

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY  
BELAGAVI-590018**



**“PROJECT WORK PHASE - 2”  
(18CSP83)  
REPORT ON**

**“REAL TIME OBJECT DETECTION USING YOLO”**

Submitted in partial fulfillment for the requirements for the Award of Degree of

**BACHELOR OF ENGINEERING  
IN  
INFORMATION SCIENCE AND ENGINEERING  
BY**

<b>ASHWAQULLA BAIG</b>	<b>1EP19IS011</b>
<b>AVINASH KUMAR SINGH</b>	<b>1EP19IS012</b>
<b>BAL KISHAN REDDY</b>	<b>1EP19IS013</b>
<b>IRAM SABHA A</b>	<b>1EP19IS038</b>

UNDER THE GUIDANCE OF

Mrs. Teena KB  
Assistant Professor  
Dept. of ISE, EPCET



**Department of Information Science and Engineering**

Approved by AICTE New Delhi| Affiliated to VTU, Belagavi  
Virgo Nagar, Bengaluru-560049

**2022-23**



(Affiliated to Visvesvaraya Technological University, Belgavi)

Bangalore-560049

## DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

### CERTIFICATE

This is to certify that the **Project Work** entitled “**REAL TIME OBJECT DETECTION USING YOLO**” carried out by Ms. **IRAM SABHA A**, USN **1EP19IS038**, Mr. **ASHWAQULLA BAIG**, USN **1EP19IS011**, Mr. **AVINASH KUMAR SINGH**, USN **1EP19IS012**, Mr. **BAL KISHAN REDDY**, USN **1EP19IS013**, bonafide students of East Point College of Engineering and Technology in partial fulfillment for the award of Bachelor of Engineering in Information Science and Engineering of Visvesvaraya Technological University, Belagavi during the year **2022-23**. It is certified that all the corrections/suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of Project Work Phase-2 (18CSP83) prescribed for the award of the said degree.

**GUIDE**  
**Prof. Teena KB**  
Asst. Professor

**HOD**  
**Dr. Lingaraju G M**  
EPCET

**PRINCIPAL**  
**Dr. Yogesh G S**  
EPCET

### Examiners

Name of the Examiners

Signature with date

1.

2.

## ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. We would like to take this opportunity to thank them all.

First and foremost, we would like to express our sincere regards and thanks to **Mr. Pramod Gowda** and **Mr. Rajiv Gowda**, CEOs, East Point Group of Institutions, Bangalore, for providing necessary infrastructure and creating good environment.

We express our gratitude to **Dr. Prakash S**, Senior Vice President, EPGI and **Dr. Yogesh G S**, Principal, EPCET who has always been a great source of inspiration.

We express our sincere regards and thanks to **Dr. Lingaraju G M**, Professor and Head, Department of Information Science and Engineering, EPCET, Bangalore, for his encouragement and support.

We are obliged to **Dr. Udayabalan B**, Associate Professor, ISE, who rendered valuable assistance as the Project coordinator.

We are grateful to acknowledge the guidance and encouragement given to us by **Prof. Teena KB, Assistant Professor**, Department of Information Science and Engineering, EPCET, Bangalore, who has rendered a valuable assistance.

We also extend our thanks to the entire faculty of the Department of **Information Science and Engineering, EPCET**, Bangalore, who have encouraged us throughout the course of the Project.

Last, but not the least, we would like to thank our family and friends for their inputs to improve the Project.

<b>ASHWAQULLA BAIG</b>	<b>1EP19IS011</b>
<b>AVINASH KUMAR SINGH</b>	<b>1EP19IS012</b>
<b>BAL KISHAN REDDY</b>	<b>1EP19IS013</b>
<b>IRAM SABHA A</b>	<b>1EP19IS038</b>

# **ABSTRACT**

Real time object detection is a vast, vibrant and complex area of computer vision. If there is a single object to be detected in an image, it is known as Image Localization and if there are multiple objects in an image, then it is Object Detection. This detects the semantic objects of a class in digital images and videos. The applications of real time object detection include tracking objects, video surveillance, pedestrian detection, people counting, self-driving cars, face detection, ball tracking in sports and many more. Convolution Neural Networks is a representative tool of Deep learning to detect objects using Open CV (Open source Computer Vision), which is a library of programming functions mainly aimed at real time computer vision.

