18]	Representative papers on object categorization, detection, and segmentation
2009	The evolution of object categorization and the challenge of image abstraction [19]	A trace of the evolution of object categorization over 4 decades
2010	Context based object categorization: a critical survey [20]	A review of contextual information for object categorization
2013	50 years of object recognition: directions forward [21]	A review of the evolution of object recognition systems over 5 decades
2011	Visual object recognition [22]	Instance and category object recognition techniques
2013	Object class detection: a survey [23]	Survey of generic object detection methods before 2011
2015	Feature representation for statistical learning-based object detection: a review [24]	Feature representation methods in statistical learning-based object detection, including handcrafted and deep learning-based features

2014	Salient object detection: a survey [25]	A survey for salient object detection
2013	Representation learning: a review and new perspectives [26]	Unsupervised feature learning and deep learning, probabilistic models, autoencoders, manifold learning, and deep networks
2015	Deep learning [27]	An introduction to deep learning and applications
2017	A survey on deep learning in medical image analysis [28]	A survey of deep learning for image classification, object detection, segmentation and registration in medical image analysis
2017	Recent advances in convolutional neural networks [29]	A broad survey of the recent advances in CNN and its applications in computer vision, speech and natural language processing
2019	Deep learning for generic object detection [30]	A comprehensive survey of deep learning for generic object detection

2. PROJECT DESCRIPTION AND GOALS

The problem definition of object detection is to determine where objects are located in a given image (object localization), and which category each object belongs to (object classification).

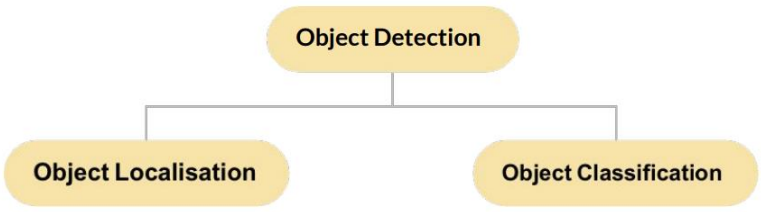


Fig. 4 – The two objectives of Object Detection

2.1 OBJECT LOCALISATION

Humans can easily detect and identify objects present in an image. The human visual system is fast and accurate and can perform complex tasks like identifying multiple objects and detect obstacles with little conscious thought. With the availability of large amounts of data, faster GPUs, and better algorithms, computers can easily be trained to detect and classify multiple objects within an image with high accuracy.

An image classification or image recognition model simply detect the probability of an object in an image. In contrast to this, object localization refers to identifying the location of an object in the image. An object localization algorithm will output the coordinates of the location of an object with respect to the image. In computer vision, the most popular way to localize an object in an image is to represent its location with the help of bounding boxes. Fig. 5 shows an example of a bounding box.

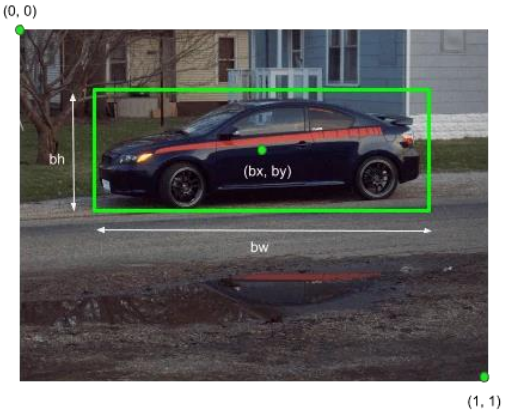


Fig. 5 - Bounding box representation used for object localization

A bounding box can be initialized using the following parameters:

a. bx, by : coordinates of the centre of the bounding box
b. bw : width of the bounding box w.r.t the image width
c. bh : height of the bounding box w.r.t the image height

A brief summary of the various machine learning algorithms used to carry out object localisation/bounding-box representation over the years can been presented in the flowchart in Fig. 6.

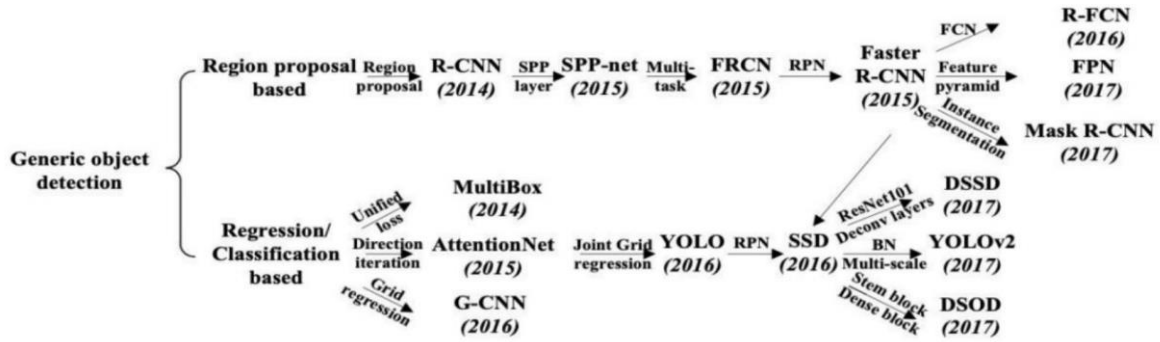


Fig. 6 – Evolution of Deep Learning Object Detection/Localisation techniques over the years

In this project, the primary focus of interest is object localisation or bounding box detection using three major and proved to be widely successful machine learning algorithms as seen in the following sections.

2.1.1 R-CNN

R-CNN or Regional Convolutional Neural Networks were majorly adopted when there were two pressing requirements persistent in the field of object localisation - (a) to improve the quality of candidate boundary boxes, and (b) to use a deep architecture for extraction of high-level features. To solve these requirements, R-CNN [31] was proposed by Ross Girshick in 2014 and obtained a mean average precision (mAP) of 53.3% with more than 30% improvement over the previous best result (DPM HSC [32]) on PASCAL VOC 2012.

Figure 7 shows the flowchart of R-CNN, which can be divided into three stages as follows:

I. **Region proposal generation:** The R-CNN adopts selective search to generate about 2k region proposals for each image. The selective search method relies on simple bottom-up grouping and saliency cues to provide more accurate candidate boxes of arbitrary sizes quickly and to reduce the searching space in object detection.

II. **CNN based deep feature extraction:** In this stage, each region proposal is warped or cropped into a fixed resolution and the CNN module in is utilized to extract a 4096-dimensional feature as the final representation. Due to large learning capacity, dominant expressive power and hierarchical structure of CNNs, a high-level, semantic and robust feature representation for each region proposal can be obtained.

III. **Classification and localization:** With pre-trained categoryspecifific linear SVMs for multiple classes, different region proposals are scored on a set of positive regions and

background (negative) regions. The scored regions are then adjusted with bounding box regression and filtered with a greedy non-maximum suppression (NMS) to produce final bounding boxes for preserved object locations.

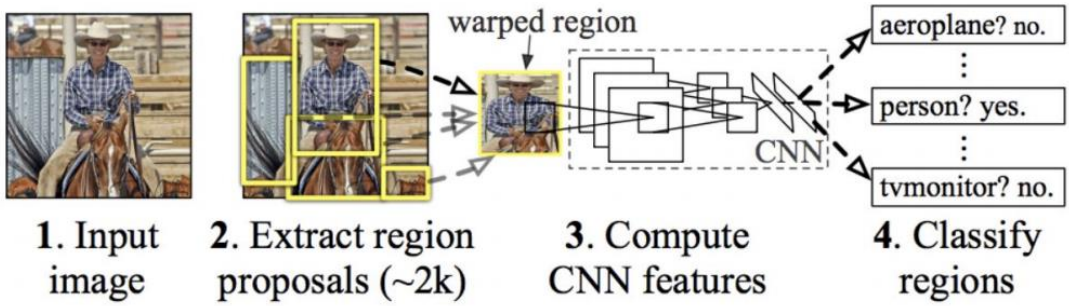


Fig. 7 – The three stages of a typical R-CNN model

The advantages and disadvantages of the R-CNN model have been discussed in the later part of the report, after its practical implementation on Dan-Tran's Raccoon Dataset in section 4.1.

2.1.2 Mask R-CNN

Instance segmentation is a challenging task that requires detecting all objects in an image and segmenting each instance (semantic segmentation). These two tasks are usually regarded as two independent processes. The most viable option before the discovery of Mask R-CNNs, i.e., the multi-task scheme had a disadvantage that would create spurious edge and exhibit systematic errors on overlapping instances.

To solve this problem, parallel to the already existing branches in Faster R-CNN for classification and bounding box regression, the Mask R-CNN adds an additional branch to predict segmentation masks in a pixel-to-pixel manner as seen in Figure 8. Different from the other two branches which are inevitably collapsed into short output vectors by FC layers, the segmentation mask branch encodes an $m \times m$ mask to maintain the explicit object spatial layout. This kind of fully convolutional representation requires fewer parameters but is more accurate than that of traditional R-CNN.

