OBJECT DETECTION AND IMAGE CLASSIFICATION

A PROJECT REPORT

Submitted by

RAJNANDAN KUMAR (730920201038)

VICKY KUMAR SINGH (730920201052)

RAHUL KUMAR (730920201035)

RAJ LAXMI (730920201036)

In partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

EXCEL ENGINEERING COLLEGE , TAMILNADU-637303

(An Autonomous Institution)

APRIL 2023

BONAFIDE CERTIFICATE

Certified that this project report "**OBJECT DETECTION AND IMAGE CLASSIFICATION"** is the bonafide work of **"RAJNANDAN KUMAR (730920201038), VICKY KUMAR SINGH (730920201052), RAHUL KUMAR (730920201035), RAJ LAXMI (730920201036)"** who carried out the project work under my supervision.

SIGNATURE

Mrs. P. Jayaprabha

HEAD OF THE DEPARTMENT,

Artificial Intelligence and Data Science,

Excel Engineering College,
Tamilnadu – 637303.

SIGNATURE

Mr. Nandakumar,

SUPERVISOR,

Artificial Intelligence and Data Science,

Excel Engineering College,
Tamilnadu – 637303.

The Mini project report submitted for the viva voce held on …….

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

This mini project focuses on object detection using deep learning techniques. The goal is to train a Convolution Neural Network(CNN) to detect in images and draw bounding boxes around them. The project involves collecting and annotating a dataset of images, training a CNN using a transfer learning approach, and evaluating the model's performance on a test set. The project utilizes the TensorFlow framework and the pre-trained models available in the TensorFlow Object Detection API. The results show that the trained model is able to accurately detect objects in the test set with a high average precision. Overall, this project demonstrates the effectiveness of deep learning techniques for project detection tasks and provides a foundation for further exploration and development in this area.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

S. NO	ABBREVATION	EXPANSION
1	YOLO	You Only Look Once
2	VGG	Visual Geometry Group
3	R-CNN	Region Classification Neural Network
4	CNN	Convolution Neural Network
5	SSD	Single Shot Detector

CHAPTER 1

INTRODUCTION

A few years ago, the creation of the software and hardware image processing systems was mainly limited to the development of the user interface, which most of the programmers of each firm were engaged in. The situation has been significantly changed with the advent of the Windows operating system when the majority of the developers switched to solving the problems of image processing itself. However, this has not yet led to the cardinal progress in solving typical tasks of recognizing faces, car numbers, road signs, analysing remote and medical images, etc. Each of these "eternal" problems are solved by trial and error by the efforts of numerous groups of the engineers and scientists.

As modern technical solutions are turn out to be excessively expensive, the task of automating the creation of the software tools for solving intellectual problems is formulated and intensively solved abroad. In the field of image processing, the required tool kit should be supporting the analysis and recognition of images of previously unknown content and ensure the effective development of applications by ordinary programmers. Just as the Windows toolkit supports the creation of interfaces for solving various applied problems.

Object recognition is to describe a collection of related computer vision tasks that involve activities like identifying objects in digital photographs. Image classification involves activities such as predicting the class of one object in an image. Object localization is referring to identifying the location of one or more objects in an image and drawing an abounding box around their extent. Object detection does the work of combines these two tasks and localizes and classifies one or more objects in an image. When a user or practitioner refers to the term “object recognition”, they often mean

“object detection”, It may be challenging for beginners to distinguish between different related computer vision tasks.

So, we can distinguish between these three computer vision tasks with this example:

Image Classification: This is done by Predict the type or class of an object in an image.

Input: An image which consists of a single object, such as a photograph.

Output: A class label (e.g. one or more integers that are mapped to class labels).

Object Localization: This is done through, Locate the presence of objects in an image and indicate their location with a bounding box.

Input: An image which consists of one or more objects, such as a photograph.

Output: One or more bounding boxes (e.g. defined by a point, width, and height).

Object Detection: This is done through, Locate the presence of objects with a bounding box and types or classes of the located objects in an image.

Input: An image which consists of one or more objects, such as a photograph.

Output: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.

One of the further extensions to this breakdown of computer vision tasks is object segmentation, also called “object instance segmentation” or “semantic segmentation,” where instances of recognized objects are indicated by highlighting the specific pixels of the object instead of a coarse bounding box. From this breakdown, we can understand that object recognition refers to a suite of challenging computer vision tasks.

For example, image classification is simply straight forward, but the differences between object localization and object detection can be confusing, especially when all three tasks may be just as equally referred to as object recognition.

Humans can detect and identify objects present in an image. The human visual system is fast and accurate and can also perform complex tasks like identifying multiple objects and detect obstacles with little conscious thought. The availability of large sets of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy. We need to understand terms such as object detection, object localization, loss function for object detection and localization, and finally explore an object detection algorithm known as “You only look once” (YOLO).

Image classification also involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is always more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all these problems are referred to as object recognition.

Object recognition refers to a collection of related tasks for identifying objects in digital photographs. Region-based Convolutional Neural Networks, or R-CNNs, is a family of techniques for addressing object localization and recognition tasks, designed for model performance. You Only Look Once, or YOLO is known as the second family of techniques for object recognition designed for speed and real-time use.

1.1 SCOPE OF THE PROJECT

The main scope of this project is to develop an efficient framework that allow us to identify and locate objects in an image or video .

1.2 OBJECTIVE OF THE PROJECT

Develop of application that detects an object and it can be used for vehicles counting, when the object is vehicle such as a bicycle or car, it can count how many vehicles have passed from a particular area or road and it can recognize human activity too.

1.3 REPORT SUMMARY

The project report is organized as follows: the chapter 2 narrates the related work done in this Object Detection domain, chapter 3 explains the existing system andmethodologies, chapter 4 depicts the architecture and design of the proposed methodology, chapter 5 discuss the system implementation requirements, chapter 6 describes the simulation and discussion for the Object detection model, the chapter 7 details the conclusion and future work of this Object detection model, and the last section provides the references and appendix.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

In various fields, there is a necessity to detect the target object and also track them effectively while handling occlusions and other included complexities.

2.2 Object Detection

Object detection is an important task, yet challenging vision task. It is a critical part of many applications such as image search, image auto-annotation and scene understanding, object tracking. Moving object tracking of video image sequences was one of the most important subjects in computer vision.

It had already been applied in many computer visions fields, such as smart video surveillance, artificial intelligence, military guidance, safety detection and robot navigation, medical and biological application. In recent years, a number of successful single-object tracking system appeared, but in the presence of several objects, object detection becomes difficult and when objects are fully or partially occluded, they are obtruded from the human vision which further increases the problem of detection.

Decreasing illumination and acquisition angle. The proposed MLP based object tracking system is made robust by an optimum selection of unique features and also by implementing the Ad boost strong classification method.

2.3 Background

The aim of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image. Generally, only a small number of instances

of the object are present in the image, but there is a very large number of possible locations and scales at which they can occur and that need to somehow be explored. Each detection of the image is reported with some form of pose information.

This is as simple as the location of the object, a location and scale, or the extent of the object defined in terms of a bounding box. In some other situations, the pose information is more detailed and contains the parameters of a linear or non-linear transformation. For example, for face detection in a face detector may compute the locations of the eyes, nose and mouth, in addition to the bounding box of the face.

An example of a bicycle detection in an image that specifies the locations of certain parts is shown in Figure 1.1. The pose can also be defined by a three-dimensional transformation specifying the location of the object relative to the camera. Object detection systems always construct a model for an object class from a set of training examples.

In the case of a fixed rigid object in an image, only one example may be needed, but more generally multiple training examples are necessary to capture certain aspects of class variability

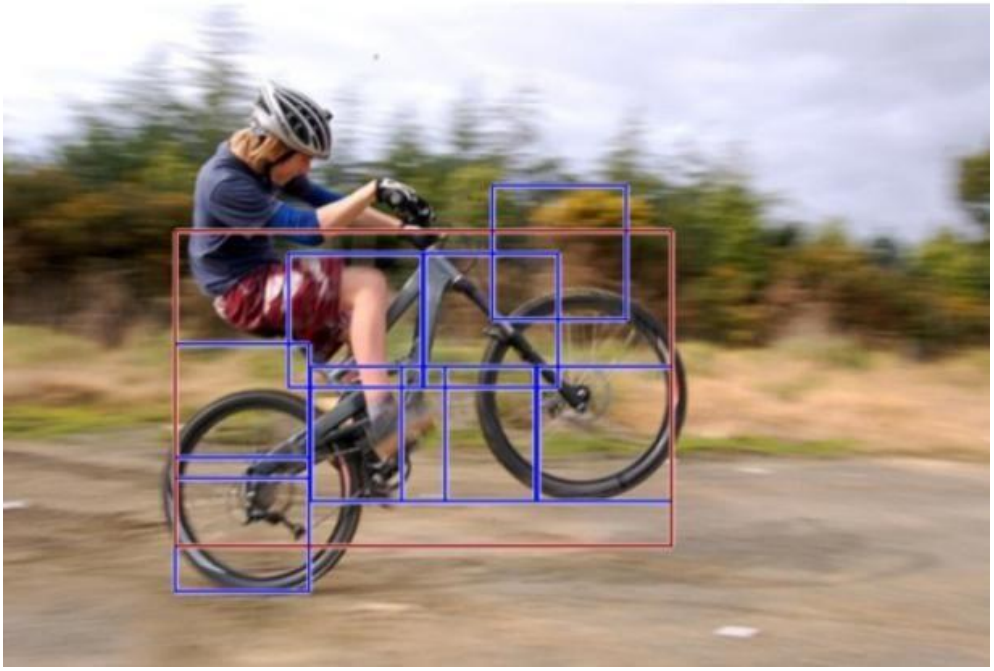
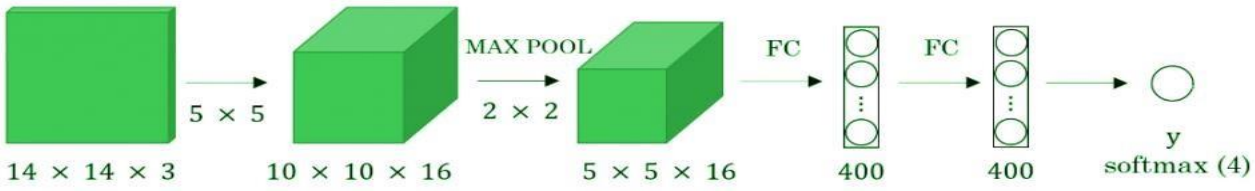


Figure 2.1: Background

Convolutional implementation of the sliding windows Before we discuss the implementation of the sliding window using convents, let us analyze how we can convert the fully connected layers of the network into convolutional layers. Fig. 2.1



show.

Figure 2.2

A fully connected layer can be converted to a convolutional layer with help of an ID convolutional layer. The width and height of layer is equal to one and the number of filters is equal to the shape of the fully connected layer. An example of this is show in Fig. 2.2.

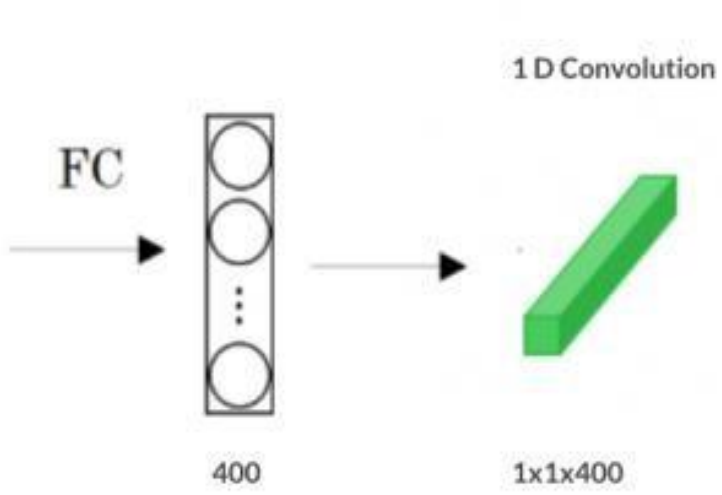


Figure 2.3

We can apply the concept of conversion of a fully connected layer a convolutional layer to the model by replacing the fully connected layer with a 1-D convolutional layer. The number of filters of the 1D convolution layer is equal to the shape of the fully connected layer. This representation is show in Fig.2.3. Also, the output SoftMax layer is also a convolutional layer of shape (1, 1, 4), where 4 is the number of classes to predict.

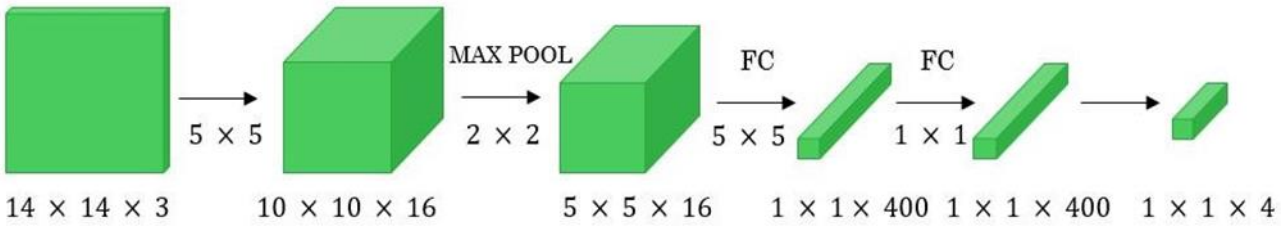


Figure 2.4

Now, let's extend the above approach to implement a convolutional version of the sliding window. First, let us consider the Conv Net that we have trained to be in the following representation (no fully connected layers).