**Keywords:** Computer vision, Deep Learning, Convolution Neural Networks

# CONTENTS

CHAPTER NO.	TOPICS	PAGE NO.
1.	<b>Introduction</b>	<b>1-11</b>
	1.1 Introduction to the project	
	1.2 Problem Statement	
	1.3 What is Digital Image Processing	
	1.4 Why Image Processing	
	1.5 Existing Methods	
	1.6 Chapter Summary	
2.	<b>Literature Survey</b>	<b>12-14</b>
3.	<b>System Requirement Specification</b>	<b>15-28</b>
	3.1 Introduction	
	3.2 System Architecture	
	3.3 Functional Requirements	
	3.4 Non-Functional Requirements	
	3.5 System Model	
4.	<b>Software Design</b>	<b>29-47</b>
	4.1 System Architecture & Decomposition	
	4.2 Data Models	
	4.3 Algorithmic Design & User Interface Design	
	4.4 Design Of The Network	
5.	<b>Implementation</b>	<b>48-59</b>
	5.1 Implementation of YOLOv8	
	5.2 Coding	
6.	<b>Software Test Specification</b>	<b>60-67</b>
	6.1 Training	
	6.2 Testing Strategies	
7.	<b>Results &amp; Snapshots</b>	<b>68-71</b>
	<b>Conclusion</b>	<b>72-73</b>
	<b>References</b>	<b>74</b>

## LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	Types of Image Processing	2
1.2	Digital Image	3
1.3	Fast R-CNN	5
1.4	Faster R-CNN	5
1.5	Non Maximum Suppression	8
1.6	Object detection and image classification	10
3.1	Yolo Predict	17
3.2	Yolo Framework	19
3.3	Unified Detection	20
3.4	Use case for Real-time Object Detection	27
3.5	Dataflow representation for Real-time Object Detection	28
4.1	Class Model for YOLO	31
4.2	Dataflow representation for YOLO	31
4.3	Structure of a typical CNN	32
4.4	Image Processes involved in R-CNN	33
4.5	Image data flow in R-CNN	34
4.6	Image data flow in Fast R-CNN	35
4.7	Image data flow in Faster R-CNN	36
4.8	ROI Pooling Layer Working in Faster R-CNN	37
4.9	Structure of YOLO	39
4.10	YOLO Configuration	43
4.11	Structure of YOLOv3	43
4.12	Anchor Box Detection	44
4.13	Benchmark scores of YOLOv3 and other networks against COCO mAP 50	46
4.14	Benchmark scores of object detection networks against COCO Dataset	47

5.1	80 Custom Dataset Classes on which YOLOv8 was Trained	48
7.1	Pre-processing of the video	68
7.2	Model Runtime and frame info on the input video	68
7.3	Output video from the model after detection and tracking	69
7.4	People Counting Using OpenCV	70
7.5	Car Counting Using OpenCV	70
7.6	Basic Image Testing Using Yolo	71
7.7	Face Recognition System	71

## **LIST OF TABLES**

<b>TABLE NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
3.1	Functional Requirements	21
7.1	System Testing	63-67



## **CHAPTER 1**

# **INTRODUCTION**

### **1.1 Introduction to The Project**

Object Detection is the process of finding and recognizing real-world object instances such as car, bike, TV, flowers, and humans out of an images or videos. An object detection technique lets you understand the details of an image or a video as it allows for the recognition, localization, and detection of multiple objects within an image. It is usually utilized in applications like image retrieval, security, surveillance, and advanced driver assistance systems (ADAS).

- **Digital Image Processing**

Computerized picture preparing is a range portrayed by the requirement for broad test work to build up the practicality of proposed answers for a given issue. A critical trademark hidden the plan of picture preparing frameworks is the huge level of testing and experimentation that Typically is required before touching base at a satisfactory arrangement. This trademark informs that the capacity to plan approaches and rapidly model hopeful arrangements by and large assumes a noteworthy part in diminishing the cost and time required to land at a suitable framework execution.