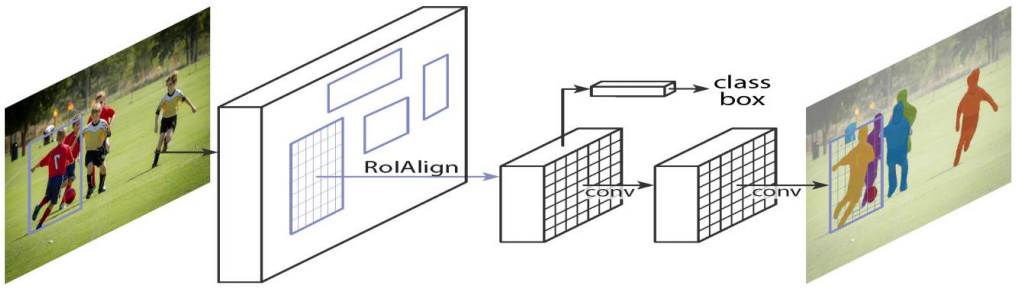


Fig. 8 – The Mask R-CNN framework for Instance Segmentation

Formally, besides the two losses for classification and bounding box regression, an additional loss for segmentation mask branch is defined to reach a multi-task loss. And this loss is only associated with ground-truth class and relies on the classification branch to predict the category. Because RoI pooling, the core operation in Faster R-CNN, performs a coarse spatial quantization for feature extraction, misalignment is introduced between the RoI and the features. It affects classification little because of its robustness to small translations. However, it has a large negative effect on pixel-to-pixel mask prediction. To solve this problem, Mask R-CNN adopts a simple and quantization-free layer, namely RoIAlign, to preserve the explicit per-pixel spatial correspondence faithfully. RoIAlign is achieved by replacing the harsh quantization of RoI pooling with bilinear interpolation, computing the exact values of the input features at four regularly sampled locations in each RoI bin.

In spite of its simplicity, this seemingly minor change improves mask accuracy greatly, especially under strict localization metrics. Given the Faster R-CNN framework, the mask branch only adds a small computational burden and its cooperation with other tasks provides complementary information for object detection. As a result, Mask R-CNN is simple to implement with promising instance segmentation and object detection results. In a nutshell, Mask R-CNN is a flexible and efficient framework for instance-level recognition, which can be easily generalized to other tasks (e.g., human pose estimation) with minimal modification.

The advantages and disadvantages of the Mask R-CNN model have further been discussed in the later part of the report, after its practical implementation on the Airbus Kaggle Dataset in section 4.2.

2.1.3 YOLO

YOLO (You Only Look Once) uses a totally different approach. YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm

applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This model has a number of benefits over other object detection methods:

- YOLO is extremely fast
- YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.
- YOLO learns generalizable representations of objects so that when trained on natural images and tested on artwork, the algorithm outperforms other top detection methods.

The major concept of YOLO is to build a CNN network to predict a (7, 7, 30) tensor. It uses a CNN network to reduce the spatial dimension to 7×7 with 1024 output channels at each location. YOLO performs a linear regression using two fully connected layers to make 7×7×2 boundary box predictions (the middle picture below in Figure 9). To make a final prediction, we keep those with high box confidence scores (greater than 0.25) as our final predictions (the right picture in Figure 9).

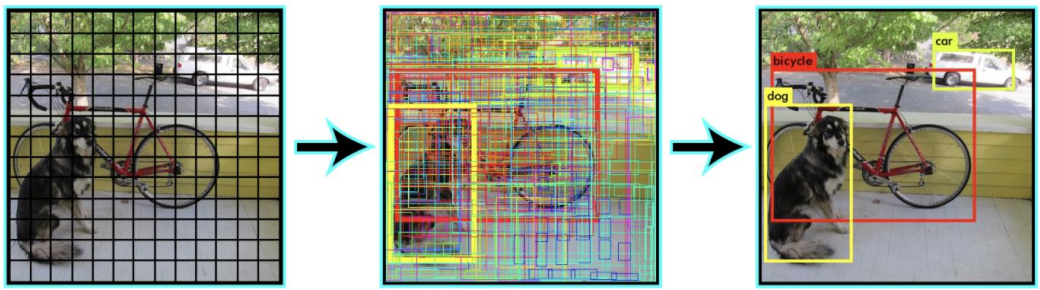


Fig. 9 – YOLO Detection Algorithm

The **class confidence score** for each prediction box is computed as:

$$\text{class confidence score} = \text{box confidence score} \times \text{conditional class probability}$$

YOLO measures the confidence on both the classification and the **localization** (where an object is located). Moreover, YOLO has 24 convolutional layers followed by 2 fully connected layers (FC). Some convolution layers use 1 × 1 reduction layers alternatively to reduce the depth of the features maps. For the last convolution layer, it outputs a tensor with shape (7, 7, 1024). The tensor is then flattened. Using 2 fully connected layers as a form of linear regression, it outputs 7×7×30 parameters and then reshapes to (7, 7, 30), i.e. 2 boundary box predictions per location. This can be visualised in Figure 10.

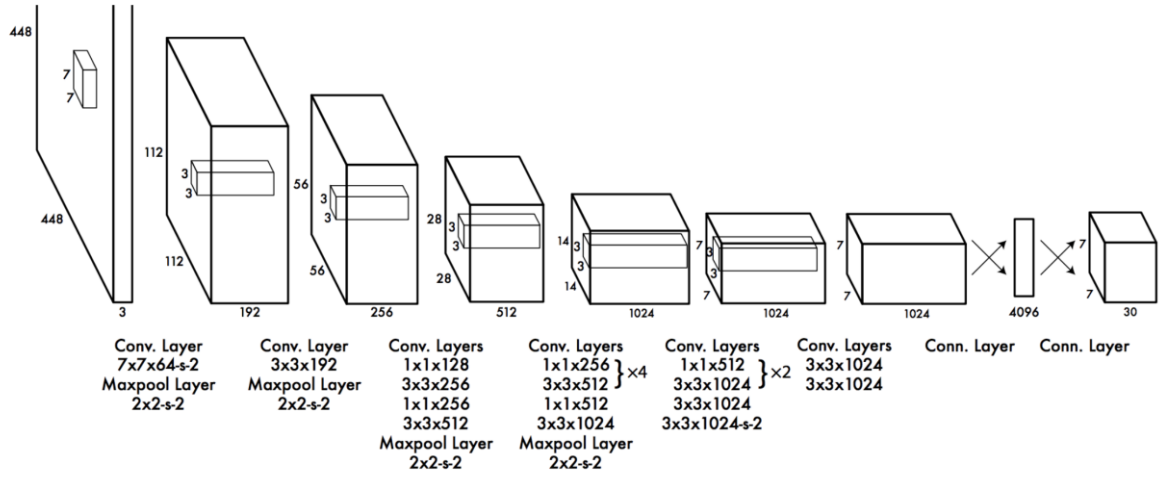


Fig. 10 – YOLO Algorithm Architecture

A deeper insight into the advantages and disadvantages of the YOLO model have been discussed in the later part of the report, after its practical implementation on the Marvel Dataset in section 4.3.

2.2 OBJECT CLASSIFICATION

<to be filled later>

<After CNN Classifier implementation>

3. TECHNICAL SPECIFICATION

3.1 Laptop Specifications – MacBook Pro 2017

Display

- Retina display
- 13.3-inch (diagonal) LED-backlit display with IPS technology; 2560-by-1600 native resolution at 227 pixels per inch with support for millions of colors
- Supported scaled resolutions:
    - 1680 by 1050
    - 1440 by 900
    - 1024 by 640
- 500 nits brightness
- Wide colour (P3)

Processor

- 2.3GHz dual-core Intel Core i5, Turbo Boost up to 3.6GHz, with 64MB of eDRAM
  Configurable to 2.5GHz dual-core Intel Core i7, Turbo Boost up to 4.0GHz, with 64MB of eDRAM

Storage

- 128GB
  128GB SSD
  Configurable to 256GB, 512GB, or 1TB SSD

Memory

- 8GB of 2133MHz LPDDR3 onboard memory
  Configurable to 16GB of memory

Graphics

- Intel Iris Plus Graphics 640

Operating System

- macOS is the operating system that powers everything one does on a Mac. macOS Mojave brings new features inspired by its most powerful users but designed for everyone.

3.2 Google Colaboratory

I have primarily relied on Google Colaboratory to implement multiple ML Models throughout my project duration.