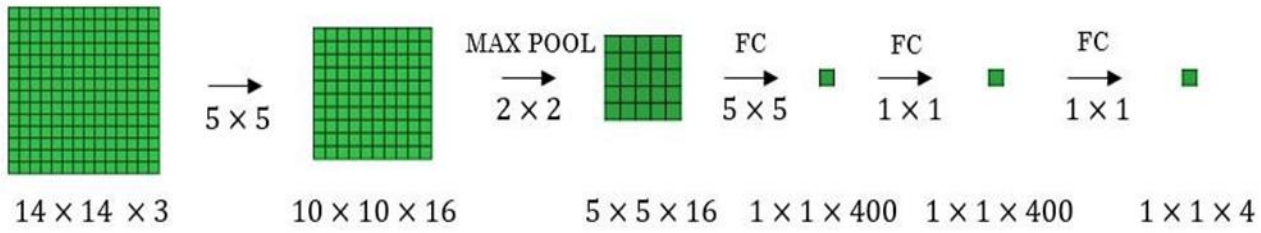


Figure2.5

Let's assume the size of the input image to be 16 × 16 × 3. If we are using the sliding window approach, then we would have passed this image to the above ConvNet four times, where each time the sliding window crops the part of the input image matrix of size 14 × 14 × 3 and pass it through the ConnNet. But instead of this, we feed the full image (with shape 16 × 16 × 3) directly into the trained ConvNet (see Fig. 6). This results will give an output matrix of shape 2 × 2 × 4. Each cell in the output matrix represents the result of the possible crop and the classified value of the cropped image. For example, the left cell of the output matrix(the green one) in Fig. 6 represents the result of the first sliding window. The other cells in the matrix represent the results of the remaining sliding window operations.

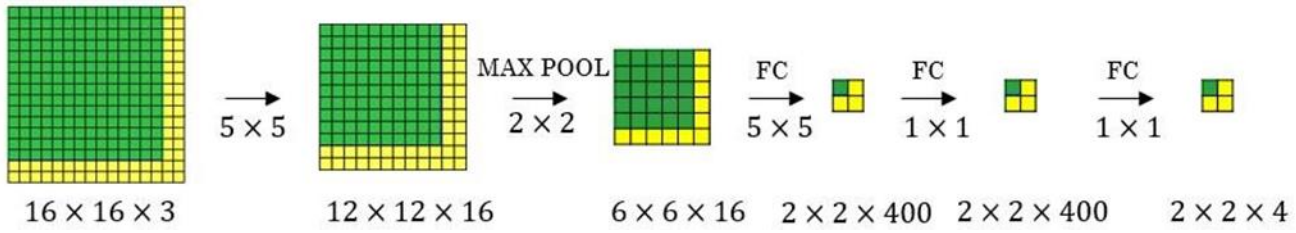


Figure 2.6

The stride of the sliding window is decided by the number of filters used in the Max Pool layer. In the example above, the Max Pool layer has two filters, and for the result, the sliding window moves with a stride of two resulting in four possible outputs to the given input.

The main advantage of using this technique is that the sliding window runs and computes all values simultaneously. Consequently, this technique is really fast. The weakness of this technique is that the position of the bounding boxes is not very accurate.

A better algorithm that tackles the issue of predicting accurate bounding boxes while using the convolutional sliding window technique is the YOLO algorithm. YOLO stands for you only look once which was developed in 2015 by Joseph Redmon, Santosh Davila, Ross Girshes, and Ali Farhadi.

It is popular because it achieves high accuracy while running in real-time. This algorithm requires only one forward propagation pass through the network to make the predictions.

This algorithm divides the image into grids and then runs the image classification and localization algorithm (discussed under object localization) on each of the grid cells. For example, we can give an input image of size 256 × 256. We place a 3 × 3 grid on the image (see Fig. 7).

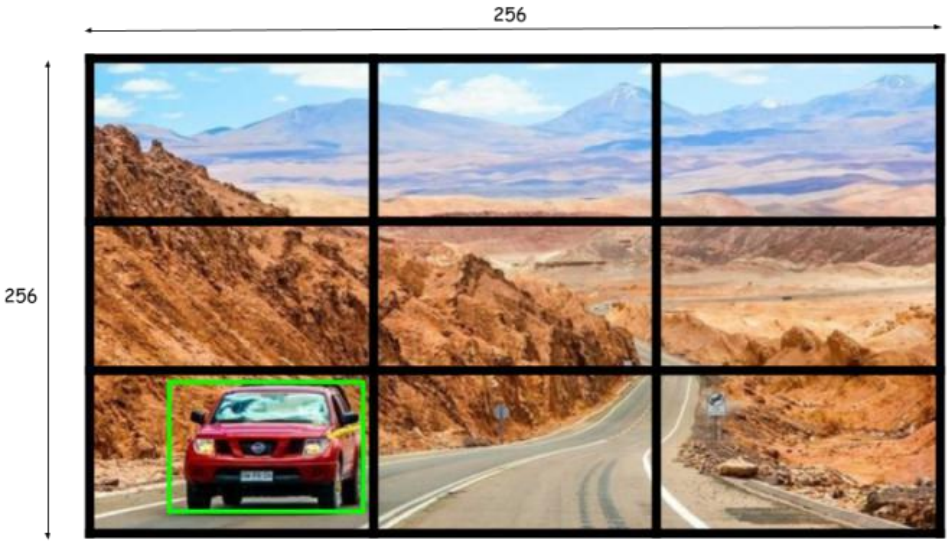


Figure 2.7: Object localization

Next, we shall apply the image classification and localization algorithm on each grid cell. In the image each grid cell, the target variable is defined as Yi,

j=[pcbxbybhbwc1c2c3c4]T(6).

Do everything once with the convolution sliding window. Since the shape of the target variable for each grid cell in the image is 1 × 9 and there are 9 (3 × 3) grid cells, the final output of the model will be:

$$Final\ Output = \underbrace{3 \times 3}_{\text{Number of grid cells}} \times \underbrace{9}_{\text{Output label for each grid cell}}$$

The advantages of the YOLO algorithm is that it is very fast and predicts much more accurate bounding boxes. Also, in practice to get the more accurate predictions, we use a much finer grid, say 19 × 19, in which case the target output is of the shape 19 × 19 × 9.

2.4 Templet matching

Template Matching is the technique of finding small parts of an image which match a template image. It slides the template from the top left to the bottom right of the image and compares for the best match with the template. The template dimension should be equal to the reference image or smaller than the reference image.

It recognizes the segment with the highest correlation as the target. Given an image S and an image T, where the dimension of S was both larger than T, output whether S contains a subset image I where I and T are suitably similar in pattern and if such I exists, output the location of I in S as in Hager and Bilheimer (1998).

Schweitzer et al (2011), derived an algorithm which used both upper and lowers bound to detect ‘k’ best matches. Euclidean distance and Walsh transform kernels are used to calculate match measure. The positive things included the usage of priority

queue improved quality of decision as to which bound-improved and when good matches exist inherent cost was dominant and its improved performance.

But there were constraints like the absence of good matches that lead to queue cost and the arithmetic operation cost was higher. The proposed methods dint use queue thereby avoiding the queue cost rather used template matching.

Visual tracking methods can be roughly categorized in two ways namely, the feature-based and region-based method as proposed by Ken Ito and Shigeyuki Sarane (2001).

The feature-based approach estimates the 3D pose of a target object to fit the image features the edges, given a 3D geometrical model of an object. This method requires much computational cost.

Region-based can be classified into two categories namely, parametric method and view-based method.

The parametric method assumes a parametric model of the images in the target image and calculates optimal fitting of the model to pixel data in a region.

CHAPTER 3

EXISTING SYSTEM

3.1 ResNet

To train the network model in a more effective manner, we herein adopt the same strategy as that used for DSSD (the performance of the residual network is better than that of the VGG network). The goal is to improve accuracy.

However, the first implemented for the modification was the replacement of the VGG network which is used in the original SSD with ResNet. We will also add a series of convolution feature layers at the end of the underlying network.

These feature layers will gradually be reduced in size that allowed prediction of the detection results on multiple scales. When the input size is given as 300 and 320, although the ResNet–101 layer is deeper than the VGG–16 layer, it is experimentally known that it replaces the SSD's underlying convolution network with a residual network, and it does not improve its accuracy but rather decreases it.

3.2 R-CNN

To circumvent the problem of selecting a huge number of regions, Ross Girshick et al. proposed a method where we use the selective search for extract just 2000 regions from the image and he called them region proposals.

Therefore, instead of trying to classify the huge number of regions, you can just work with 2000 regions. These 2000 region proposals are generated by using the selective search algorithm which is written below.

Selective Search:

1. Generate the initial sub-segmentation, we generate many candidate regions.
2. Use the greedy algorithm to recursively combine similar regions into larger ones.
3. Use generated regions to produce the final candidate region proposals.

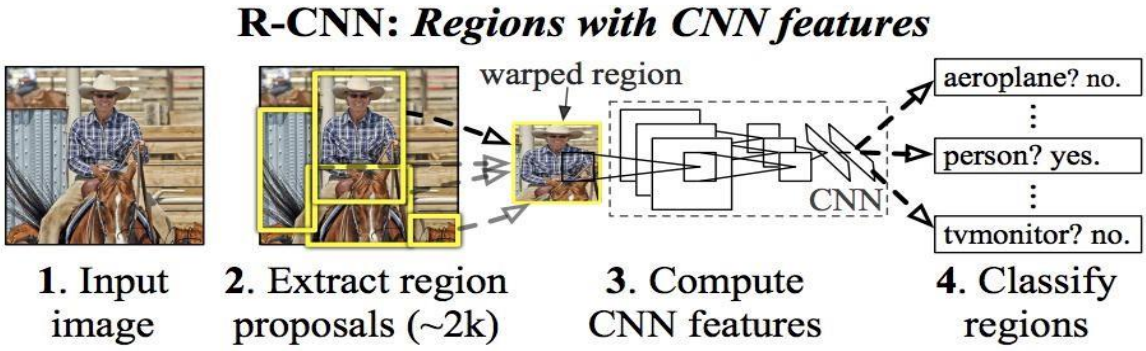


Figure 3.1 : R-CNN

These 2000 candidate regions which are proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN plays a role of feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM for the classify the presence of the object within that candidate region proposal.

In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values for increasing the precision of the bounding box. For example, given the region proposal, the algorithm might have predicted the presence of a person but the face of that person within that region proposal could have been cut in half. Therefore, the offset values which is given help in adjusting the bounding box of the region proposal.

These 2000 candidate regions which are proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN plays a role of feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM for the classify the presence of the object within that candidate region proposal. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values for increasing the precision of the bounding box. For example, given the region proposal, the algorithm might have predicted the presence of a person but the face of that person within that region proposal could have been cut in half. Therefore, the offset values which is given help in adjusting the bounding box of the region proposal.

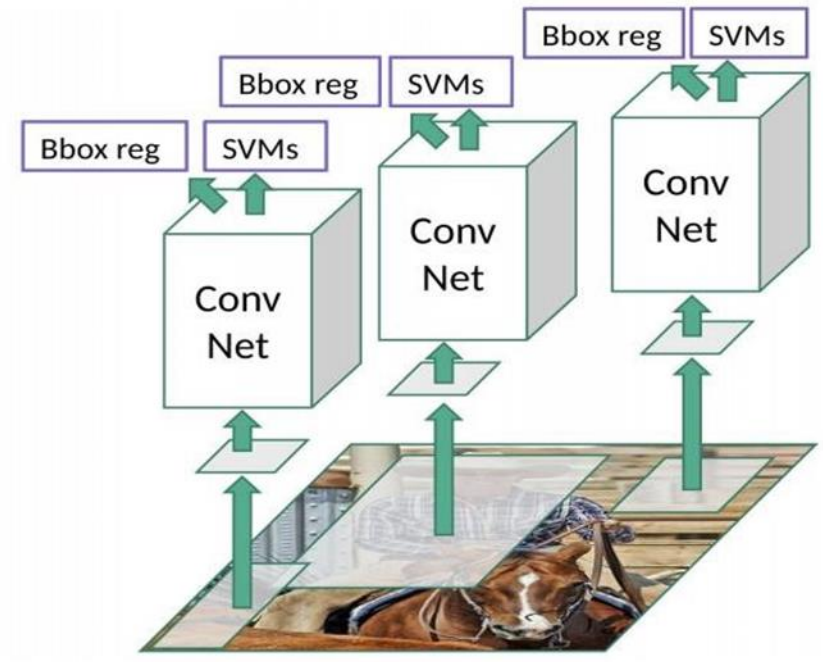


Figure 3.2 : R-CNN

3.3 Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.

- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

3.4 FAST R-CNN

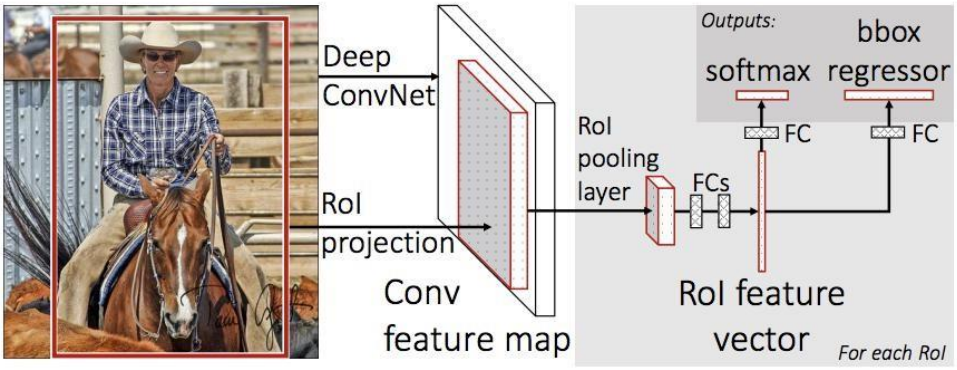


Figure 3.3: FAST R-CNN

The same author of the previous paper(R-CNN) solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.