- **What Is Digital Image Processing?**

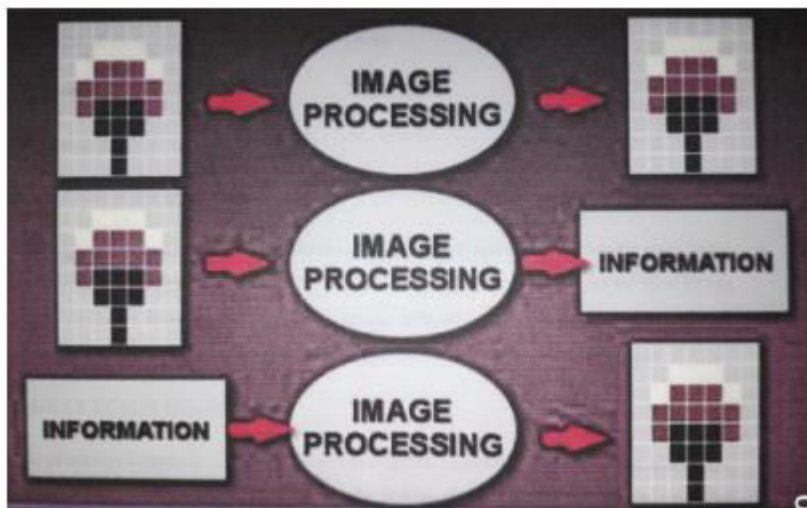
A picture might be characterized as a two-dimensional capacity  $f(x, y)$ , where  $x, y$  are spatial directions, and the adequacy off at any combine of directions  $(x, y)$  is known as the power or dark level of the picture by then. Whenever  $x, y$  and the abundance estimation of are all limited discrete amounts, we call the picture a computerized picture. The field of DIP alludes to preparing advanced picture by methods for computerized PC. Advanced picture is made a limited number of components, each of which has a specific area and esteem. The components are called pixels.

### ▪ Why Image Processing?

Since the digital image is invisible, it must be prepared for viewing on one or more output device (laser printer, monitor at). The digital image can be optimized for the application by enhancing the appearance of the structures within it.

There are three of image processing used. They are

- Image to Image transformation
- Image to Information transformations
- Information to Image transformations



**Fig. 1.1 Types of Image Processing**

### • Pixel

Pixel is the smallest element of an image. Each pixel corresponds to any one value. In an 8-bit gray scale image, the value of the pixel between 0 and 255. Each pixel stores a value proportional to the light intensity at that particular location. It is indicated in either Pixels per inch or Dots per inch.

### • Resolution

The resolution can be defined in many ways. Such as pixel resolution, spatial resolution, temporal resolution, spectral resolution. In pixel resolution, the term resolution refers to the total number of count of pixels in an digital image. For example, If an image has M rows

and N columns, then its resolution can be defined as  $M \times N$ . Higher is the pixel resolution, the higher is the quality of the image.



**Fig. 1.2 Digital Image**

### **Processing on image:**

Processing on image can be of three types They are low-level, mid-level, high level.

#### **Low-level Processing:**

- Preprocessing to remove noise.
- Contrast enhancement.
- Image sharpening.

#### **Medium Level Processing:**

- Segmentation.
- Edge detection.
- Object extraction.

#### **High Level Processing:**

- Image analysis.
- Scene interpretation.

## **1.2 Problem Statement**

There are several potential problems that you may encounter when implementing live object detection using the YOLO framework:

- Limited training data: If you do not have a large and diverse enough dataset, your model may not be able to accurately detect objects in new, unseen images.
- Overfitting: If you do not use sufficient regularization, your model may over fit to the training data and perform poorly on new, unseen data.

- Poor network architecture: If you do not choose an appropriate network architecture, your model may not be able to effectively learn to detect objects in the images.
- Inefficient post-processing: If you do not implement an efficient post-processing step.
- Lack of robustness to variations in appearance: If your model is not robust to variations in the appearance of the objects.

### ▪ Existing Methods

#### • Res-Net

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 Res-Net. 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 Res-Net-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.