Colaboratory, or "**Colab**" for short, is a product from Google Research. **Colab** allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

Fig. 11 – Google Colaboratory Logo

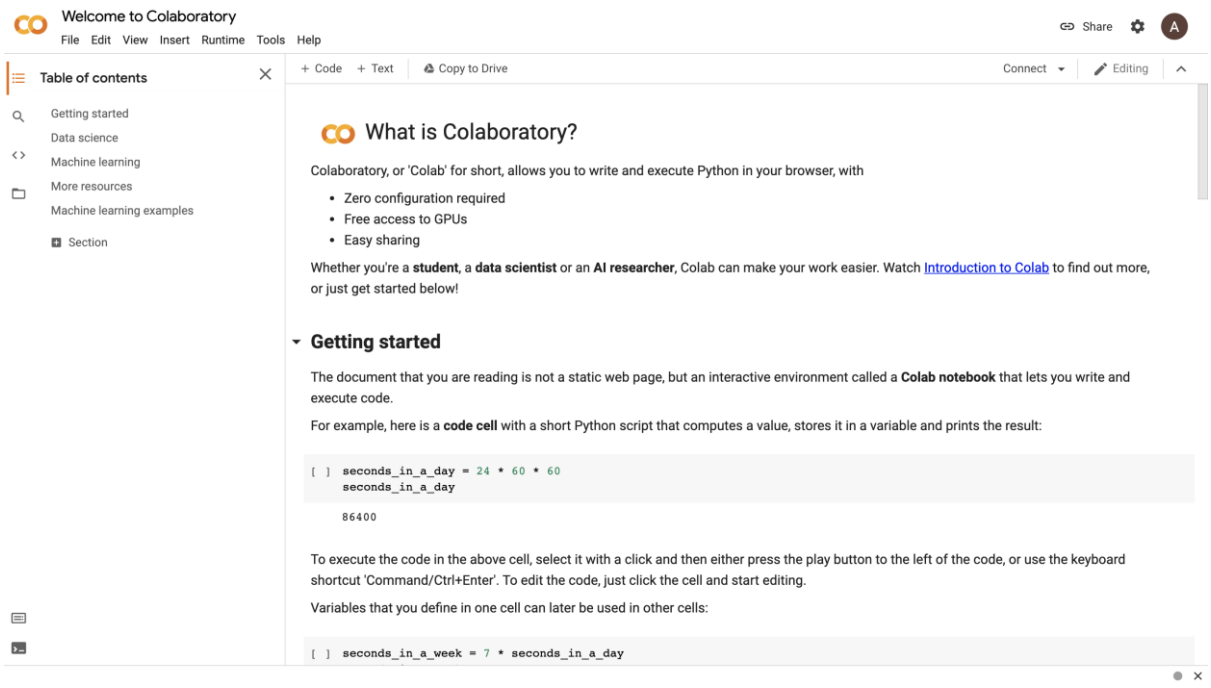


Fig. 12 – A snippet of the Interface

3.2 Advantages of using Google Colaboratory

I. Easy to Share across multiple platforms

One can share his/her Google Colab notebooks very easily. Thanks to Google Colab everyone with a Google account can just copy the notebook on his/her own Google Drive account. No need to install any modules to run any code, modules come preinstalled within Google Colab.

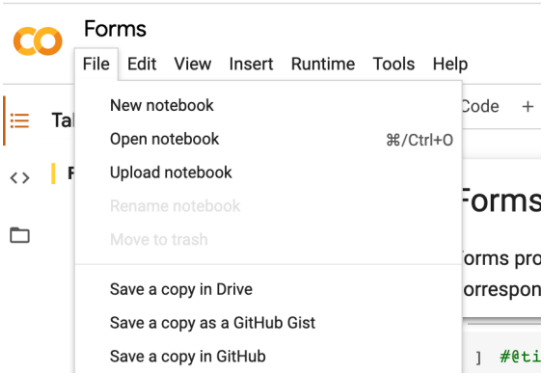


Fig. 13 – Easily transferable to Drive or GitHub

II. Pre-Installed Libraries

One of the main benefits of using Colab is that it has most of the common libraries that are needed for machine learning like *TensorFlow*, *Keras*, *ScikitLearn*, *OpenCV*, *numpy*, *pandas*, etc. pre-installed. Having all of these dependencies means that one can just open a notebook and start coding without having to set up anything at all. Any libraries that are not pre-installed can also be installed using standard terminal commands. While the syntax for executing terminal commands remains the same, you must add an exclamation mark (!) at the start of the command so that the compiler can identify it as a terminal command.

Fig. 14 – The pre-installed libraries that come with Colab

III. Cloud-Based Interface

Another feature is that the *Colab* environment is independent of the computing power of the local computer itself. Since it is a cloud-based system, as long as one has internet connectivity, he/she can run even heavy machine learning operations from a relatively old computer that, ordinarily, wouldn't be able to handle the load of executing those operations locally. Additionally, Google also offers a **GPU (*Graphics Processing Unit*)** and a **TPU (*Tensor***

***Processing Unit*)** for free. These hardware accelerators can enable us to run heavy machine learning operations on large datasets much faster than any local environment.

4. DESIGN APPROACH AND DETAILS

4.1 Phase 1: Implementation of R-CNN

4.1.1 Objective

The first project that I ever worked on at DRDO, this gave me the opportunity to dive deep into object detection and enhance my skills of Tensorflow and Keras. R-CNNs serve as a great starting point to solve any major object detection problem, due to its well-maintained documentation and varied use cases. I was advised by Dr. Rajesh and Shri. Dhipu at DRDO to begin my research by first modelling R-CNN on a small-scale dataset.

4.1.2 Dataset

The dataset that I chose for this objective was the Dat-Tran's 2017 Raccoon Dataset, that consists of 196 images of raccoons in varied sizes, shapes and orientation. Owing to more than a single raccoon present in a few images, the total number of bounding boxes in the ground-truth dataset equals to 213. As it can be observed, this is a single class problem, and a great starting point to perform and explore object detection. The images in this dataset vary from 0.04 to 2.67 megapixels and the median image size can be approximated at 480 x 360, i.e., 0.18 megapixels. A small snippet of the dataset can be seen in Figure 15.

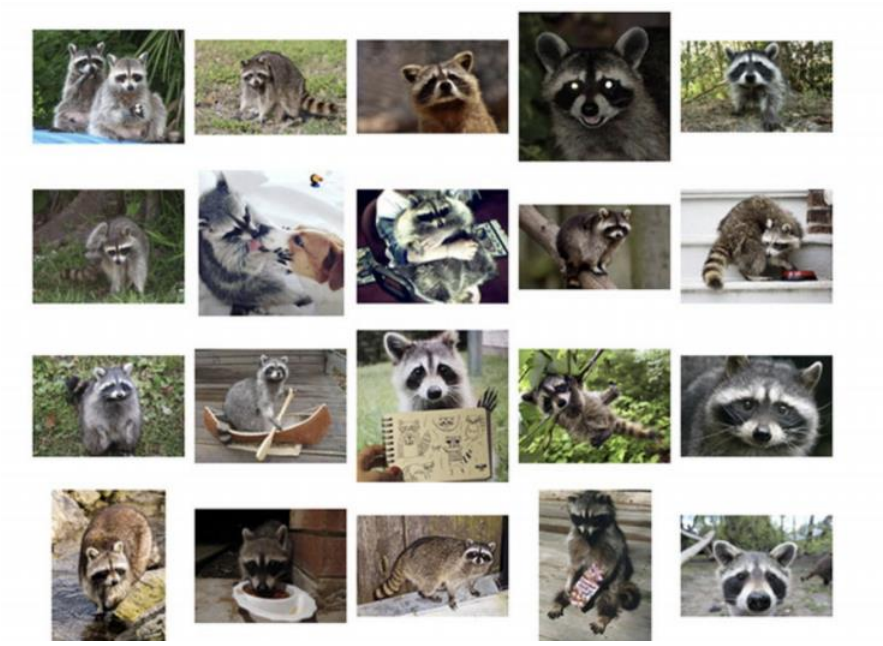


Fig. 15 – A snippet of Dat-Tran's Raccoon Dataset

4.1.3 Methodology and Code Snippets

Implementing an R-CNN object detector is a somewhat complex multistep process. A summary of the flow that has been followed by me has been summarised in the below flowchart, as shown in Figure 16.

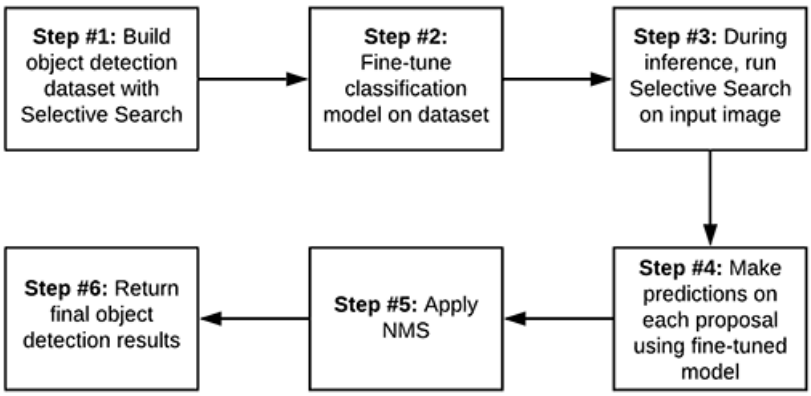


Fig. 16 – The Object Detection Workflow

Firstly, I used Selective Search, along with a bit of post-processing logic, to identify regions of an input image that *do* and *do not* contain a potential object of interest, in this case, a raccoon. These regions were used as the training data for the object detection model.

Next, a MobileNet (pre-trained on ImageNet) was fine-tuned to classify and recognise objects from the above curated dataset.

Finally, the last step involves implementing a Python script for inference and prediction by applying Selective Search to an input image, classifying the region proposals generates, and then to display the final output of the R-CNN.