From the convolutional feature map, we can identify the region of the proposals and warp them into the squares and by using an Roil pooling layer we reshape them into the fixed size so that it can be fed into a fully connected layer. From the Roil feature vector, we can use a SoftMax layer to predict the class of the proposed region and also the offset values for the bounding box.

The reason "Fast R-CNN" is faster than R-CNN is because you don't have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is always done only once per image and a feature map is generated from it.

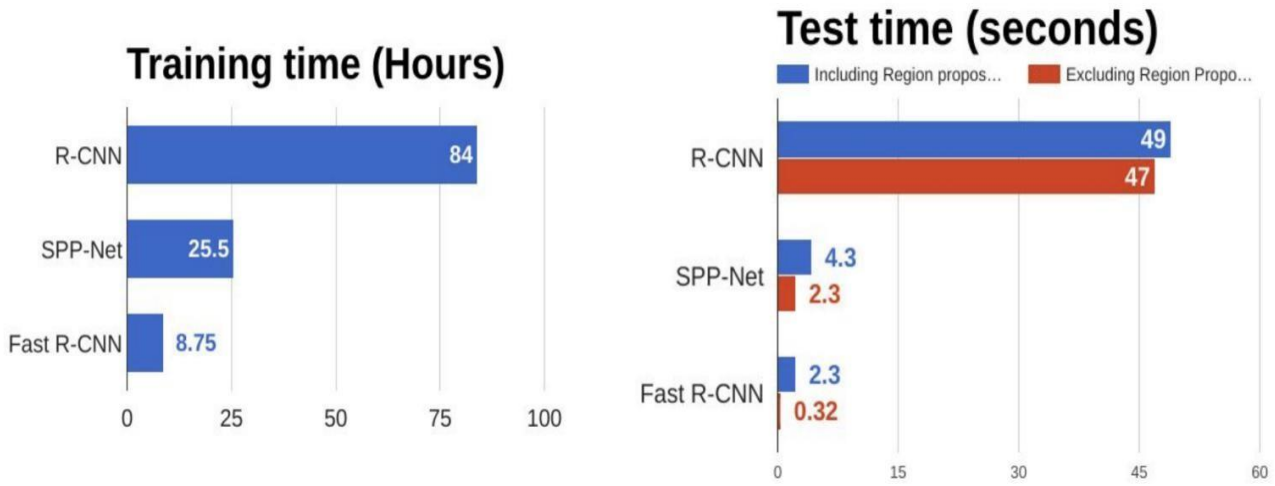


Figure 3.4: Comparison of object detection algorithm

From the above the graphs, you can infer that Fast R-CNN is significantly faster in training and testing sessions over R-CNN. When you look at the performance of Fast R-CNN during testing time, including region proposals slows down the algorithm significantly when compared to not using region proposals.

Therefore, the region which is proposals become bottlenecks in Fast R-CNN algorithm affecting its performance.

3.5 FASTER R-CNN

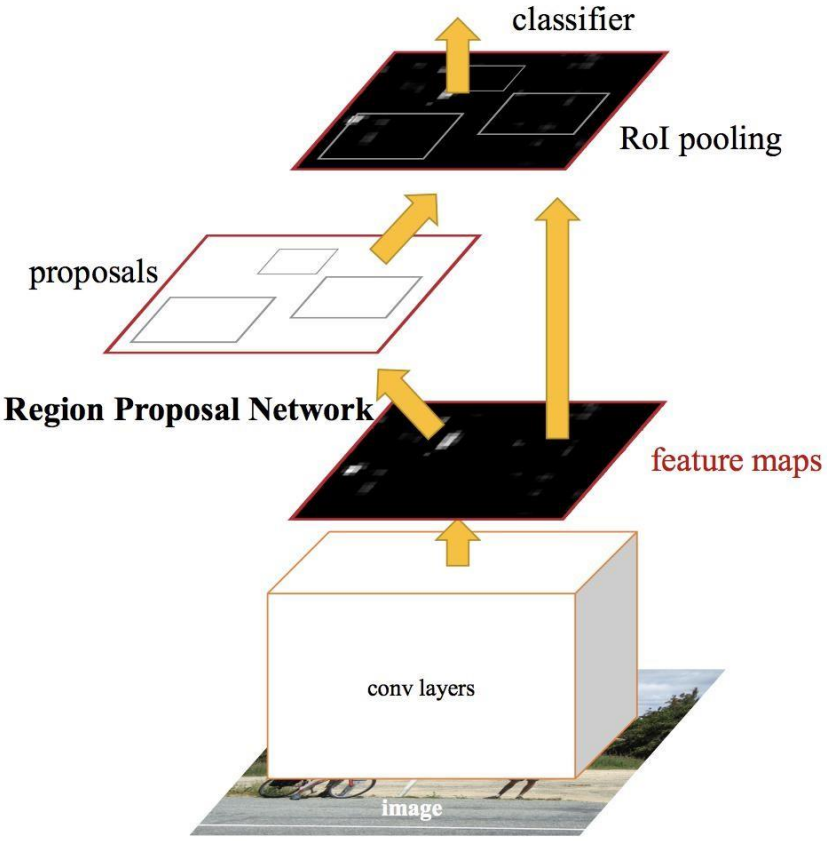


Figure 3.5: FASTER R-CNN

Both of the above algorithms(R-CNN & Fast R-CNN) uses selective search to find out the region proposals. Selective search is the slow and time-consuming process which affect the performance of the network.

Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using the selective search algorithm for the feature map to identify the region proposals, a separate network is used to predict the region proposals.

The predicted the region which is proposals are then reshaped using an RoI pooling layer which is used to classify the image within the proposed region and predict the offset values for the bounding boxes.

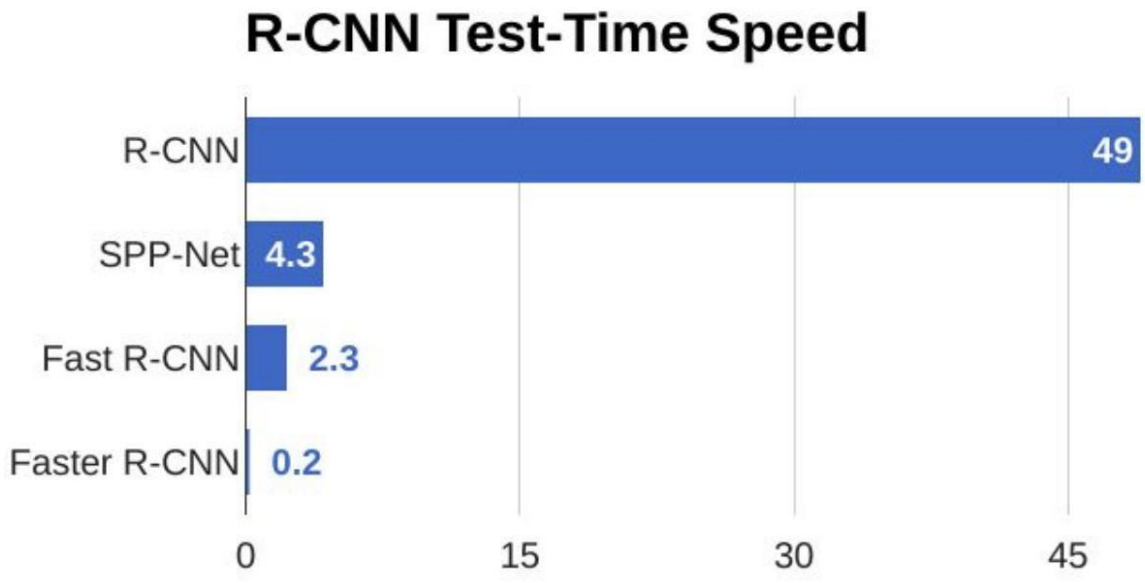


Figure 3.6: Comparison of test-time speed of object detection algorithms.

From the above graph, you can see that Faster R-CNN is much faster than its predecessors. Therefore, it can even be used for real-time object detection.

3.6 YOLO — You Only Look Once

All the previous object detection algorithms have used regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which has high probabilities of containing the object.

YOLO or You Only Look Once is an object detection algorithm much is different from the region-based algorithms which seen above. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.

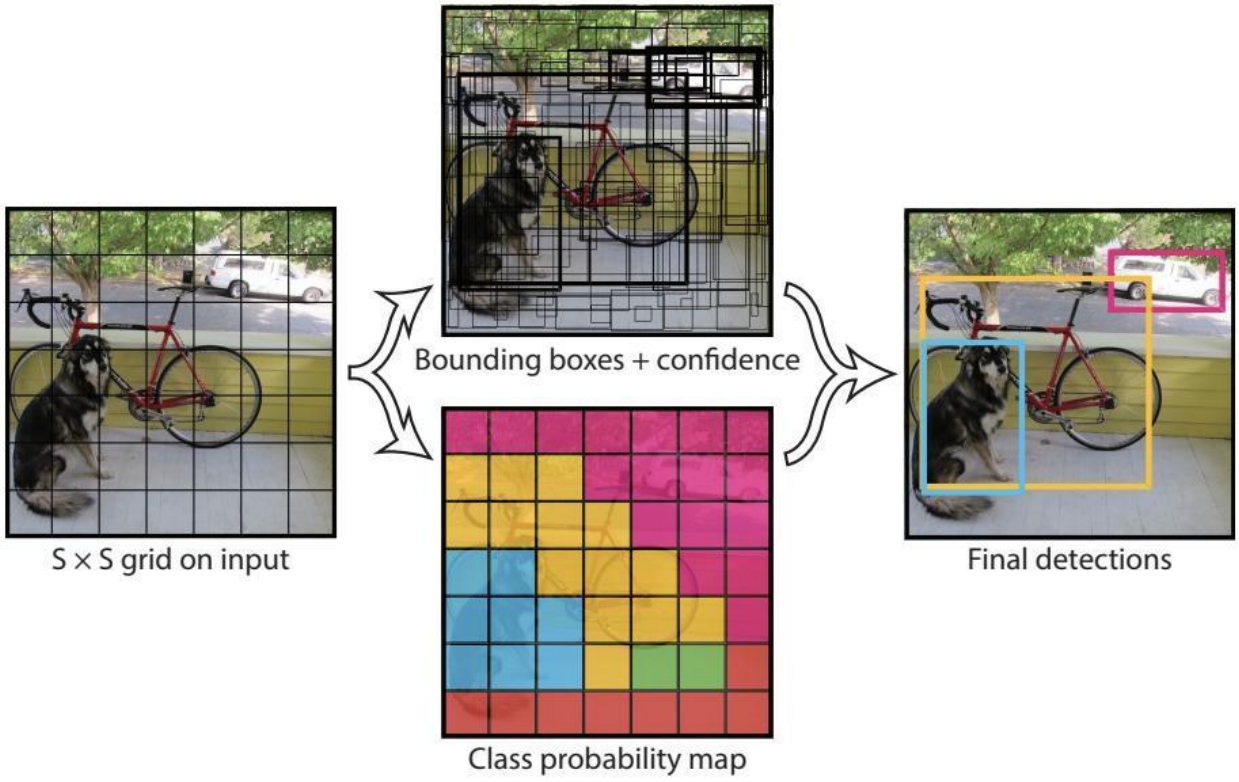


Figure 3.7: You Only Look Once

YOLO works by taking an image and split it into an S x S grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network gives an output a class probability and offset values for the bounding box. The bounding boxes have the class probability above a threshold value is selected and used to locate the

object within the image.

YOLO is orders of magnitude faster(45 frames per second) than any other object detection algorithms. The limitation of YOLO algorithm is that it struggles with the small objects within the image, for example, it might have difficulties in identifying a flock of birds. This is due to the spatial constraints of the algorithm.

3.7 SDD:

The SSD object detection composes of 2 parts:

1. Extract feature maps, and

Apply convolution filters to detect objects.

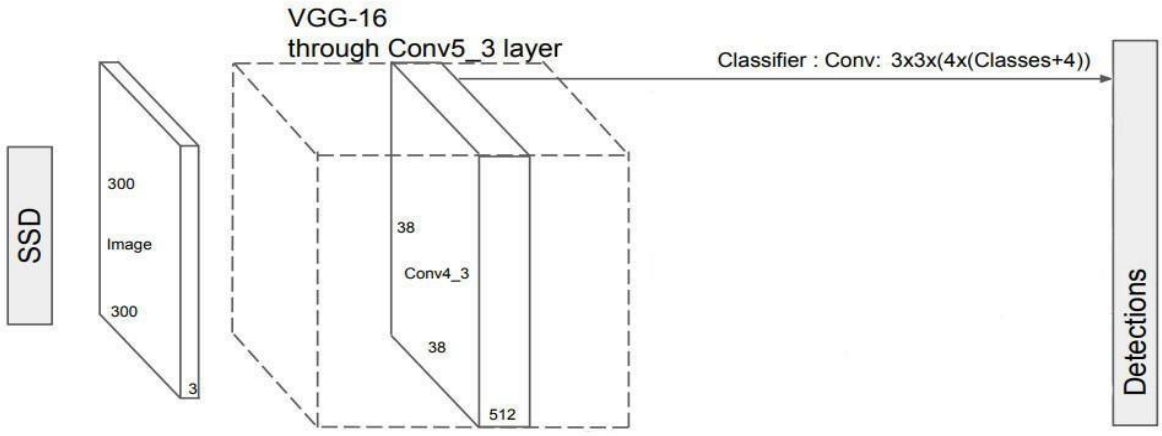


Figure 3.8 : SSD

SSD uses VGG16 to extract feature maps. Then it detects objects using the Conv4_3 layer. For illustration, we draw the Conv4_3 to be 8 × 8 spatially (it should be 38 × 38). For each cell in the image(also called location), it makes 4 object predictions.