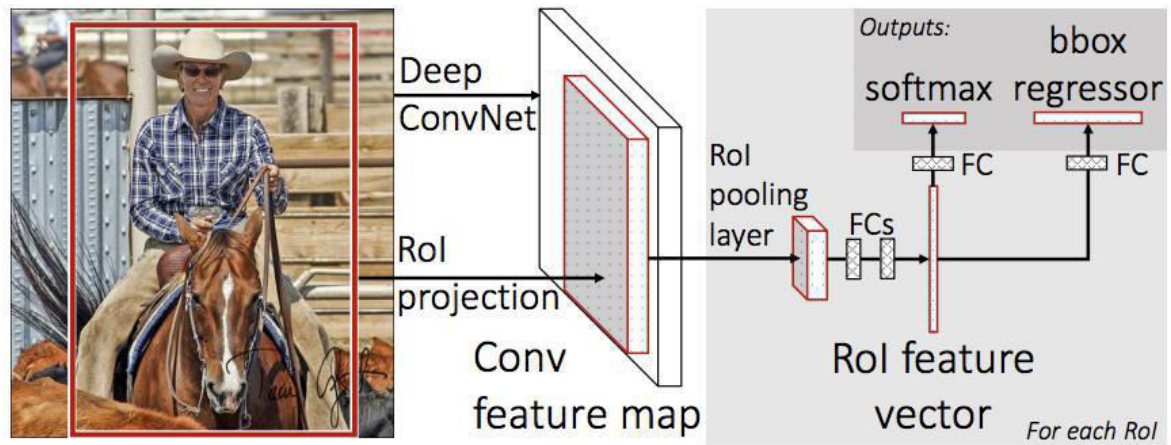
#### • R-CNN

To circumvent the problem of selecting a huge number of regions, Ross 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:

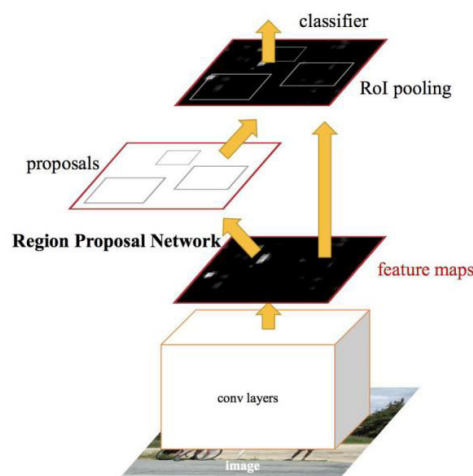
1. Generate the initial sub-segmentation, we generate many candidate regions
2. Use the greedy algorithm to recursively combine similar regions into larger ones.

- **Fast R-CNN**



**Fig. 1.3 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 ROI pooling layer we reshape them into the fixed size so that it can be fed into a fully connected layer.



**Fig. 1.4 Faster R-CNN**

### 1.4 PROPOSED SYSTEM

- There are several potential ways to improve the existing YOLO-based system for live object detection:
- Incorporate additional data augmentation techniques: By applying a wider range of data augmentation techniques during training, you can improve the robustness of your model and reduce the risk of overfitting.
- Use a more powerful CNN architecture: By using a more powerful CNN architecture, such as a deeper network or one that uses more advanced building blocks (e.g., residual connections), you may be able to improve the accuracy of your object detection system.
- Implement more efficient post-processing: You can improve the efficiency of your object detection system by implementing more efficient post-processing techniques, such as lightweight non-maximum suppression algorithms.
- Incorporate temporal information: In a live object detection system, you can leverage temporal information by incorporating information from previous frames in the video feed. This can help to improve the accuracy of your object detections, particularly for objects that are partially occluded or moving quickly.

Use domain-specific knowledge: If you are working on a specific application (e.g., detecting pedestrians in a city), you can incorporate domain-specific knowledge into your model to improve its accuracy. This could include using additional training data that is relevant to your specific application or incorporating domain-specific features into your CNN architecture.

#### ➤ What is YOLO?

You only look once (YOLO) is a state-of-the-art, real-time object detection system. Object detection is one of the classical problems in computer vision where you work to recognize what and where — specifically what objects are inside a given image and also where they are in the image. The problem of object detection is more complex than classification, which also can recognize objects but doesn't indicate where the object is located in the image. In addition, classification doesn't work on images containing more than one object.

YOLO 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.