The accuracy of the object detection algorithm was measured by using a metric called "**Intersection over Union (IOU)**", that can be calculated using the below-mentioned formula.

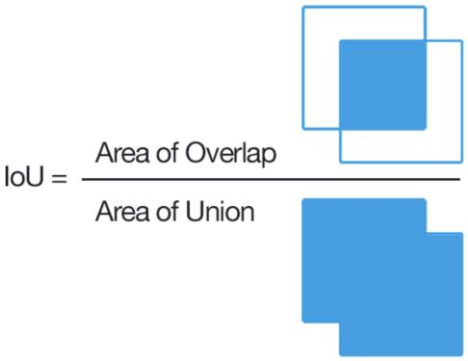


Fig. 17 – IOU Equation

Examining the above equation and observing Figure 18, it can be seen that Intersection over Union (IOU) is simply a ratio:

- In the numerator, the **area of overlap** between the predicted bounding box and the ground-truth bounding box is computed.
- The denominator is the **area of union**, or more simply, the area encompassed by *both* the predicted bounding box and the ground-truth bounding box.
- Dividing the area of overlap by the area of union yields the final score — ***the Intersection over Union*** (hence the name).

In this project, IOU is used to measure object detection accuracy, including how much a given Selective Search proposal overlaps with a ground-truth bounding box (which is useful when generating positive and negative examples for the training data).

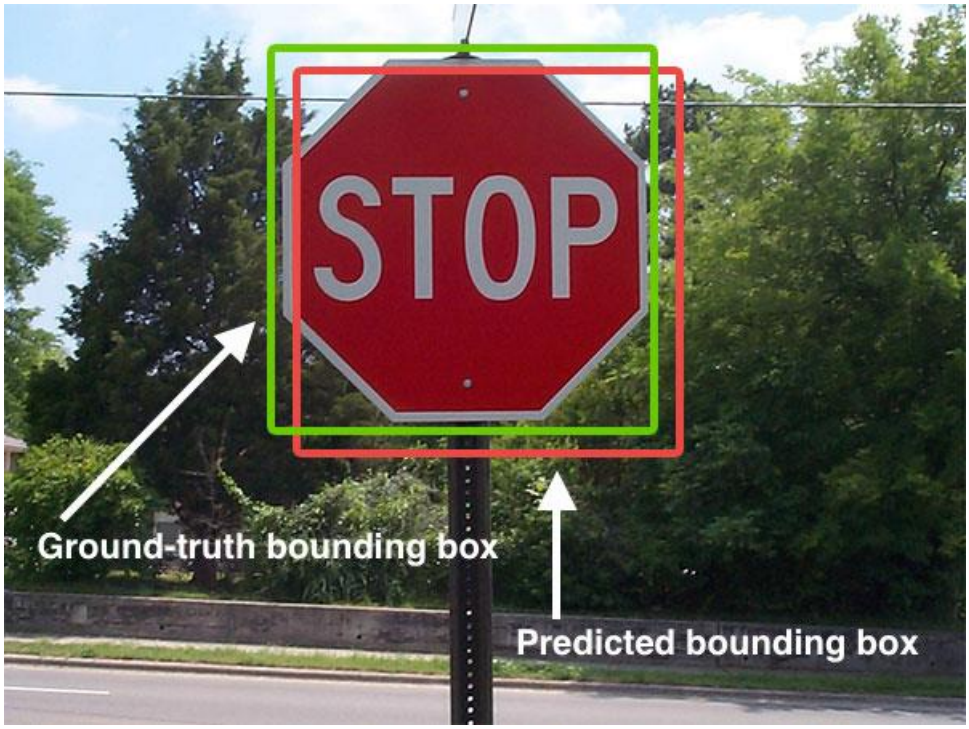


Fig. 18 – Ground-truth and Predicted bounding boxes of an image

The Function defined in Python for detecting IOU for any image in an object detection problem has been elaborated below, in Figure 19.

```
def compute_iou(boxA, boxB):
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the intersection area
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou
```

Fig. 19 – Code snippet of the IOU function

Another important concept that I implemented during the course of this project is the **"Non-Maxima Suppression (NMS)."** Any typical Object detection pipeline has one component for

generating proposals for classification. Proposals are nothing but the candidate regions for the object of interest. Most of the approaches employ a sliding window over the feature map and assigns foreground/background scores depending on the features computed in that window. The neighbourhood windows have similar scores to some extent and are considered as candidate regions. This leads to hundreds of proposals. As the proposal generation method should have high recall, it is often advised to keep loose constraints in this stage. However, processing these many proposals all through the classification network is cumbersome. This leads to a technique which filters the proposals based on some criteria called Non-maximum Suppression. This phenomenon can be observed in Figure 20.

Fig. 20 – A summary of NMS

In the Results section, more about the effects of NMS on this particular project have been outlined.

4.1.4 Results

```
[INFO] loading images...
[INFO] compiling model...
[INFO] training head...
Train for 94 steps, validate on 752 samples
Train for 94 steps, validate on 752 samples
Epoch 1/5
94/94 [==============================] - 77s 817ms/step - loss: 0.3072 - accuracy: 0.8647 -
val_loss: 0.1015 - val_accuracy: 0.9728
Epoch 2/5
94/94 [==============================] - 74s 789ms/step - loss: 0.1083 - accuracy: 0.9641 -
val_loss: 0.0534 - val_accuracy: 0.9837
Epoch 3/5
94/94 [==============================] - 71s 756ms/step - loss: 0.0774 - accuracy: 0.9784 -
val_loss: 0.0433 - val_accuracy: 0.9864
Epoch 4/5
94/94 [==============================] - 74s 784ms/step - loss: 0.0624 - accuracy: 0.9781 -
val_loss: 0.0367 - val_accuracy: 0.9878
Epoch 5/5
94/94 [==============================] - 74s 791ms/step - loss: 0.0590 - accuracy: 0.9801 -
val_loss: 0.0340 - val_accuracy: 0.9891
[INFO] evaluating network...
              precision    recall  f1-score   support

   no_raccoon       1.00      0.98      0.99       440
      raccoon       0.97      1.00      0.99       312

     accuracy                           0.99       752
    macro avg       0.99      0.99      0.99       752
 weighted avg       0.99      0.99      0.99       752

[INFO] saving mask detector model...
[INFO] saving label encoder...

real    6m37.851s
user    31m43.701s
sys     33m53.058s
```

Fig. 21 – Results obtained upon execution

Upon training and testing the model for 5 epochs, results that have been highlighted in Figure 21 were obtained. A graph depicting the training loss and accuracy obtained during every epoch, the model was trained for, has been depicted in Figure 22, as seen below. Once the accuracy and the loss ceased to change, as seen in the below graph, the training of the model was stopped.

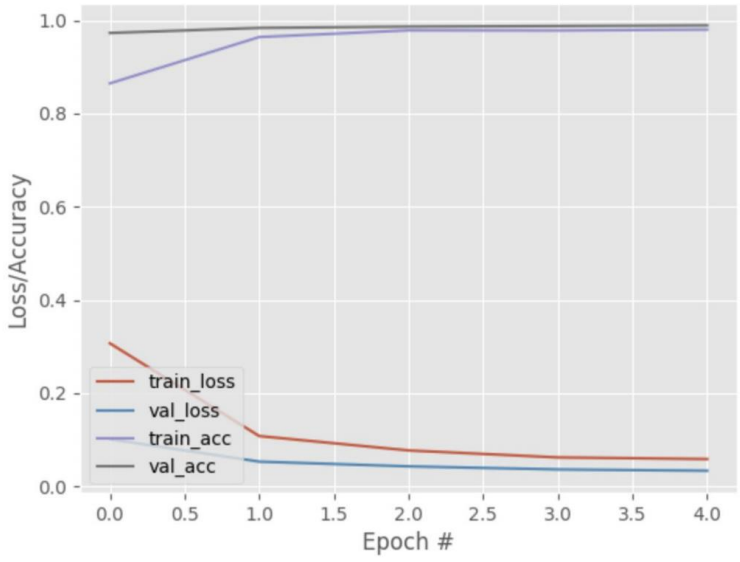


Fig. 22 – Training Loss and Accuracy of the model

Figure 23 and 24 show snippets of a few output images with the bounding box as well as the IOU labelled on it.