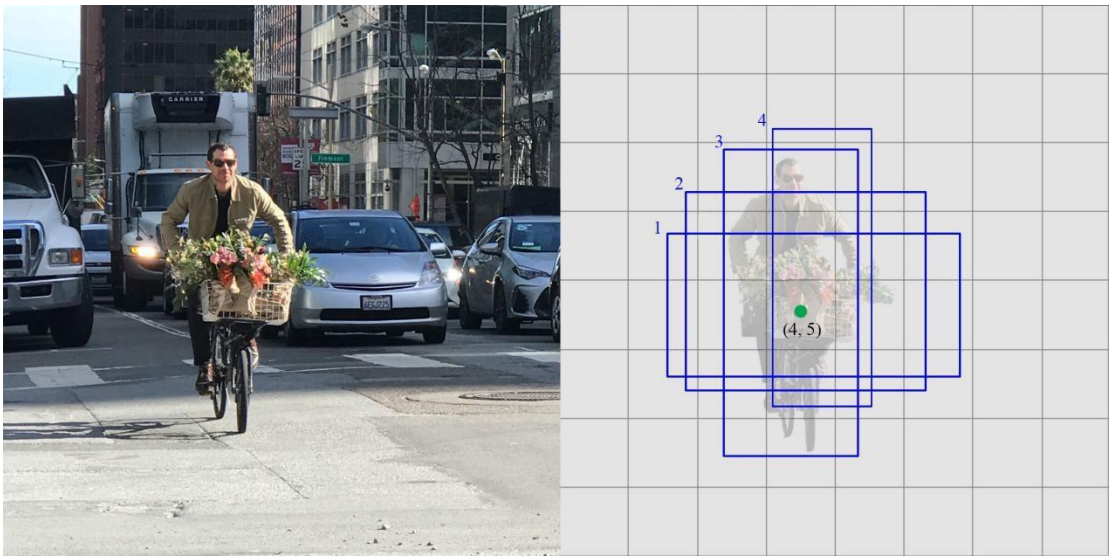


Figure 3.9

Each prediction composes of a boundary box and 21 scores for each class (one extra class for no object), and we pick the highest score as the class for the bounded object. Conv4_3 makes total of 38 × 38 × 4 predictions: four predictions per cell regardless of the depth of feature maps. An expected, many predictions contain no object. SSD reserves a class “0” to indicate

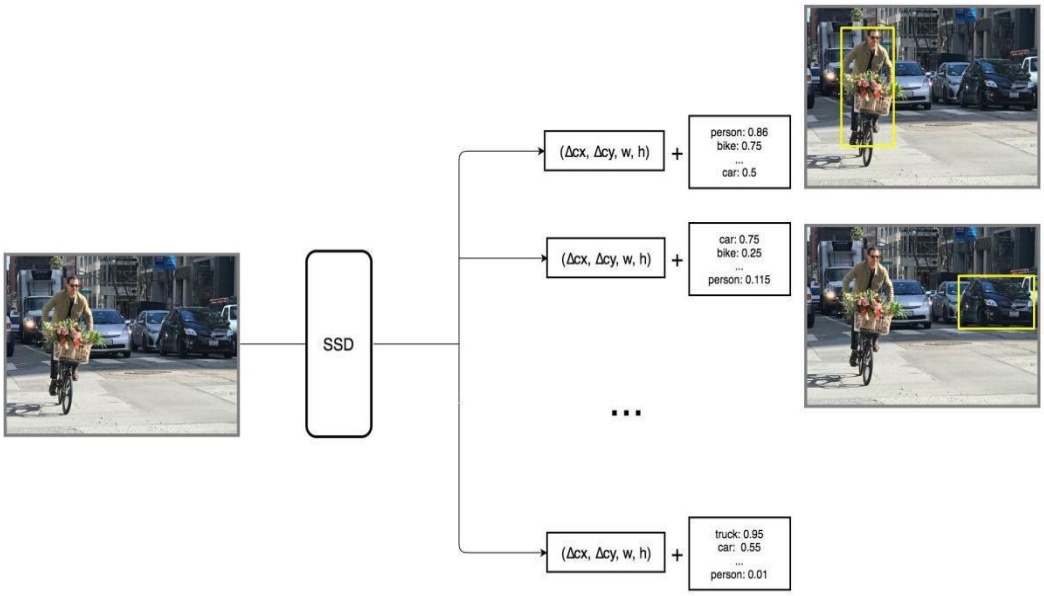


Figure 3.10

SSD does not use the delegated region proposal network. Instead, it resolves to a very simple method. It computes both the location and class scores using small convolution filters. After extraction the feature maps, SSD applies 3 × 3 convolution filters for each cell to make predictions. (These filters compute the results just like the regular CNN filters.) Each filter gives outputs as 25 channels: 21 scores for each class plus one boundary box.

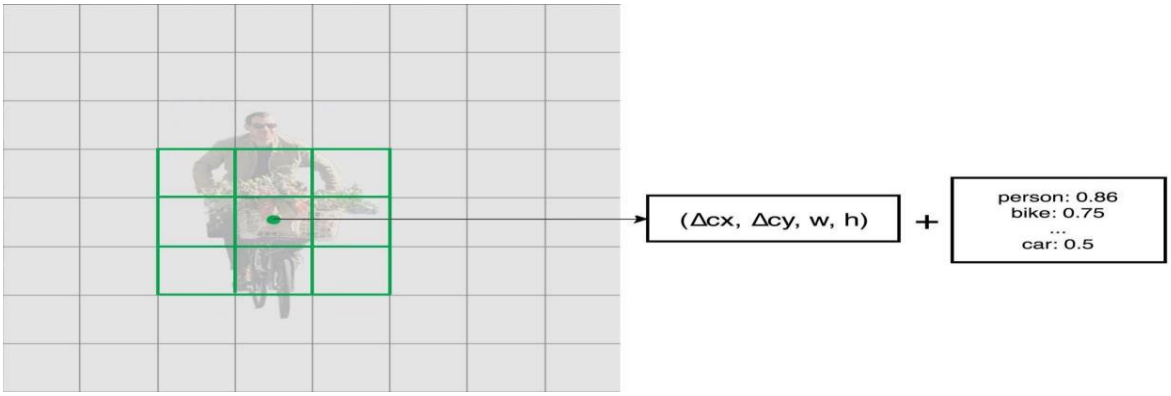


Figure 3.11

Beginning, we describe the SSD detects objects from a single layer. Actually, it uses multiple layers (multi-scale feature maps) for the detecting objects independently. As CNN reduces the spatial dimension gradually, the resolution of the feature maps also decreases. SSD uses lower resolution layers for the detect larger-scale objects. For example, the 4× 4 feature maps are used for the larger-scale object.

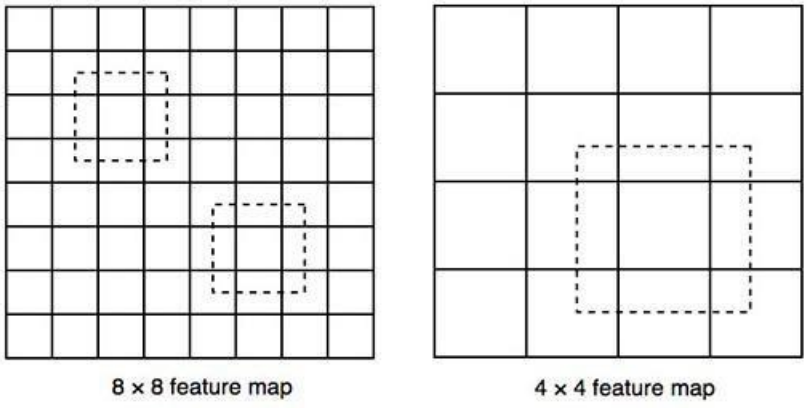


Figure 3.12

SSD adds 6 more auxiliary convolution layers to image after VGG16. Five of these layers will be added for object detection. In which three of those layers, we make 6 predictions instead of 4. In total, SSD makes 8732 predictions using 6 convolution layers.

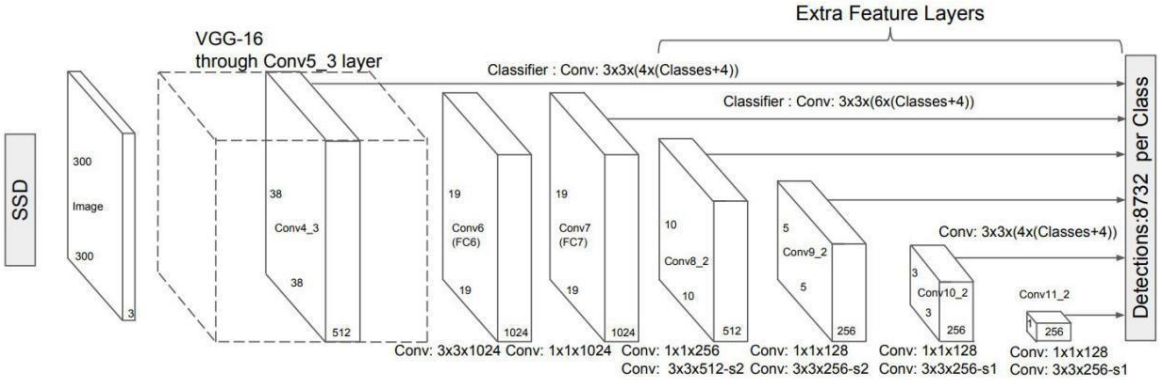


Figure 3.13

Multi-scale feature maps enhance accuracy. The accuracy with different number of feature map layers is used for object detection.

Prediction source layers from:						mAP use boundary boxes?		# Boxes
38 × 38	19 × 19	10 × 10	5 × 5	3 × 3	1 × 1	Yes	No	
✔	✔	✔	✔	✔	✔	74.3	63.4	8732
✔	✔	✔				70.7	69.2	9864
	✔					62.4	64.0	8664

Table 1

3.8 MANet:

Target detection is fundamental challenging problem for long time and has been a hotspot in the area of computer vision for many years. The purpose and objective of target detection is, to determine if any instances of a specified category of objects exist in an image. If there is an object to be detected in a specific image, target detection returns the spatial positions and the spatial extent of the instances of the objects (based on the use a bounding box, for example).

As one of cornerstones of image understanding and computer vision, target and object detection forms the basis for more complex and higher-level visual tasks, such as object tracking, image capture, instance segmentation, and others. Target detection is also widely used in areas such as artificial intelligence and information technology, including machine vision, automatic driving vehicles, and human–computer interaction. In recent times, the method automatic learning of represented features from data based on deep learning has effectively improved performance of target detection. Neural networks are foundation of deep learning.

Therefore, design of better neural networks has become a key issue toward improvement of target detection algorithms and performance. Recently developed object detectors that has been based on convolutional neural networks (CNN).

CHAPTER 4

PROPOSED WORK

4.1 Introduction

Object detection is a fundamental task in computer vision and has numerous practical applications, including surveillance, autonomous driving, robotics, and image retrieval. The ability to accurately and efficiently detect objects in images and videos has become increasingly important with the exponential growth of visual data in recent years. Despite significant progress in object detection techniques, there are still many challenges and limitations that need to be addressed.

The proposed research aims to develop an object detection system using deep learning techniques that can accurately and efficiently detect objects in complex scenes. The system will be designed to overcome the limitations of existing object detection algorithms and provide superior performance in terms of accuracy and speed.

The main research question for this proposed research is: How can deep learning techniques be leveraged to develop an object detection system that can accurately and efficiently detect objects in complex scenes? To answer this question, the proposed research will investigate various deep learning algorithms, including Convolutional Neural Networks (CNNs) and their variants, and evaluate their performance on different datasets.

The objectives of this proposed research are:

1. To review and analyze the latest research in the field of object detection and deep learning.
2. To develop a deep learning-based object detection system that can accurately and

efficiently detect objects in complex scenes.

3. To evaluate the performance of the proposed object detection system on benchmark datasets and compare it with existing state-of-the-art algorithms.

The significance and potential impact of this proposed research include the development of a highly accurate and efficient object detection system that can be applied to various practical applications. The proposed research will contribute to the advancement of the field of object detection and deep learning and have significant implications for computer vision and related fields.

4.2 Object Detection Framework

An object detection framework is a software architecture that provides a set of tools and libraries for developing object detection systems using machine learning or deep learning techniques. A framework typically includes pre-trained models, datasets, and evaluation metrics, as well as APIs for training and inference.

Here are some popular object detection frameworks:

1. TensorFlow Object Detection API: This is an open-source framework developed by Google that provides a set of pre-trained models, including Faster R-CNN, SSD, and YOLO, and tools for training and evaluation.
2. PyTorch Object Detection: This is a framework built on top of PyTorch that provides a collection of pre-trained models, such as Faster R-CNN, Mask R-CNN, and RetinaNet, and tools for training and evaluation.
3. Detectron2: This is a framework developed by Facebook AI Research (FAIR) that provides a set of pre-trained models, such as Mask R-CNN, Faster R-CNN, and Retina Net, and tools for training and evaluation.
4. OpenCV: This is an open-source computer vision library that provides tools and

algorithms for image and video processing, including object detection.

5. Caffe: This is a deep learning framework developed by Berkeley AI Research (BAIR) that provides tools for training and inference, and includes pre-trained models for object detection, such as R-CNN and Fast R-CNN.

Each framework has its strengths and weaknesses, depending on the specific requirements of the application. It is important to carefully evaluate the performance and scalability of the frameworks before selecting one for a particular project.