### ➤ YOLOv3

YOLOv3 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, you can easily tradeoff between speed and accuracy simply by changing the size of the model, no retraining required! How it works? Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections. YOLOv3 uses a few tricks to improve training and increase performance, including: multiscale predictions, a better backbone classifier, and more.

### ➤ Feature Extractor Network (Darknet-53):

DarkNet-53 is a convolutional neural network that is 53 layers deep. You can load a pre-trained version of the network trained on more than a million images from the ImageNet database [1]. The pre-trained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 256-by256. DarkNet-53 is often used as the foundation for object detection problems and YOLO workflows.

YOLO v3 also uses a variant of Darknet, for the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3. This is the reason behind the slowness of YOLO v3 compared to YOLO v2. Here is how the architecture of YOLO now looks like. Input and Output Formats: Test the compatibility and integration with different input and output formats. Validate that the system can handle various image file formats, such as JPEG or PNG. Verify that the output predictions can be seamlessly integrated with other systems or tools that consume the object detection results.



### ➤ Non-maximum suppression:

We have also used the non-maximum suppression in this. Non-Maximum Suppression (NMS) is a technique used in many computer vision algorithms. It is a class of algorithms to select one entity (e.g. bounding boxes) out of many overlapping entities. The selection criteria can be chosen to arrive at particular results. Most commonly, the criteria are some form of probability number along with some form of overlap measure.



**Fig. 1.5 Non-maximum suppression**

## 1.5 Objectives of the Project

### ➤ What is Object Detection?

Object Detection is a common Computer Vision problem which deals with identifying and locating object of certain classes in the image. Interpreting the object localization can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which contains the object (called segmentation). Object detection is a key technology behind advanced driver assistance systems (ADAS) that enable cars to detect driving lanes or perform pedestrian detection to improve road safety. Object detection is also useful in applications such as video surveillance or image retrieval systems.



### ➤ Difference between Object detection and image classification?

In the field of Computer Vision, the doubt which is most common we have is what is the difference between the classification of image, detection of object and segmentation of image. Let's take an image of a dog as shown below, we start making classifications of it to which it belongs (like a dog, which is an instance here). And that what is known as the Image Classification. As we go through, we see that there was only a particular object, a dog. We can easily make use of such image classification model and try predicting that we have an image of a dog. But, just imagine the fact if we have double images like one of cat and other of dog,

- **How would we classify them?**

For this problem we need to train a multi-label classifier. By doing this we would see another problem that we aren't aware of the location of either animal/object in the image. Therefore, we make use of Image Localization which helps to identify the object location in that particular image. In that case, if we have multiple objects in an image, we go for the next step known as Object Detection. Now we are able to predict the location and the class for each object in an image using Object Detection. In order to detect the objects and classifying the images, we first require the understanding of what an image comprises of. Now comes the role of Image Segmentation. Either we can divide or make partitions of an image into parts known to be segments. It won't be reliable to make a processing of an entire image at the very same time because there will be some regions where we do not contain information of the image. By making divisions in the image and converting it into segments, we can pick some of the important segments for processing of an image. This is how Image Segmentation has its working. Knowing the definition of the image that it is a collection of pixels; we start making groups of the pixels which have similar attributes using the Image Segmentation method. After applying Object Detection method, we are only able to build up a bounding box which is corresponding to each of the class in an image. But it won't be letting us know anything about the shape of the image since the bounding boxes maybe rectangular or square in shape type.

The models of image segmentation on the other hand will be able to create a pixel-wise mask for each type of object in an image. Such type of technique gives us far more granular way of understanding the object(s) in an image. Now summarizing each method, we observed:

1. **Image Classification** helps in classification of what is contained in a particular image
2. **Image Localization** does the specification of location of a single object in a particular image
3. **Object Detection** specifies the location of multiple images
4. **Image Segmentation** does creation of pixels' wise mask of each object in a particular image