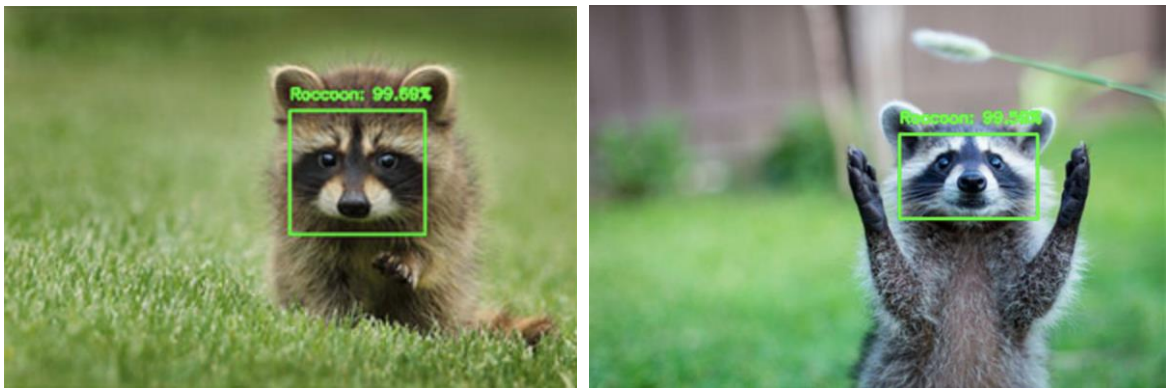


Fig. 23 – A few output images of the R-CNN model

A key point to note is the importance of Non-Maxima Suppression (NMS). As seen in Figure 20, NMS plays an invaluable role to eliminate lesser accurate bounding boxes during object detection. A before-NMS and an after-NMS comparison has been made in Figure 24.

Fig. 24 - (a) Before NMS was applied, **(b)** After NMS was applied

As it can be seen, the redundant bounding box was eliminated upon applying NMS.

4.1.5 Key Takeaways

1. The Raccoon project taught me how to implement and use Tensorflow as well as Keras. It was my first project at DRDO, so it also helped me get accustomed to the working environment, and pick up a few skills that I wasn't used to before.

2. Even though the accuracy of my model was great and the results shown were promising, it was mostly because of the smaller size of the dataset, with easily distinguishable objects in it.
3. As the next step, I was advised by mentors at DRDO to try my R-CNN model on databases that involved air-borne objects such as helicopters, aeroplanes as well as the ones that involved ships, to inch closer to the requirements of my problem statement of ship detection.

4.2 Phase 2: Implementation of Mask R-CNN

4.2.1 Objective

Upon the successful implementation of R-CNNs, I took one further step in the direction of solving my problem statement of ship detection. This time around, the moderately-sized dataset contained of satellite images of ships, in contrast to the small-size dataset of Raccoons used previously. Moreover, the Mask R-CNN algorithm is well-suited to tackle the problem of object detection such as ships and other maritime objects.

4.2.2 Dataset

For this phase, a public dataset from Kaggle on the Airbus Ship Detection Challenge is obtained. The dataset contains more than 100 thousand 768 × 768 images taken from satellite. The total size of the dataset is more than 30 Gb. Along with the images in the dataset, is a CSV file that lists all the images ids and their corresponding pixels coordinates. These coordinates represent segmentation bounding boxes of ships. Not having pixel coordinates for an image means that particular image doesn't have any ships.

4.2.3 Methodology and Code Snippets

The methodology I followed has been outlined in the following steps -

Step 1**:** Initial Setup

I downloaded the dataset, installed required dependencies and directory files, using a Python script.

Step 2: Splitting into Training and Validation sets

The dataset is split into training and validation set with 80% and 20% images respectively. Since a few images had more than just a single ship, it was important to split each category of number of ships by a 80:20 ratio. This can be observed in Figure 25.

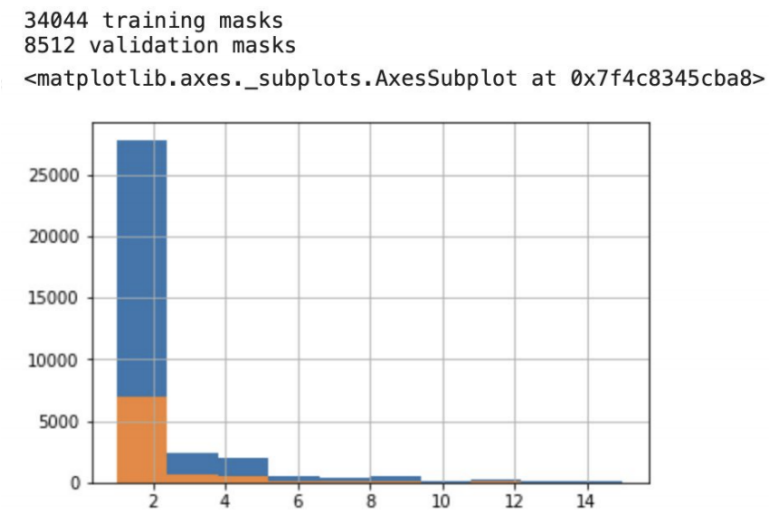


Fig. 25 – Graph depicting the 80:20 split

Blue denotes the training set and orange denotes the validation set. I generated 34,044 training images and 8,512 validation images. The X-axis denotes the number of ship masks in each image.

Step 3: Applying Masks to all images

The next step of the process was to apply masks to all images using a simple function written in Python. The code has been explained in Figure 26, and the output ship images with masks have been represented in Figure 27.

```
def masks_as_image(in_mask_list, shape=SHAPE):
    all_masks = np.zeros(shape, dtype = np.int16)
    for mask in in_mask_list:
        if isinstance(mask, str):
            all_masks += rle_decode(mask)
    return np.expand_dims(all_masks, -1)
```

Fig. 26 – A Code Snippet of the Mask Generation Function

Fig. 27 – A few examples of masks generated for ship images

Step 4: Encoding and Decoding Masks

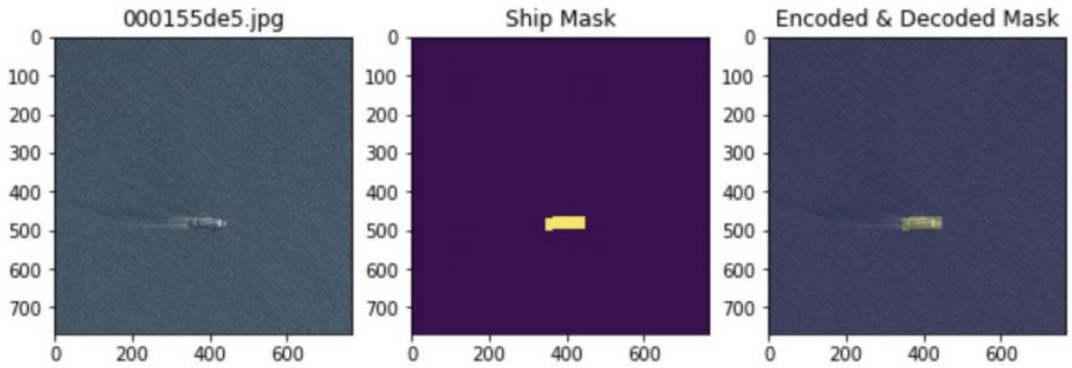
Next, a function is created for encoding and decoding mask on top of every image then Mask R-CNN for segmenting the ships in image with a confidence score between 0 and 1

is set. The time required to mask out ships in the complete dataset was 9.82 second. While the function in Python to do so has been presented in Figure 28, a few encoded and decoded masks are displayed in Figure 29.

```
def shows_decode_encode(image_id, path=TRAIN_DATA_PATH):

    fig, axarr = plt.subplots(1, 3, figsize = (10, 5))
    img_0 = imread(os.path.join(path, image_id))
    axarr[0].imshow(img_0)
    axarr[0].set_title(image_id)
    rle_1 = masks.query('ImageId=="{}"'.format(image_id))['EncodedPixels']
    img_1 = masks_as_image(rle_1)
    axarr[1].imshow(img_1[:, :, 0])
    axarr[1].set_title('Ship Mask')
    rle_2 = multi_rle_encode(img_1)
    img_2 = masks_as_image(rle_2)
    axarr[2].imshow(img_0)
    axarr[2].imshow(img_2[:, :, 0], alpha=0.3)
    axarr[2].set_title('Encoded & Decoded Mask')
    plt.show()
    print(image_id , ' Check Decoding->Encoding',
          'RLE_0:', len(rle_1), '->',
          'RLE_1:', len(rle_2))
```