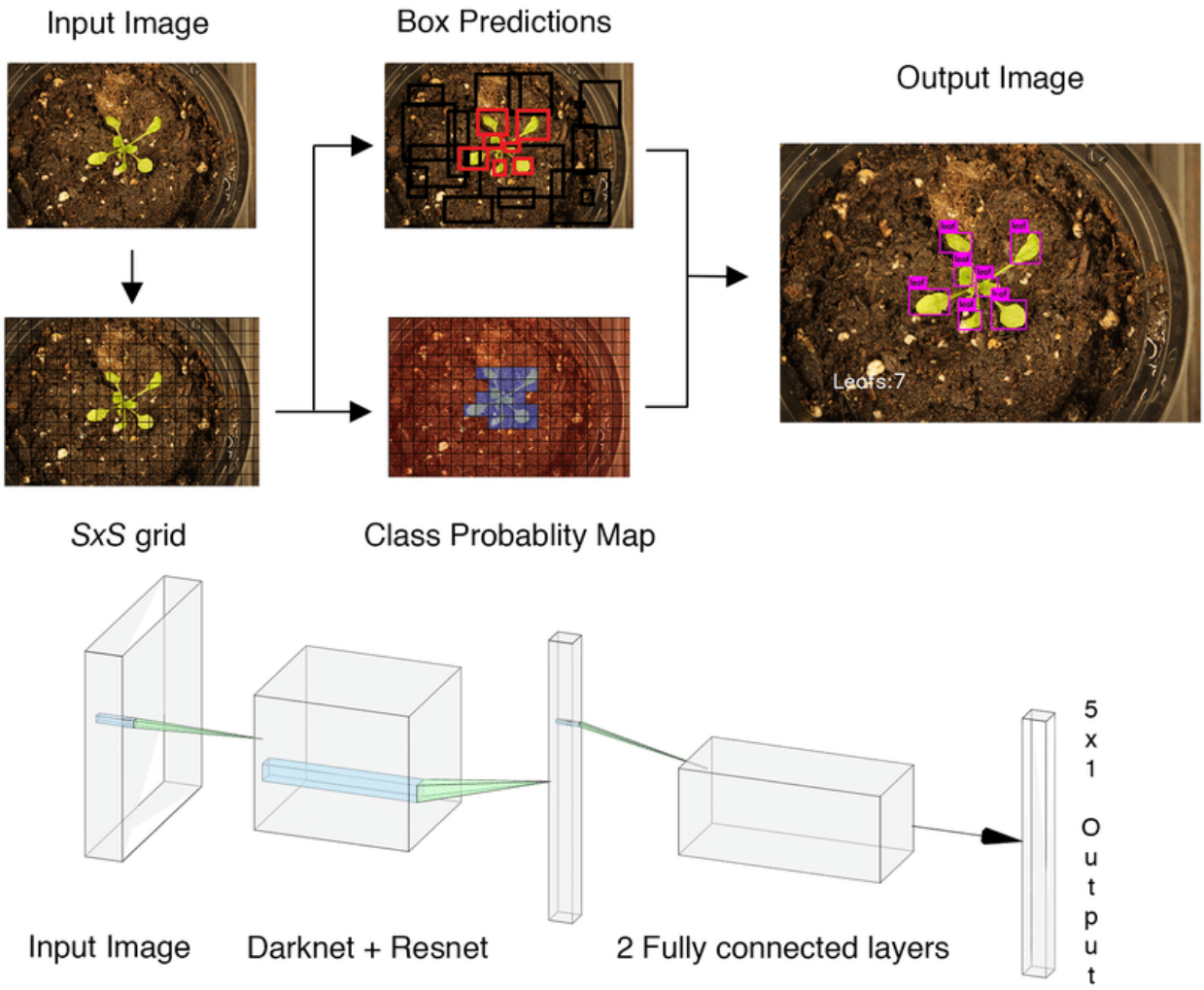


Figure 4.1 : YOLO-VGG16 Algorithm

4.3 YOLO Architecture

YOLO (You Only Look Once) is a single stage object detector, they take images and detect objects within the image instead of classifying the image according to a class. To be more specific, YOLO uses a single neural network pretrained on ImageNet to predict bounding boxes, which define where the object is in the image, and class probabilities. YOLO is the improved version in detection that works like the original network, but it runs faster and detects small objects better due to a new and improved network. During pretraining, the network was pretrained on 224 x 224 images, but during detection, YOLO works on images that are double the resolution. YOLO is a best in class ongoing item recognition framework, which beats Faster R-CNN, RetinaNet, and SSD strategies.

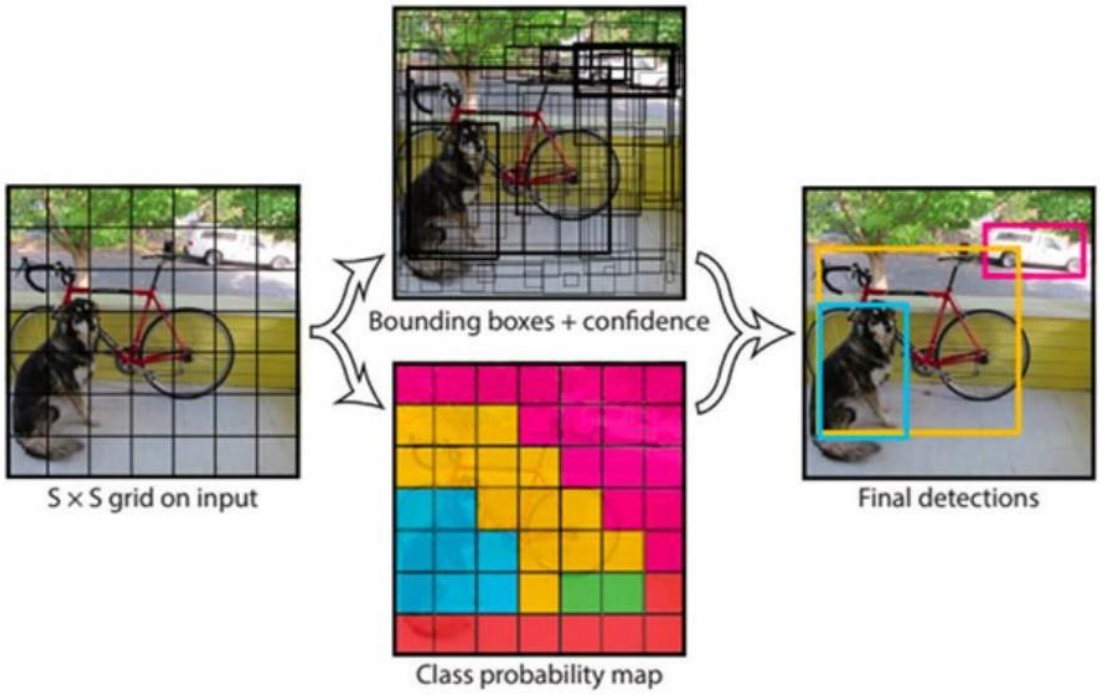


Figure 4.2 : Architecture of YOLO object detection

4.4 Pseudo code for YOLO algorithm

```
image = readImage()
NoOfCells = 7
NoOfClasses = 4
threshold = 0.7
step = height(image)/NoOfCells
prediction_class_array = new_array(size(NoOfCells,NoOfCells,NoOfClasses))
predictions_bounding_box_array =
new_array(size(NoOfCells,NoOfCells,NoOfCells,NoOfCells))
final_predictions = []
for (i<0; i<NoOfCells; i=i+1):
        for (j<0; j<NoOfCells;j=j+1):
                cell = image(i:i+step,j:j+step)
                prediction_class_array[i,j] = class_predictor(cell)
                predictions_bounding_box_array[i,j] =
bounding_box_predictor(cell)
                best_bounding_box =  [0 if predictions_bounding_box_array[i,j,0,
4] > predictions_bounding_box_array[i,j,1, 4] else 1]


                predicted_class =
index_of_max_value(prediction_class_array[i,j])
                if predictions_bounding_box_array[i,j,best_bounding_box, 4] *
max_value(prediction_class_array[i,j]) > threshold:
        final_predictions.append([predictions_bounding_box_array[i,j,best_boun
ding_box, 0:4], predicted_class])
print final_predictions
```

4.5 YOLO based VGG-16 Architecture

The backbone network in YOLO is the VGG16 network, was to evaluate the network's accuracy on the Ship SAR dataset. VGG16 is an object recognition network that uses a convolutional neural network (CNN) with 16 layers [11] to recognize a 224 x 224 image as input. It possesses 3 fully-connected layers, the last of which is a Softmax layer that contains the output features corresponding to each of the seven image classes. It is pretrained on ImageNet [5].

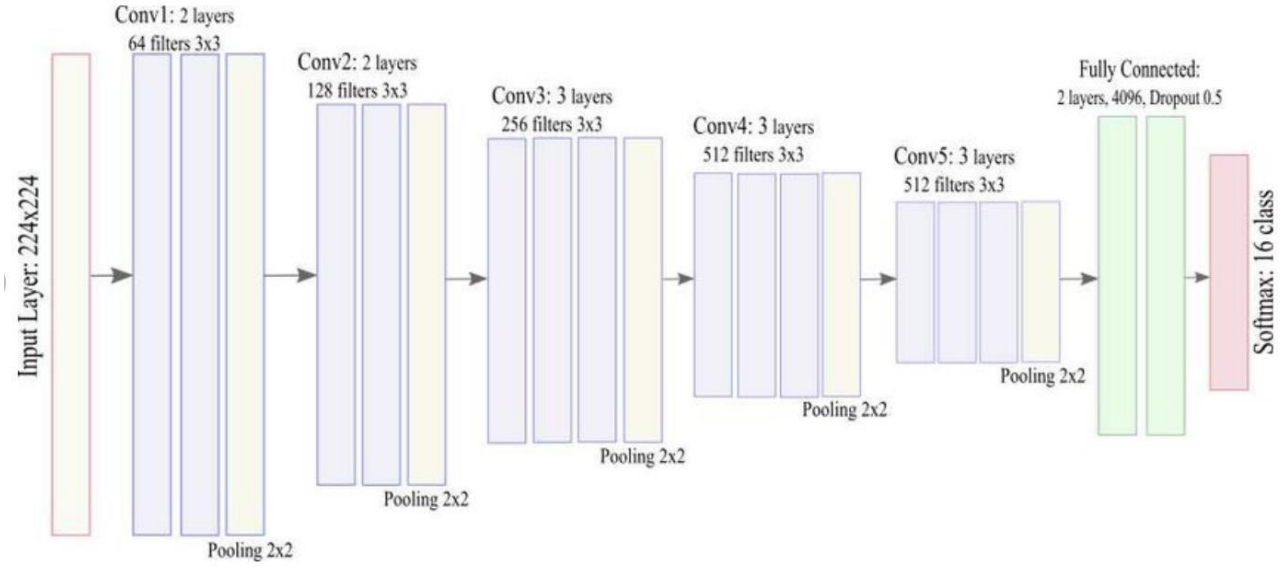


Figure 4.3 : Architecture of primary YOLO based VGG-16

4.6 Pseudo code for primary YOLO based VGG-16 algorithm

```
vgg16 = applications.VGG16(include_top=False, weights='imagenet')

bottleneck_features_train = vgg16.predict_generator(generator,
predict_size_train)
np.save('bottleneck_features_train.npy', bottleneck_features_train)

bottleneck_features_validation = vgg16.predict_generator(

    generator, predict_size_validation)
np.save('bottleneck_features_validation.npy', bottleneck_features_validation)

bottleneck_features_test = vgg16.predict_generator(

    generator, predict_size_test)
np.save('bottleneck_features_test.npy', bottleneck_features_test)
```

4.7 Framework Modules

The primary algorithm for the YOLO is VGG-16, which is mainly used for the ship detection framework which can easily make the analysis using the convolution layers in the network, which gives top accuracy and computational efficiency.

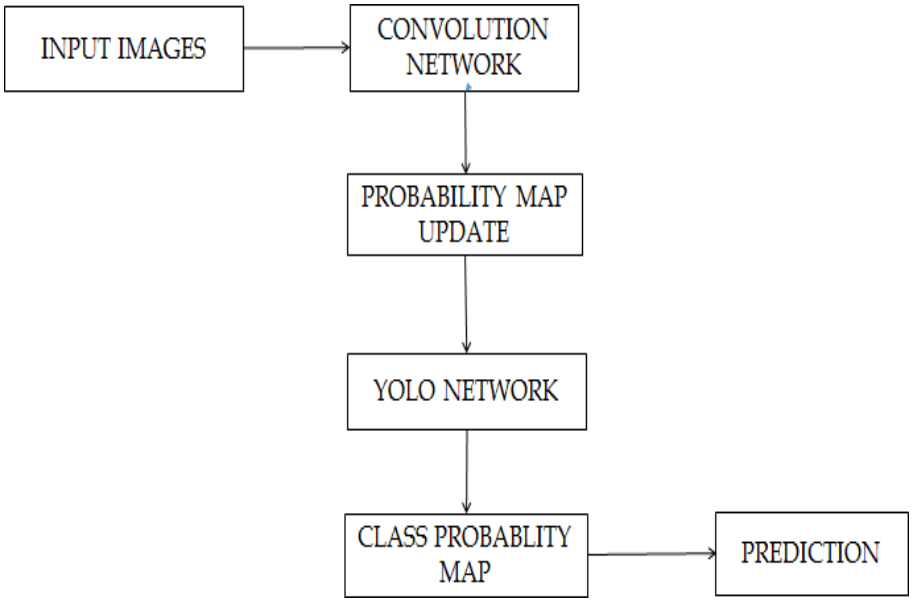


Figure 4.4 : Proposed Model Flow

The modules in this ship detection framework realize the following steps: Step 1: Data Collection which collects the data from the SAR free open dataset (SFOD) named SAR Ship Dataset.