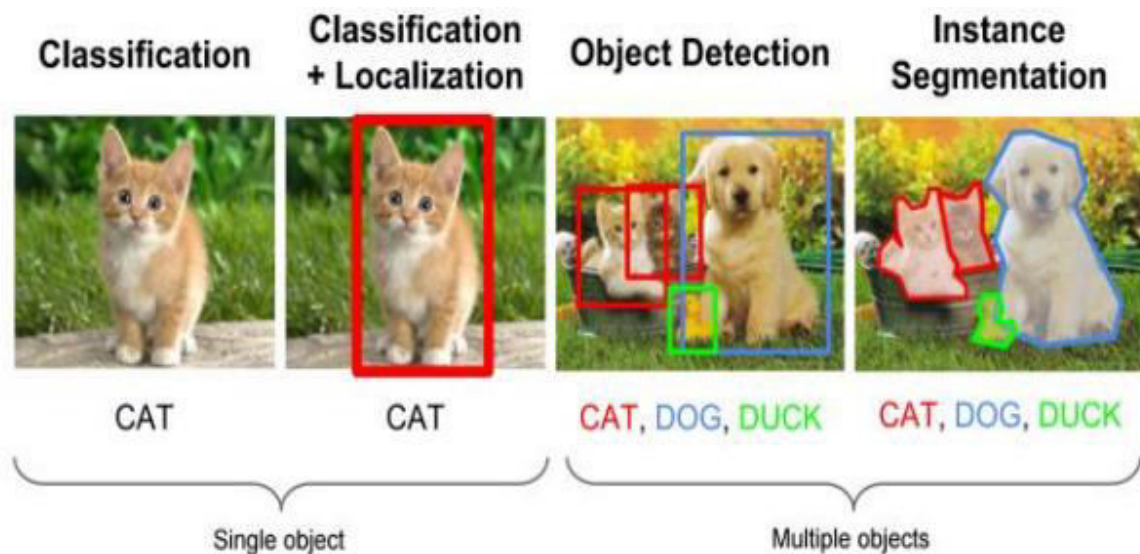


Fig. 1.6 Object detection and image classification

### 1.6 Chapter Summary

The chapter begins with an introduction to the challenges of object detection in real-time scenarios, such as video surveillance or autonomous systems, where quick and accurate detection is crucial. It highlights the limitations of traditional object detection algorithms and explains how YOLO addresses these limitations.

The YOLO algorithm is then explained in detail, starting with its architecture and working principles. The chapter covers the concept of dividing the input image into a grid and predicting bounding boxes and class probabilities for each grid cell. It also explores the different versions of YOLO, including YOLOv1, YOLOv2, and YOLOv3, highlighting their improvements and advancements.

The chapter also delves into the technical aspects of training and optimizing YOLO models, including data preparation, loss functions, and network architectures. It discusses the importance of anchor boxes and how they aid in accurate object localization. Additionally, techniques such as data augmentation and transfer learning are explored to enhance YOLO's performance.

Practical implementation and evaluation of YOLO in real-time scenarios are presented through case studies and experiments. The chapter showcases how YOLO can be used for object detection in various domains, such as traffic surveillance, robotics, and industrial automation. It discusses the challenges encountered during implementation and provides insights on fine-tuning YOLO for specific use cases.

Finally, the chapter concludes with a summary of the advantages and limitations of YOLO and highlights potential areas for future research and improvements. It emphasizes YOLO's speed, accuracy, and real-time capabilities as key factors contributing to its popularity and widespread adoption.

In essence, the chapter on "Real-Time Object Detection Using YOLO" serves as a comprehensive guide to understanding the YOLO algorithm, its implementation, and its practical applications in real-time object detection.

## **CHAPTER 2**

### **LITERATURE SURVEY**

**[1]. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,”**

Multilayer neural networks trained with the back-propagation algorithm constitute the best example of a successful gradient based learning technique. Given an appropriate network architecture, gradient-based learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns, such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional neural networks, which are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques. Real-life document recognition systems are composed of multiple modules including field extraction, segmentation recognition, and language modeling. A new learning paradigm, called graph transformer networks (GTN), allows such multi-module systems to be trained globally using gradient-based methods so as to minimize an overall performance measure. Two systems for online handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of graph transformer networks. A graph transformer network for reading a bank cheque is also described. It uses convolutional neural network character recognizers combined with global training techniques to provide record accuracy on business and personal cheques. It is deployed commercially and reads several million cheques per day.