Fig. 28 – A Code Snippet with the Encode-Decode Mask Function



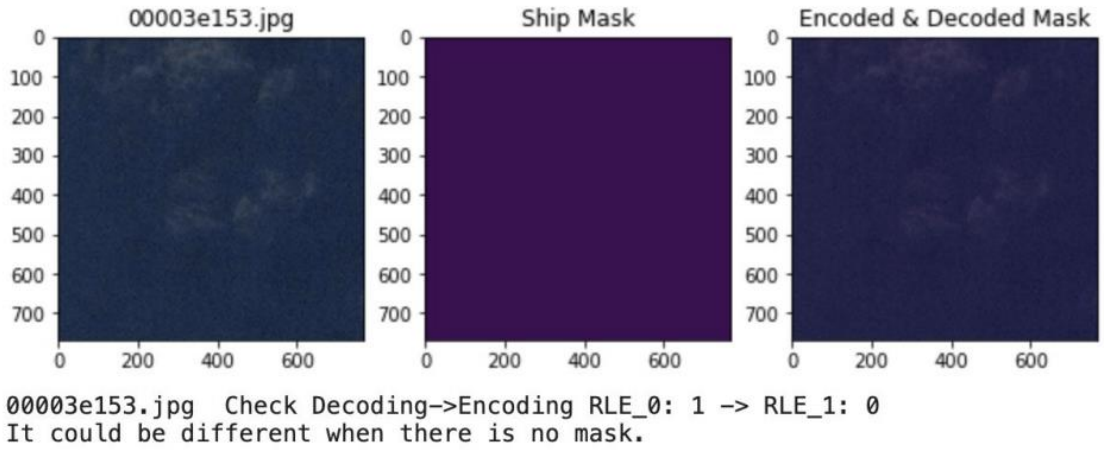


Fig. 29 – Encoded and Decoded Ship Masks

Step 5: Training the Model

Pre-trained MS COCO weights are used for training the model. Since these weights have already been trained on a large variety of objects, hence they provide a good place to start learning from. The results obtained with each passing epoch during training has been mentioned in Figure 30. The model was trained for 5 epochs.

```
Epoch 1/5
300/300 [==============================] - 320s 1s/step - loss: 1.1494 - rpn_class_loss: 0.0137 - rpn_bbox_loss: 0.4088 - mrcnn_class_loss:
0.0550 - mrcnn_bbox_loss: 0.3604 - mrcnn_mask_loss: 0.3115 - val_loss: 1.3324 - val_rpn_class_loss: 0.0168 - val_rpn_bbox_loss: 0.6377 - val_
mrcnn_class_loss: 0.0551 - val_mrcnn_bbox_loss: 0.2997 - val_mrcnn_mask_loss: 0.3232
Epoch 2/5
300/300 [==============================] - 253s 842ms/step - loss: 0.9392 - rpn_class_loss: 0.0126 - rpn_bbox_loss: 0.3684 - mrcnn_class_los
s: 0.0413 - mrcnn_bbox_loss: 0.2426 - mrcnn_mask_loss: 0.2743 - val_loss: 1.0614 - val_rpn_class_loss: 0.0109 - val_rpn_bbox_loss: 0.5029 - v
al_mrcnn_class_loss: 0.0233 - val_mrcnn_bbox_loss: 0.2541 - val_mrcnn_mask_loss: 0.2701
Epoch 3/5
300/300 [==============================] - 250s 833ms/step - loss: 0.8708 - rpn_class_loss: 0.0102 - rpn_bbox_loss: 0.3276 - mrcnn_class_los
s: 0.0342 - mrcnn_bbox_loss: 0.2261 - mrcnn_mask_loss: 0.2727 - val_loss: 0.9469 - val_rpn_class_loss: 0.0096 - val_rpn_bbox_loss: 0.3484 - v
al_mrcnn_class_loss: 0.0396 - val_mrcnn_bbox_loss: 0.2488 - val_mrcnn_mask_loss: 0.3005
Epoch 4/5
300/300 [==============================] - 249s 829ms/step - loss: 0.9003 - rpn_class_loss: 0.0098 - rpn_bbox_loss: 0.3483 - mrcnn_class_los
s: 0.0413 - mrcnn_bbox_loss: 0.2233 - mrcnn_mask_loss: 0.2777 - val_loss: 0.9566 - val_rpn_class_loss: 0.0088 - val_rpn_bbox_loss: 0.3909 - v
al_mrcnn_class_loss: 0.0271 - val_mrcnn_bbox_loss: 0.2334 - val_mrcnn_mask_loss: 0.2963
Epoch 5/5
300/300 [==============================] - 247s 822ms/step - loss: 0.9356 - rpn_class_loss: 0.0104 - rpn_bbox_loss: 0.3824 - mrcnn_class_los
s: 0.0289 - mrcnn_bbox_loss: 0.2296 - mrcnn_mask_loss: 0.2843 - val_loss: 1.0667 - val_rpn_class_loss: 0.0103 - val_rpn_bbox_loss: 0.4062 - v
al_mrcnn_class_loss: 0.0382 - val_mrcnn_bbox_loss: 0.2681 - val_mrcnn_mask_loss: 0.3439
Train model: 1495.0196392536163
```

Fig. 30 – Training the Mask RCNN Model

4.2.4 Results

It is found that using mask shape of size 14*14 and Resnet101 backbone, the Mask RCNN algorithm is able to detect and segment ships with a mean average precision of 0.605. It took the model approximately 30 seconds to segment 30 images. The graph depicting training loss versus epoch has been visualised in Figure 31, while a few snippets of the output images have been presented in Figure 32.

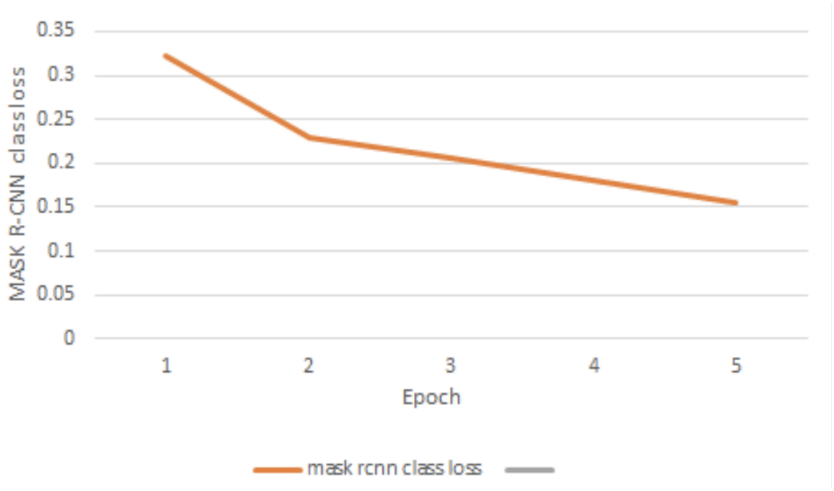
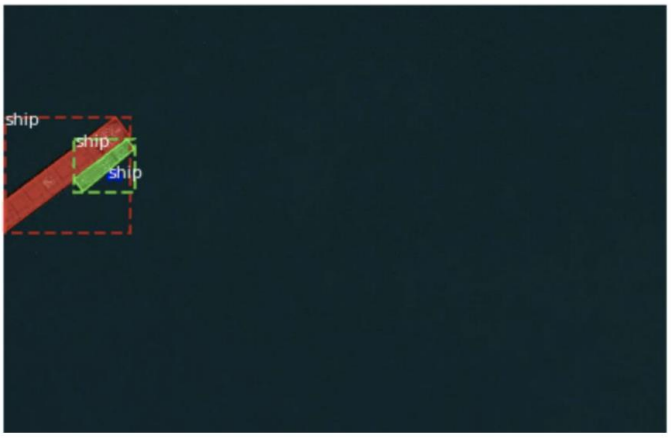


Fig. 31 – Training Loss vs Epoch for Mask R-CNN

```
Re-starting from epoch 3
original_image           shape: (768, 768, 3)         min:    0.00000  max:  250.00000  uint8
image_meta               shape: (14,)                 min:    0.00000  max: 7698.00000  int64
gt_class_id              shape: (3,)                  min:    1.00000  max:    1.00000  int32
gt_bbox                  shape: (3, 4)                min:    0.00000  max:  263.00000  int32
gt_mask                  shape: (768, 768, 3)         min:    0.00000  max:    1.00000  bool
```



```
Processing 1 images
image                    shape: (768, 768, 3)         min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 768, 768, 3)      min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max:  768.00000  int64
anchors                  shape: (1, 147312, 4)        min:   -0.47202  max:    1.38858  float32
```

```
Processing 1 images
image                    shape: (768, 768, 3)         min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 768, 768, 3)      min: -107.70000  max:  147.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max:  768.00000  int64
anchors                  shape: (1, 147312, 4)        min:   -0.47202  max:    1.38858  float32

*** No instances to display ***
```

```
Processing 1 images
image                    shape: (768, 768, 3)         min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 768, 768, 3)      min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max:  768.00000  int64
anchors                  shape: (1, 147312, 4)        min:   -0.47202  max:    1.38858  float32
```

Fig. 32 – A few snippets of the Mask R-CNN results

4.2.5 Key Takeaways

1. Even though the model was trained for 5 epochs till the loss was minimal, a few problems persist. In a few images, the land in and around the ships has also been getting detected, as seen in the Figure 33. A workaround for these false positives needed to worked out.

2. Moreover, MASK R-CNN involves generation of mask as images that takes up a lot of storage space as well as computation power. It is also extremely slow to execute.

Fig. 33 – Drawback of using Mask R-CNN (Generation of False Positive Masks)

4.3 Phase 3: Implementation of YOLO

4.3.1 Objective

For the final phase of the project, I decided to go ahead with YOLOv4 algorithm since using the YOLOv4 model has several advantages over general classifier-based systems.

For example, YOLO looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Mask R-CNN.

4.3.2 Dataset

After my research with small-sized and medium-sized databases, I used a large-scale Image dataset for Maritime Vessels, known as the MARVEL dataset. It consists of 2 million user uploaded images and their attributes including vessel identity, type, photograph category and year of built, collected from a community website. It can be categorized into 109 vessel type classes and 26 super-classes by combining heavily populated classes with a semi-automatic clustering scheme. A snippet of the dataset has been presented in Figure 34.

However, DRDO handed over a mini-version of the MARVEL dataset for me to perform my pre-liminary analysis. This consisted of 2,800 images, spread uniformly across five major maritime classes, namely, cargo, carrier, cruise, military and tanker ships.