Step 2: Data preprocessing is a technique that transforms raw data into an understandable format.

Step 3: Training the model which realizes image classification that is ten classes is categorized, each level takes five images

Step 4: Object detection which achieves a hybrid YOLO algorithm, which uses YOLO-VGG16 for object detection added with Spatial Based Pixel Descriptor (SEPD) and K-Means Clustering.

Step 5: Optimizer Analysis is then used for better screening and training of images.

Activation function determines the output for deep learning network whether it is activated or not, the input is relevant for the prediction of the model. The output of each neuron has the finite value ranges between 0 to1 or -1 to 1. It has its computational efficiency. In this case LeakyRelu and Softmax are used as activation functions.

LeakyRelu activation function is a hyperparameter, whose value is set before the learning process begins. Instead of being zero, leaky Relu has a slight non –zero limits. Softmax activation function calculates the probability dissemination of each objective class over all the probable objective modules, and then finally the probabilities calculated will be helpful in determining the objective modules for the given inputs.

Optimizer helps in minimizing or maximizing the error function. It trains the model effectively and efficiently. Optimization algorithm produces accurate results. In the simulation, RMSprop and Cross entropy Loss are realized as optimization functions.

RMSprop optimizer is used for increasing the learning rate in the ship detection framework and this optimizer algorithm takes loftier phases in horizontal direction congregating more rapid learning rate and efficiency.

CHAPTER 5

SYSTEM SPECIFICATION

- Install Python on your computer system

1. Install ImageAI and its dependencies like tensorflow, Numpy,OpenCV, etc.

2. Download the Object Detection model file(Retinanet)

5.1 Steps to followed :-

1) Download and install Python version 3 from official Python Language website

https://python.org

- Install the following dependencies via pip:

5.2 Tensorflow:

TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is an symbolic math library, and is also used for machine learning application such as neural networks, etc. It is used for both research and production by Google.

TensorFlow is developed by the Google Brain team for internal Google use. It is released under the Apache License 2.0 on November 9,2015.

TensorFlow is Google Brain's second-generation system.1st Version of tensor flow was released on February 11, 2017.While the reference implementation runs on

single devices, TensorFlow can run on multiple CPU's and GPU (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on various platforms such as64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

The architecture of TensorFlow allows the easy deployment of computation across a variety of platforms (CPU's, GPU's, TPU's), and from desktops - clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

- pip install tensor flow -command :

5.3 NumPy:

NumPy is library of Python programming language, adding support for large, multi-dimensional array and matrices, along with large collection of high-level mathematical function to operate over these arrays.

The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several developers. In 2005 Travis Oliphant created NumPy by incorporating features of computing Namarari into Numeric, with extension modifications. NumPy is open-source software and has many contributors.

- pip install NumPy -command

5.4 SciPy:

SciPy contain modules for many optimizations, linear algebra, integration, interpolation, special function, FFT, signal and image processing, ODE solvers and other tasks common in engineering.

SciPy abstracts majorly on NumPy array object, and is the part of the NumPy stack which include tools like Matplotlib, pandas and Simply, etc., and an expanding set of scientific computing libraries. This NumPy stack has similar uses to other applications such as MATLAB, Octave, and Skylab. The NumPy stack is also sometimes referred as the SciPy stack.

The SciPy library is currently distributed under BSD license, and its development is sponsored and supported by an open community of developers. It is also supported by unfocused, community foundation for supporting reproducible and accessible science.

- pip install scipy -command

5.5 OpenCV:

OpenCV is a library of programming functions mainly aimed on real time computer vision. originally developed by Intel, it is later supported by Willow Garage then Itzel. The library is a cross-platform and free to use under the open-source BSD license.

- pip install OpenCV-python -command

5.6 Pillow:

Python Imaging Library is a free Python programming language library that provides support to open, edit and save several different formats of image files. Windows, Mac OS X and Linux are available for this.

- pip install pillow -command

5.7 Matplotlib:

Matplotlib is a Python programming language plotting library and its NumPy numerical math extension. It provides an object-oriented API to use general-purpose GUI toolkits such as Tintern, python, Qt, or GTK+ to embed plots into applications.

- pip install h5py

5.8 Keras:

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or Plaid ML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

- pip install keras

5.9 Image AI:

Image AI provides API to recognize 1000 different objects in a picture using pre-trained models that were trained on the ImageNet-1000 dataset. The model implementations provided are Squeeze Net, ResNet, InceptionV3 and Dense Net.

- pip3 install imageai --upgrade

3) Download the RetinaNet model file that will be used for object detection using following link

https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/resnet50_coco_best_v2.0.1.h5

Copy the RetinaNet model file and the image you want to detect to the folder that contains the python file.

CHAPTER 6

IMPLEMENTATION AND RESULTS

6.1 Before Detection:

Figure 6.1 : Before Detection

This is a sample image we feed to the algorithm and expect our algorithm to detect and identify objects in the image and label them according to the class assigned to it.

6.2 After Detection:

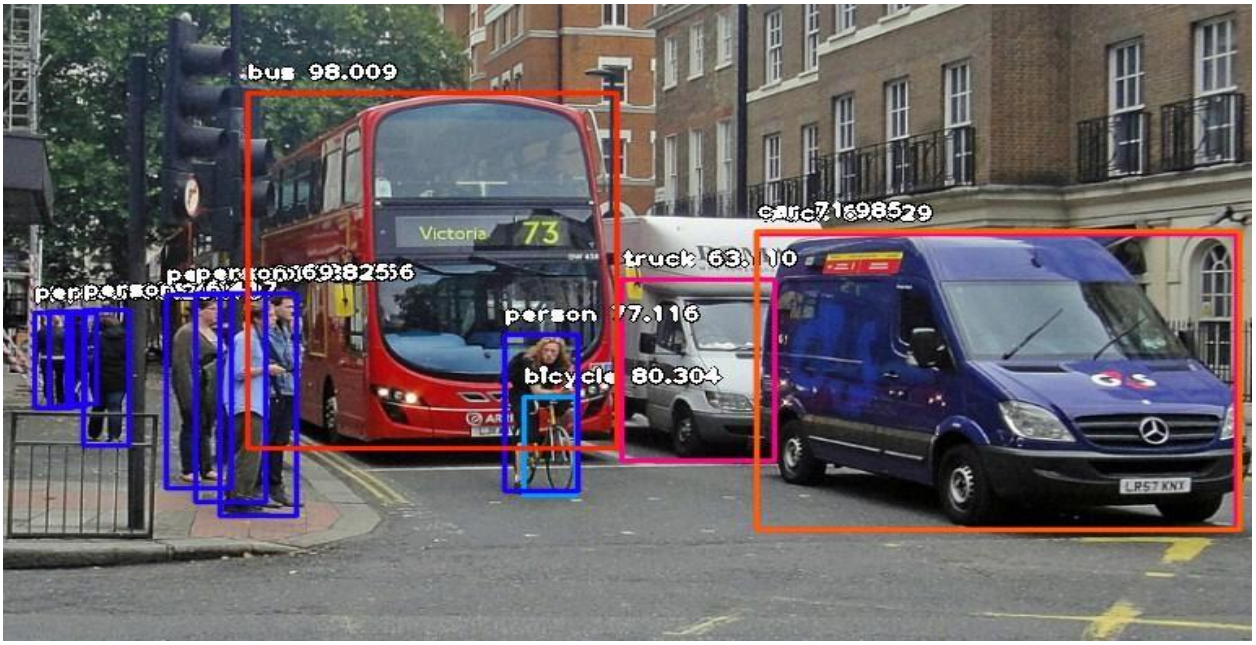


Figure 6.2 : After Detection

As expected, our algorithm identifies the objects by its classes an assigns each object by its tag and has dimensions on detected image.

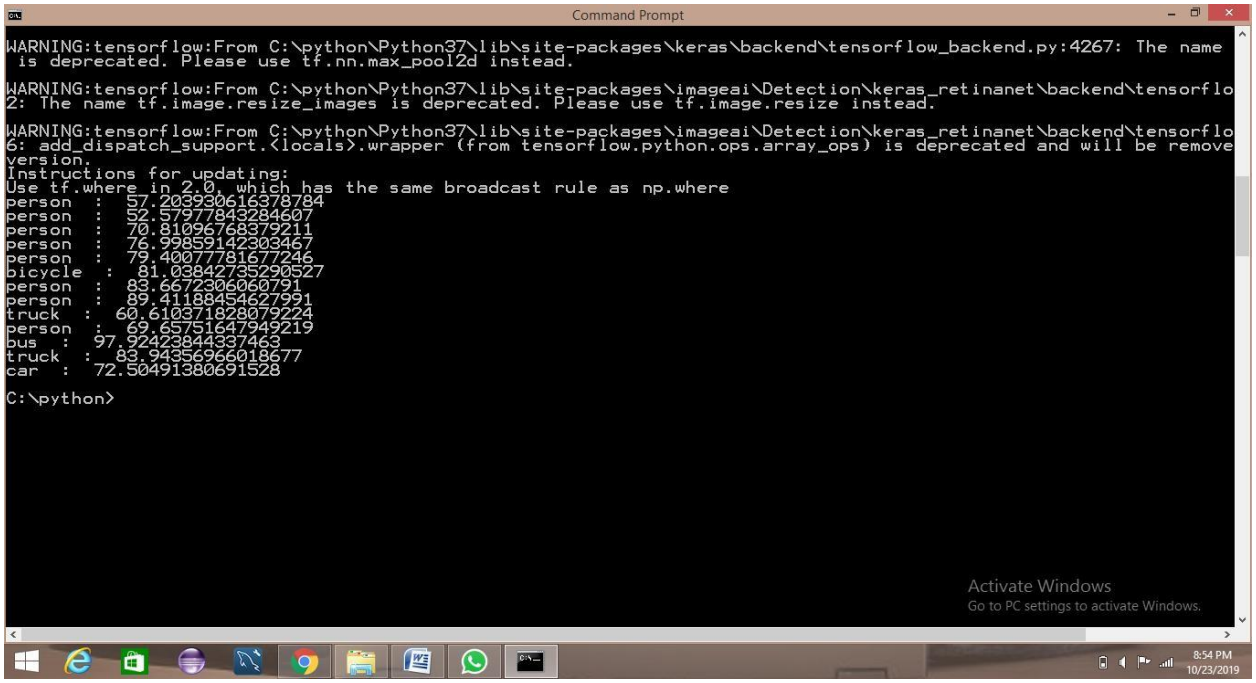


Figure 6.3 : console result for above image

Image AI provides many more features useful for customization and production capable deployments for object detection tasks. Some of the features supported are:

- Adjusting Minimum Probability: By default, objects detected with a probability percentage of less than 50 will not be shown or reported. You can increase this value for high certainty cases or reduce the value for cases where all possible objects are needed to be detected.

- Custom Objects Detection: Using a provided Custom Object class, you can tell the detection class to report detections on one or a few numbers of unique objects.

- Detection Speeds: You can reduce the time it takes to detect an image by setting the speed of detection speed to “fast”, “faster” and “fastest”.

- Input Types: You can specify and parse in file path to an image, NumPy array or file stream of an image as the input image.

- Output Types: You can specify that the detect Objects from Image function should return the image in the form of a file or NumPy array.

6.3 Detection Speed:-

Image AI now provides detection speeds for all object detection tasks. The detection speeds allow you to reduce the time of detection at a rate between 20% - 80%, and yet having just slight changes but accurate detection results. Coupled with lowering the minimum-percentage-probability parameter, detections can match the normal speed and yet reduce detection time drastically. The available detection speeds are **"normal"**(default), **"fast"**, **"faster"** , **"fastest"** and **"flash"**. All you need to do is to state the speed mode you desire when loading the model in the code.

```
detector.loadModel(detection_speed="fast")
```

6.4 Hiding/Showing Object Name and Probability:-

Image AI provides options to hide the name of objects detected and / or the percentage probability from being shown on the saved/returned detected image. Using the detect Objects from Image() and detect Custom Objects from Image() functions, the parameters display-object-name anddisplay percentage-probability can be set to True of False individually.

```
detections=detector.detectObjectsFromImage(input_image=os.path.join(execution_path,"image3.jpg"),output_image_path=os.path.join(execution_path,"image3new_nodetails.jpg"), minimum_percentage_probability=30,display_percentage_probability=False, display_object_name=False)
```

CHAPTER 7

PERFORMANCE COMPARISON

7.1 COMPARISON ANALYSIS

The proposed model is compared with existing frameworks for ship detection such as Faster RNN, CNN and Retina Net [16]. Table II depicts the performance analysis of the Object detection using Object detection dataset which was performed using various existing algorithms and the proposed Hybrid YOLO-VGG16 model.

Table 2: Comparison Analysis

Models	Accuracy	Precision	Recall	F1-score
CNN	0.57	0.70	0.50	0.67
RNN	0.62	0.73	0.60	0.33
Faster-RNN	0.65	0.79	0.70	0.69
RetinaNet	0.67	0.85	0.80	0.64
Hybrid YOLO-VGG16	**0.72**	**0.89**	**0.80**	**0.80**

7.2 Accuracy

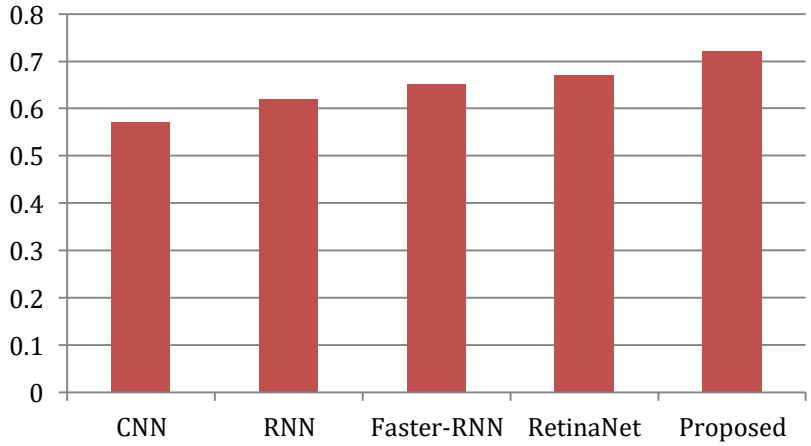


Figure 7.1 : Accuracy Comparison

Figure 7.1 inscribes the accuracy performance of proposed hybrid YOLO-VGG16 compared with existing algorithms. It proves that the proposed model achieves an accuracy of 0.72 which is better than other existing frameworks.

7.3 Precision

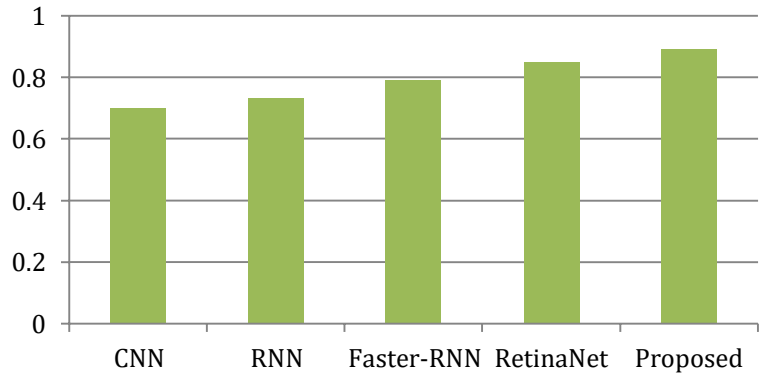


Figure 7.2 : Precision Comparison

Figure.7.2 engraves the Precision performance of proposed hybrid YOLO-VGG16 compared with existing algorithms. It proves that the proposed model achieves an accuracy of 0.89 which outperforms all other existing frameworks.

7.4 Recall

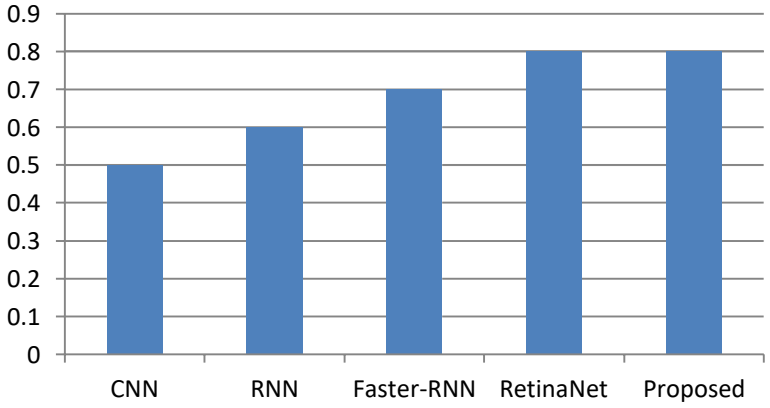


Figure 7.3 : Recall Comparison

Figure.7.3 incises the recall performance of proposed hybrid YOLO-VGG16 compared with existing algorithms. It proves that the proposed model achieves an accuracy of 0.80 which beats up the other existing frameworks.

7.5 F1-Score

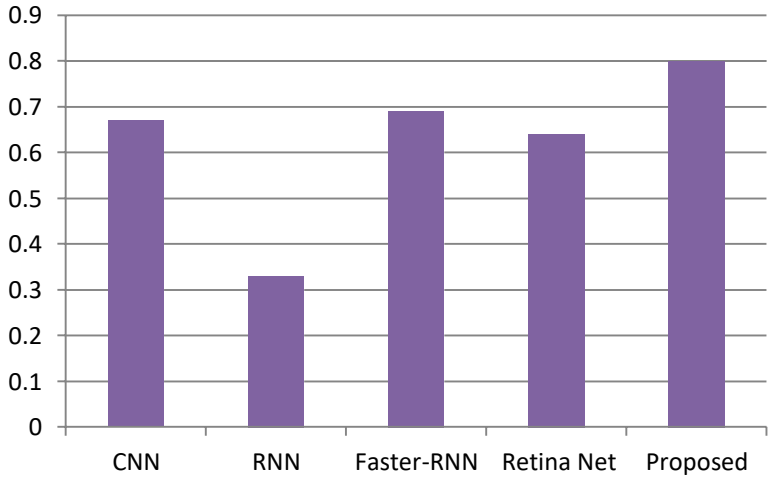


Figure 7.4 : F1-Score Comparison

Figure.7.4 etches the F1-Score performance of proposed hybrid YOLO-VGG16 compared with existing algorithms. It proves that the proposed model achieves an accuracy of 0.80 which top performs than other existing frameworks.

From the above performance analysis, the proposed hybrid YOLO-VGG16 outperforms all other efficient existing ship detection algorithms in terms of accuracy, precision, recall and F1-score.

CHAPTER 8

CONCLUSION

By using this thesis and based on experimental results we are able to detect object more precisely and identify the objects individually with exact location of an object in the picture in x, y axis. This paper also provides experimental results on different methods for object detection and identification and compares each method for their efficiencies.

APPENDIX – I

CODING

Program

```
from ultralytics import YOLO
import os
import cv2
import cvzone
import math
import time
from tkinter import *
from tkinter import Label
from tkinter import Button

# from tkinter import filedialog
# file1=filedialog.askopenfilename()

# import fileinput
window=Tk()
window.geometry("1280x520")
bg=PhotoImage(file="obj.png")
window.iconphoto(False,bg)
l=Label(window,image=bg)
l.pack(pady=0)
l1=Label(window,text=" welcome to object detection
software",bg="blue",font=30)
l1.place(x=100,y=10,height=30,width=1080)
# l2=Label(window, text="click for real time object detection",
bg="gray",font=30)
# l2.place(x=300,y=100,height=30,width=500)



#
#
model = YOLO("yolov8l.pt")

classNames = ["person", "bicycle", "car", "motorbike", "aeroplane",
"bus", "train", "truck", "boat",
              "traffic light", "fire hydrant", "stop sign", "parking
meter", "bench", "bird", "cat",
              "dog", "horse", "sheep", "cow", "elephant", "bear",
"zebra", "giraffe", "backpack", "umbrella",
              "handbag", "tie", "suitcase", "frisbee", "skis",
```

```
"snowboard", "sports ball", "kite", "baseball bat",
              "baseball glove", "skateboard", "surfboard", "tennis
racket", "bottle", "wine glass", "cup",
              "fork", "knife", "spoon", "bowl", "banana", "apple",
"sandwich", "orange", "broccoli",
              "carrot", "hot dog", "pizza", "donut", "cake",
"chair", "sofa", "pottedplant", "bed",
              "diningtable", "toilet", "tvmonitor", "laptop",
"mouse", "remote", "keyboard", "cell phone",
              "microwave", "oven", "toaster", "sink",
"refrigerator", "book", "clock", "vase", "scissors",
              "teddy bear", "hair drier", "pen"

            ]

# prev_frame_time = 0
# new_frame_time = 0
def Realtime():
    prev_frame_time = 0
    new_frame_time = 0
    cap = cv2.VideoCapture(0)  # For Webcam
    # cap = cv2.VideoCapture("../Videos/cars.mp4")  # For Video
    cap.set(3, 1280)
    cap.set(4, 720)
    # cap = cv2.VideoCapture("../Videos/motorbikes.mp4")  # For
Video

    while True:
        new_frame_time = time.time()
        success, img = cap.read()
        results = model(img, stream=True)
        for r in results:
            boxes = r.boxes
            for box in boxes:
                # Bounding Box
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                # cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,255),3)
                w, h = x2 - x1, y2 - y1
                cvzone.cornerRect(img, (x1, y1, w, h))
                # Confidence
                conf = math.ceil((box.conf[0] * 100)) / 100
                # Class Name
                cls = int(box.cls[0])

                cvzone.putTextRect(img, f'{classNames[cls]} {conf}',
(max(0, x1), max(35, y1)), scale=1, thickness=1)
```

```
        fps = 1 / (new_frame_time - prev_frame_time)
        prev_frame_time = new_frame_time
        print(fps)

        cv2.imshow("Image", img)
        cv2.waitKey(1)
def carExample():
    prev_frame_time = 0
    new_frame_time = 0
    # cap = cv2.VideoCapture(0)  # For Webcam
    cap = cv2.VideoCapture("cars.mp4")  # For Video
    cap.set(3, 1280)

   cap.set(4, 720)
    # cap = cv2.VideoCapture("../Videos/motorbikes.mp4")  # For
Video

    while True:
        new_frame_time = time.time()
        success, img = cap.read()
        results = model(img, stream=True)
        for r in results:
            boxes = r.boxes
            for box in boxes:
                # Bounding Box
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                # cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,255),3)
                w, h = x2 - x1, y2 - y1
                cvzone.cornerRect(img, (x1, y1, w, h))
                # Confidence
                conf = math.ceil((box.conf[0] * 100)) / 100
                # Class Name
                cls = int(box.cls[0])

                cvzone.putTextRect(img, f'{classNames[cls]} {conf}',
(max(0, x1), max(35, y1)), scale=1, thickness=1)

        fps = 1 / (new_frame_time - prev_frame_time)
        prev_frame_time = new_frame_time
        print(fps)

        cv2.imshow("Image", img)
        cv2.waitKey(1)
def bikeExample():
    prev_frame_time = 0
    new_frame_time = 0
    # cap = cv2.VideoCapture(0)  # For Webcam
```

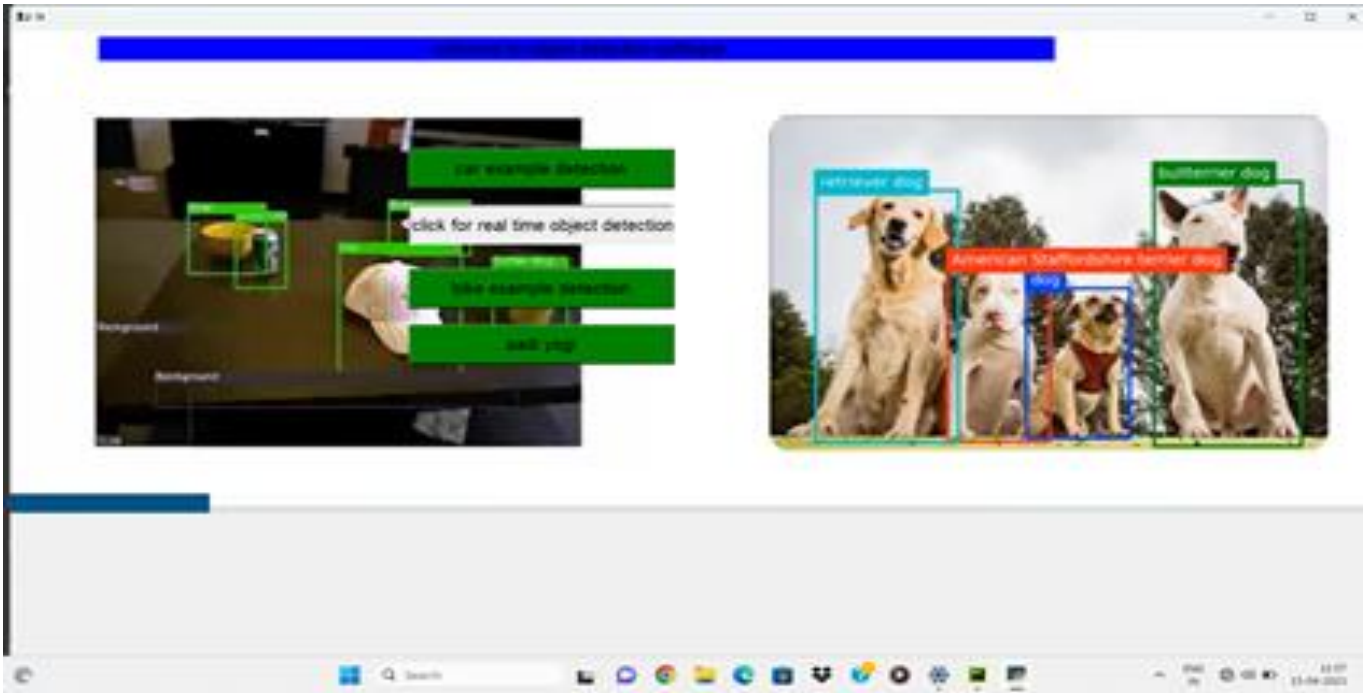
```
    cap = cv2.VideoCapture("motorbikes.mp4")  # For Video
    cap.set(3, 1280)
    cap.set(4, 720)
    # cap = cv2.VideoCapture("../Videos/motorbikes.mp4")  # For
Video

    while True:
        new_frame_time = time.time()
        success, img = cap.read()
        results = model(img, stream=True)
        for r in results:
            boxes = r.boxes
            for box in boxes:

                # Bounding Box
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                # cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,255),3)
                w, h = x2 - x1, y2 - y1
                cvzone.cornerRect(img, (x1, y1, w, h))
                # Confidence
                conf = math.ceil((box.conf[0] * 100)) / 100
                # Class Name
                cls = int(box.cls[0])

                cvzone.putTextRect(img, f'{classNames[cls]} {conf}',
(max(0, x1), max(35, y1)), scale=1, thickness=1)

        fps = 1 / (new_frame_time - prev_frame_time)
        prev_frame_time = new_frame_time
        print(fps)

        cv2.imshow("Image", img)
        cv2.waitKey(1)


def sample():
    prev_frame_time = 0
    new_frame_time = 0
    # cap = cv2.VideoCapture(0)  # For Webcam
    cap = cv2.VideoCapture("adiyogii.mp4")  # For Video
    cap.set(3, 1280)
    cap.set(4, 720)
    # cap = cv2.VideoCapture("../Videos/motorbikes.mp4")  # For
Video

    while True:
        new_frame_time = time.time()
```

```
        success, img = cap.read()
        results = model(img, stream=True)
        for r in results:
            boxes = r.boxes
            for box in boxes:
                # Bounding Box
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                # cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,255),3)
                w, h = x2 - x1, y2 - y1
                cvzone.cornerRect(img, (x1, y1, w, h))
                # Confidence

             conf = math.ceil((box.conf[0] * 100)) / 100
                # Class Name
                cls = int(box.cls[0])

                cvzone.putTextRect(img, f'{classNames[cls]} {conf}',
(max(0, x1), max(35, y1)), scale=1, thickness=1)

        fps = 1 / (new_frame_time - prev_frame_time)
        prev_frame_time = new_frame_time
        print(fps)

        cv2.imshow("Image", img)
        cv2.waitKey(1)

B1=Button(text="car example
detection",bg="green",font=30,command=carExample)
B1.place(x=450,y=150,height=50,width=300)

B2=Button(text="click for real time object detection",
bg="green",font=30, command=Realtime)
B2.place(x=450,y=220,height=50, width=300)
B2=Button(text="bike example detection", bg="green",font=30,
command=bikeExample)
B2.place(x=450,y=300,height=50, width=300)

B2=Button(text="aadi yogi", bg="green",font=30,command=sample)
B2.place(x=450,y=370,height=50, width=300)

# file1=Label(window,text="please select a file (video or
image)",bg="orange",font=30)
# file1.place(x=380,y=100,height=50,width=500)


window.mainloop()
```