Fig. 34 – The Marvel Dataset

4.3.3 Methodology and Code Snippets

<u>Step 1</u>**:** <u>Labelling of Ground-Truth Images</u>

Since the MARVEL dataset did not come with pre-assigned labels, like other custom-made datasets, I had to label each image manually using a tool known as LabelMe. An open

annotation tool developed by MIT, the goal of the open-source project was to develop datasets for computer vision and image processing.

A snippet of the interface has been provided in Figure 35.

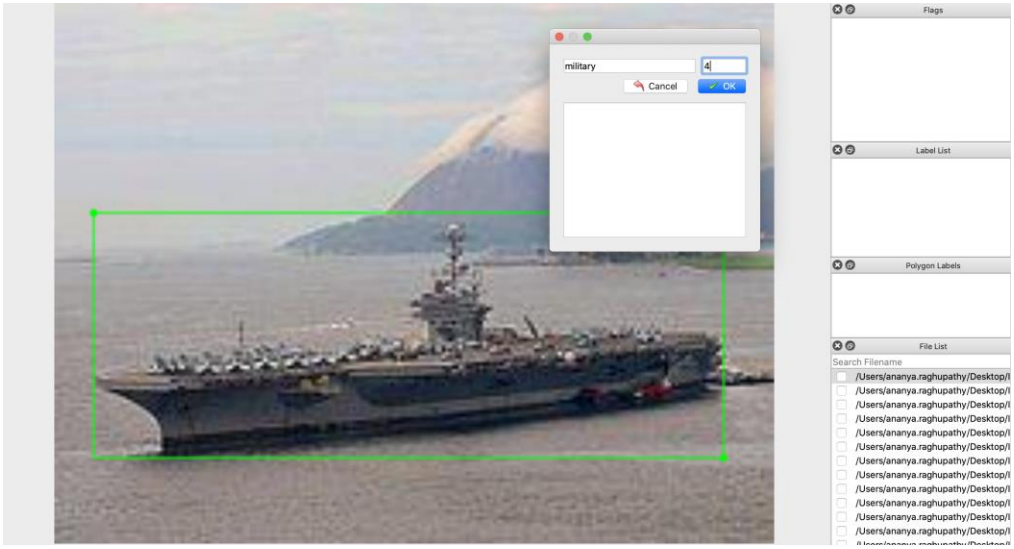


Fig. 35 – The LabelMe Annotation Tool Interface

The output of the LabelMe tool saves the coordinates of the bounding box of an image in a json file that looks like Figure 36.

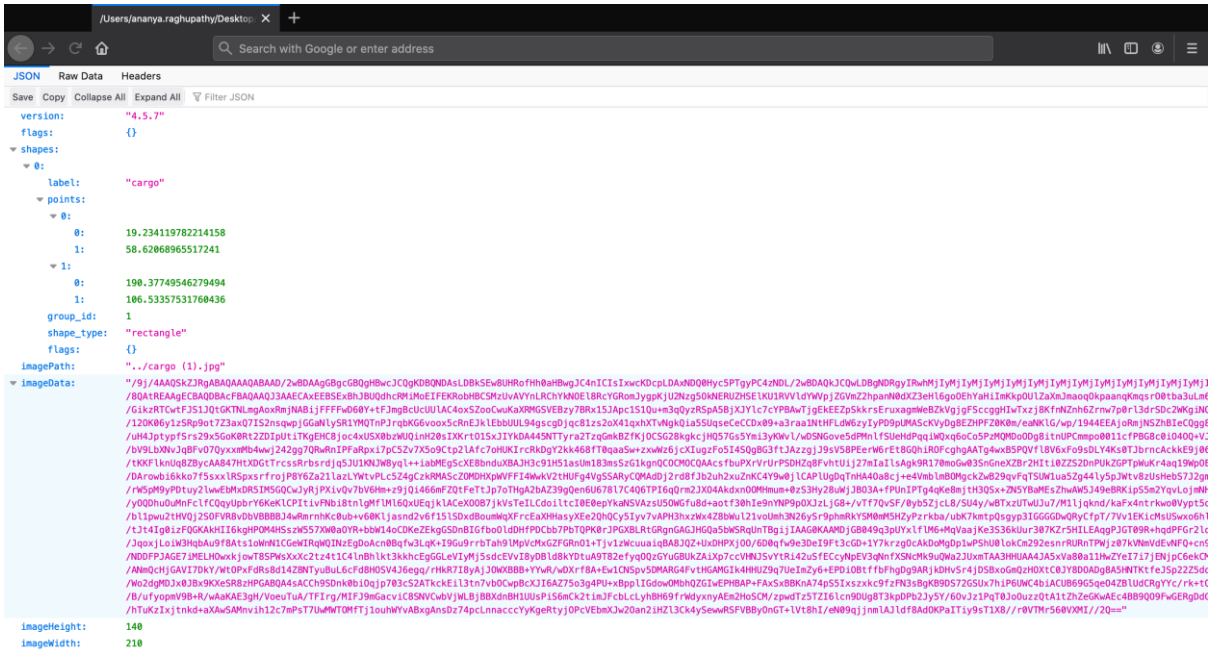
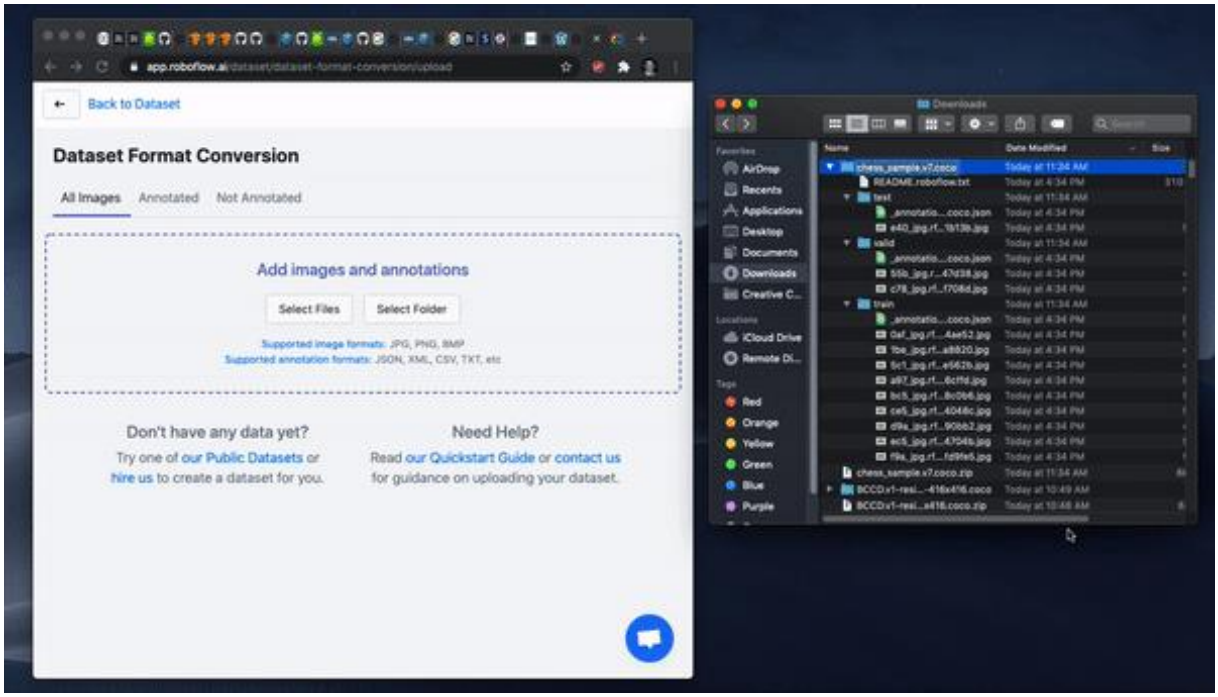