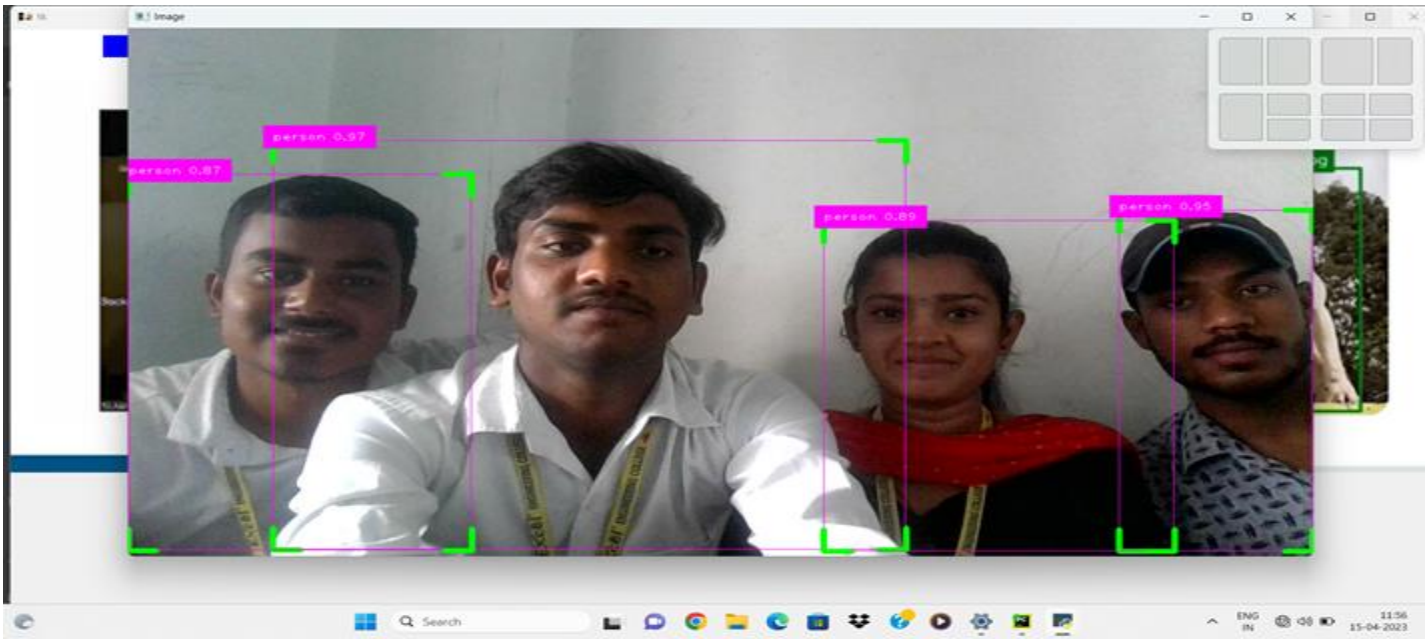
APPENDIX – II

OUTPUT

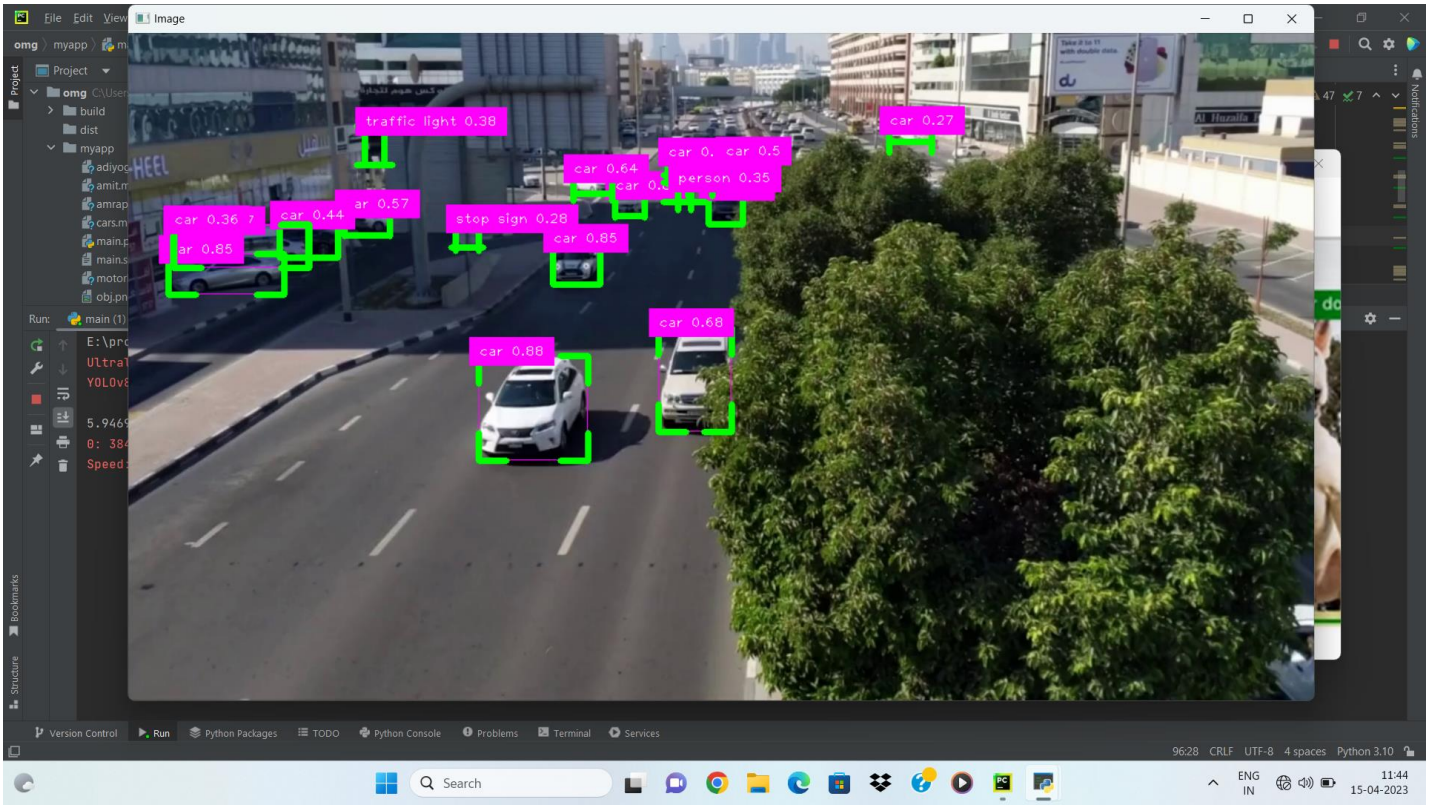
GUI Interface



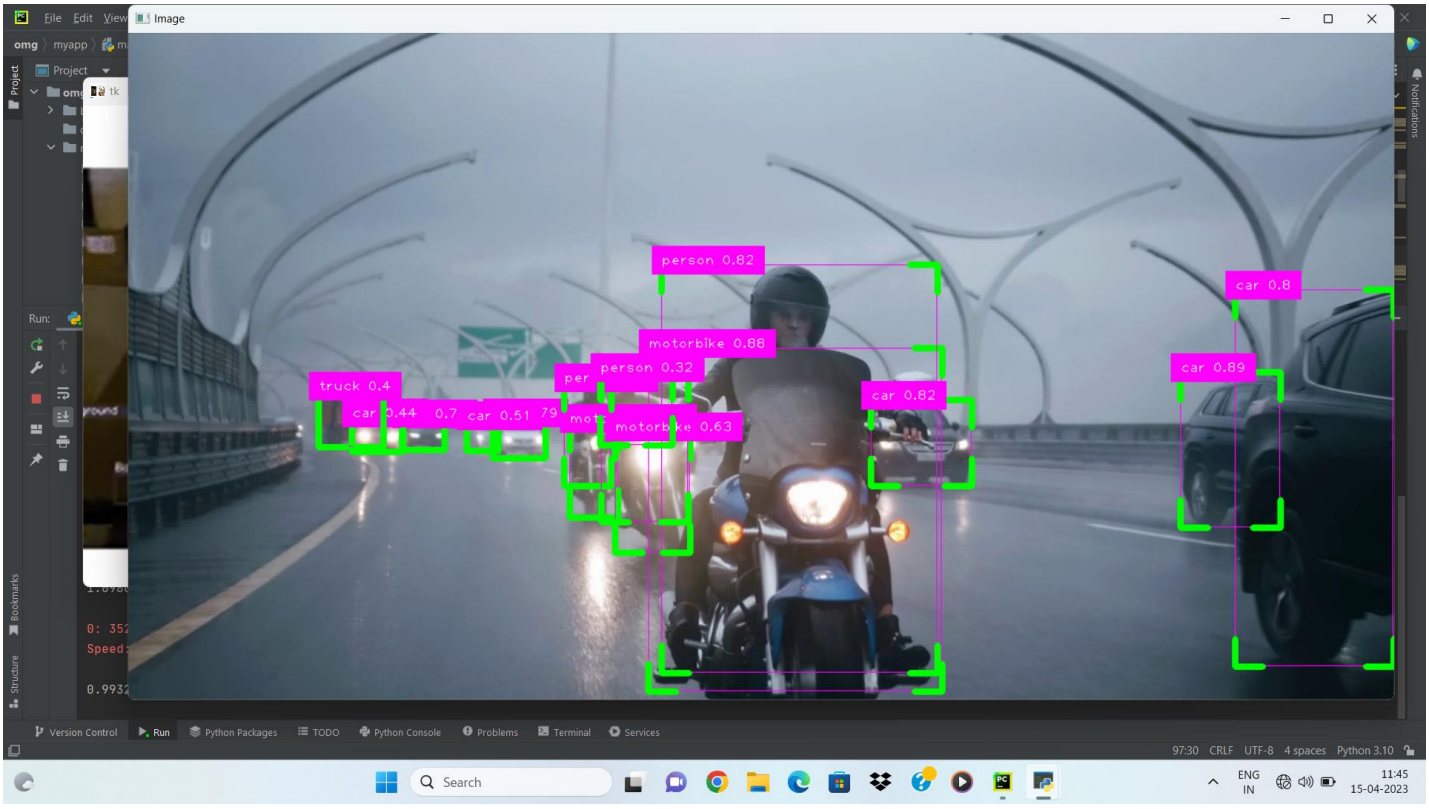
Real time image



Car Detection



Bike Detection



REFERENCE

1 "Object Detection with Deep Learning: A Review" by Muhammad Umer, Adel Abusitta, and Abdullah Alwadie, IEEE Access (2019). This paper provides an overview of different deep learning techniques for object detection, including region-based methods, single-shot detectors, and two-stage detectors.

2 "Image Classification using Convolutional Neural Networks" by Adrian Rosebrock, PyImageSearch (2018). This article explains how to use Convolutional Neural Networks (CNNs) to classify images, and provides a step-by-step tutorial using Python and the Keras library.

3 "YOLO: Real-Time Object Detection" by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, CVPR (2016). This paper introduces the YOLO (You Only Look Once) algorithm for object detection, which is fast and accurate, and can run in real-time.

4 "A Survey on Deep Learning Techniques for Image and Video Analysis" by Saeed Shokri, Mohammad Hossein Rohban, and Amirhossein Taherinia, Journal of Big Data (2021). This paper provides an overview of deep learning techniques for image and video analysis, including image classification, object detection, semantic segmentation, and instance segmentation.

5 "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" by Mingxing Tan and Quoc V. Le, ICML (2019). This paper introduces the EfficientNet architecture for image classification, which achieves state-of-the-art accuracy with fewer parameters than previous models.

6 "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" by Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, NIPS (2015). This paper introduces the Faster R-CNN algorithm for object detection, which combines a region proposal network with a detection network, and achieves state-of-the-art performance on several benchmarks.

7 "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, NIPS (2012). This paper introduces the AlexNet architecture for image classification, which was the first deep neural network to win the ImageNet Large Scale Visual Recognition Challenge, and sparked the deep learning revolution