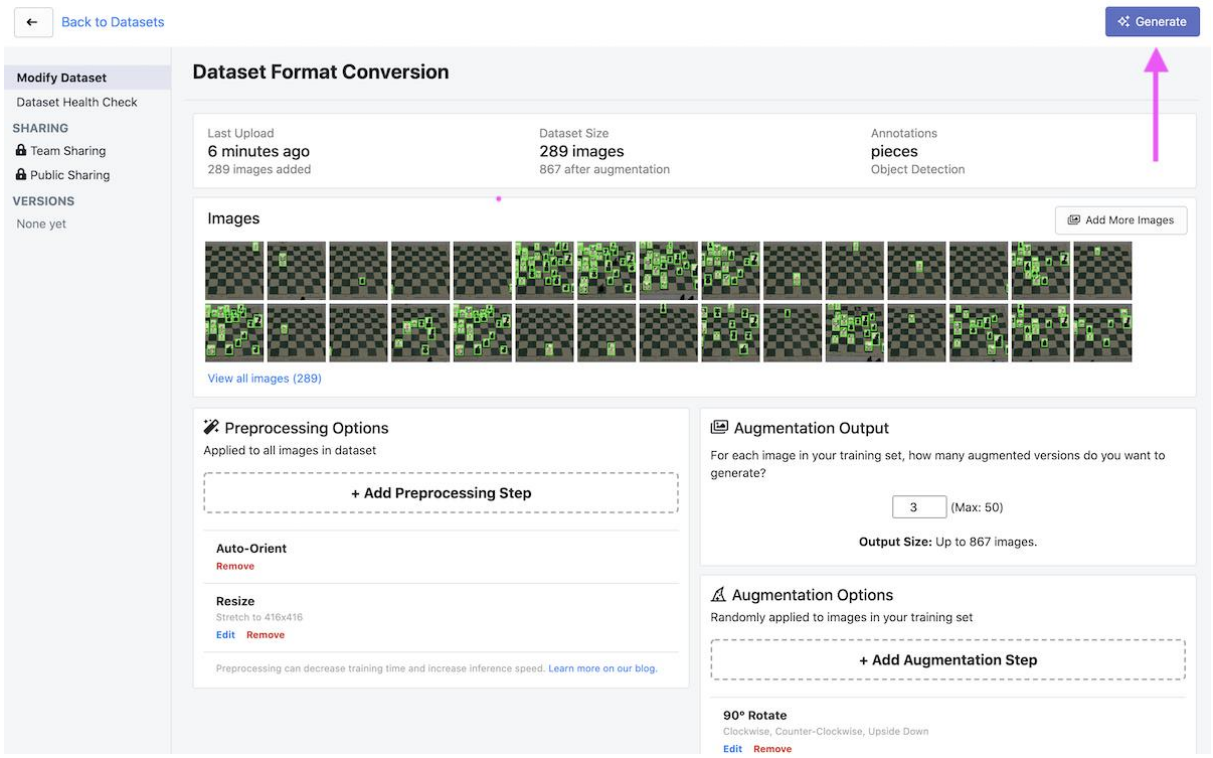
Fig. 36 – The output from the LabelMe Annotation Tool

Step 2**:** Converting the JSON files into YOLO .txt files

Since the LabelMe annotation tool saves the output of each image in JSON format, it is mandatory to convert them into a format that YOLO model can interpret. In YOLO labelling format, a .txt file with the same name is created for each image file in the same directory. Each .txt file contains the annotations for the corresponding image file, that is object class, object coordinates, height and width. Most YOLO versions can only read .txt files with a specific format as mentioned above.

As a result, to convert my JSON files to .txt files for each corresponding image, I used the RoboFlow platform. The process has been recorded in screenshots, which are presented in Figure 37.


Back to Dataset
Dataset Format Conversion
All Images
Annotated
Not Annotated
Add images and annotations
Select Files
Select Folder
Supported image formats: JPG, PNG, BMP
Supported annotation formats: JSON, XML, CSV, TXT, etc
Don't have any data yet?
Try one of our Public Datasets or hire us to create a dataset for you.
Need Help?
Read our Quickstart Guide or contact us for guidance on uploading your dataset.


Back to Datasets
Generate
Modify Dataset
Dataset Health Check
SHARING
Team Sharing
Public Sharing
VERSIONS
None yet
Dataset Format Conversion
Last Upload
6 minutes ago
289 images added
Dataset Size
289 images
867 after augmentation
Annotations
pieces
Object Detection
Images
Add More Images
View all images (289)
Preprocessing Options
Applied to all images in dataset
+ Add Preprocessing Step
Auto-Orient
Remove
Resize
Stretch to 416x416
Edit Remove
Preprocessing can decrease training time and increase inference speed. Learn more on our blog.
Augmentation Output
For each image in your training set, how many augmented versions do you want to generate?
3
(Max: 50)
Output Size: Up to 867 images.
Augmentation Options
Randomly applied to images in your training set
+ Add Augmentation Step
90° Rotate
Clockwise, Counter-Clockwise, Upside Down
Edit Remove

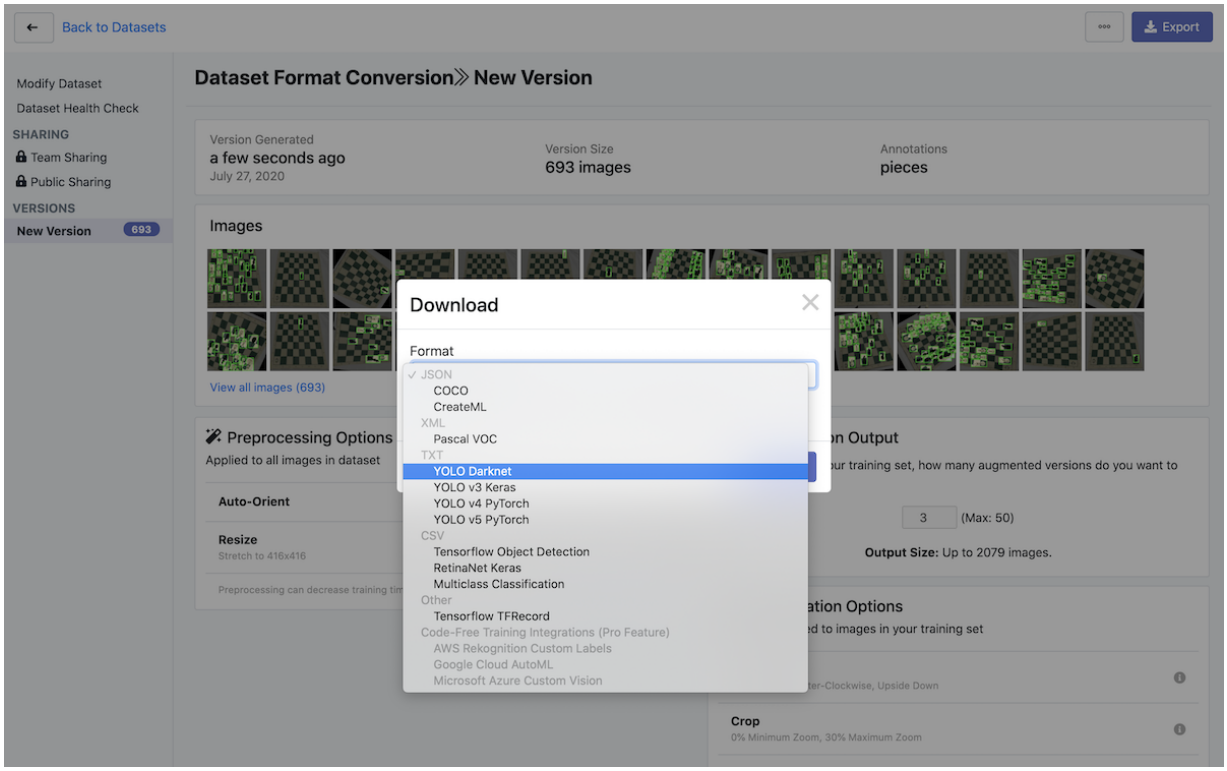


Fig. 37 – The RoboFlow Process

Step 3**:** Training the Model

As mentioned earlier, YOLO is an object detection algorithm touted to be one of the fastest. It was trained on the COCO dataset and achieved a mean Average Precision (mAP) of 33.0 at the speed of 51ms per image on Titan X GPU, which is commendable. The major highlight of the algorithm is that it divides the input image into several individual grids, and each grid predicts the objects inside it. This way, the whole image is processed at once, and the inference time is reduced.

The YOLO model used by me is trained on the Darknet Framework, which is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.

After the initial configuration and setup, I trained my YOLOv4 darknet model for over 2000 iterations, at the end of which I received the following results, as seen in the below section.

4.2.4 Results

At the end of my training after 2000 iterations, the results that I obtained have been presented in Figure 38.

```
 Last accuracy mAP@0.5 = 82.02 %, best = 82.02 %
 2000: 0.594563, 0.990275 avg loss, 0.001000 rate, 6.906656 seconds, 128000 images, 11.574178 hours left
Resizing to initial size: 416 x 416  try to allocate additional workspace_size = 111.05 MB
 CUDA allocate done!

 calculation mAP (mean average precision)...
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
180
 detections_count = 651, unique_truth_count = 275
class_id = 0, name = ship, ap = 99.64%           (TP = 52, FP = 2)
class_id = 1, name = cargo, ap = 98.39%          (TP = 61, FP = 0)
class_id = 2, name = carrier, ap = 91.99%        (TP = 100, FP = 18)
class_id = 3, name = cruise, ap = 39.35%         (TP = 30, FP = 37)

 for conf_thresh = 0.25, precision = 0.81, recall = 0.88, F1-score = 0.85
 for conf_thresh = 0.25, TP = 243, FP = 57, FN = 32, average IoU = 67.08 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.823406, or 82.34 %
Total Detection Time: 5 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.5) = 0.823406
New best mAP!
Saving weights to /mydrive/yolov4/training/yolov4-custom_best.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_2000.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_last.weights
```

Fig. 38 – Results using the YOLOv4 model

With a mean average precision (mAP) value at 82.02 percent, the results obtained were commendable.

<loss + accuracy graph vs epoch>

<results snapshots>

4.2.4 Key Takeaways

<need to look into what I can put in>

SCHEDULE