**[2]. Joseph Redmon and Ali Farhadi, “YOLO: Real-Time Object Detection”**

YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the

whole detection pipeline is a single network, it can be optimized end-to-end directly on detection.

Unified Architecture is extremely fast. YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the MAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. Finally, YOLO learns very general representations of objects. It outperforms all other detection methods, including DPM and R-CNN, by a wide margin when generalizing from natural images to artwork on both the Picasso Dataset and the People-Art Dataset.

### **[3]. A. Dhiman, "Object Tracking using DeepSORT in TensorFlow 2," 27 October 2020**

Detecting and tracking objects are among the most prevalent and challenging tasks that a surveillance system has to accomplish in order to determine meaningful events and suspicious activities. In this article the concept of Object Tracking, challenges, traditional methods and implement such a system in TensorFlow 2.0.

Videos are actually sequences of images, each of which called a frame, displayed in fast enough frequency so that human eyes can percept the continuity of its content. It is obvious that all image processing techniques can be applied to individual frames. Besides, the contents of two consecutive frames are usually closely related.

Object detection in videos involves verifying the presence of an object in image sequences and possibly locating it precisely for recognition. Object tracking is to monitor an object's spatial and temporal changes during a video sequence, including its presence, position, size, shape, etc. This is done by solving the temporal correspondence problem, the problem of matching the target region in successive frames of a sequence of images taken at closely-spaced time intervals. These two processes are closely related because tracking usually starts with detecting objects, while detecting an object repeatedly in subsequent image sequence is often necessary to help and verify tracking.

### **[4]. Joseph Redmon, Ali Farhadi and Yann LeCun “Real-Time Object Detection using YOLO”**

Different strategies have been proposed to solve the problem of object identification throughout the years. These techniques focus on the solution through multiple stages. Namely, these core stages include recognition, classification, localization, and object detection. Along with the technological progression over the years, these techniques have been facing challenges such as output accuracy, resource cost, processing speed and complexity issues. With the invention of the first Convolutional Neural Network (CNN) algorithm in the 1990s inspired by the Neocognitron by Yann LeCun et al. and significant inventions like AlexNet, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 (thus later referred to as ImageNet) CNN algorithms have been capable of providing solutions for the object detection problem in various approaches. With the purpose of improving accuracy and speed of recognition, optimization focused algorithms such as VGGNet, Google Net and Deep Residual Learning (ResNet) have been invented over the years. Although these algorithms improved over time, window selection or identifying multiple objects from a single image was still an issue. To bring solutions to this issue, algorithms with region proposals, crop/warp features, SVM classifications and bounding box regression such as Regions with CNN (R-CNN) were introduced. Although R-CNN was comparatively high in accuracy with the previous inventions, its high usage of space and time later led to the invention of Spatial Pyramid Pooling Network (SPPNet). Despite SPPNet's speed, to reduce the similar drawbacks it shared with R-CNN; Fast R-CNN was introduced. Although Fast R-CNN could reach real-time speeds using very deep networks, it held a computational bottleneck. Later Faster R-CNN, an algorithm based on ResNet, was introduced. Due to Faster RCNN not yet capable of surpassing state of the art detection systems, YOLO was introduced. This paper reviews the dominating real-time object detection algorithm You Only Look Once (YOLO). Consisting of layers in the basic CNN architecture and YOLO networks, each layer's characteristics and the two versions of YOLO; YOLO-V1 and YOLO-V2 would be reviewed under this paper. The strengths and weaknesses of YOLO would be exposed, finally being followed by a summarized conclusion.

## CHAPTER 3

# SYSTEM REQUIREMENT SPECIFICATION

### 3.1 Introduction

- **Existing System**

- The YOLO (You Only Look Once) framework is a popular method for performing real-time object detection. It was first introduced in 2015 and has since been improved and refined in subsequent versions, including YOLOv2, YOLOv3, and YOLOv4.
- The basic idea behind YOLO is to use a single CNN to predict the bounding boxes and class labels for objects in an image. The model is trained on a dataset of labeled images, and during inference, it takes an input image and predicts a set of bounding boxes and class probabilities for the objects present in the image.
- One of the key advantages of YOLO is its efficiency, as it is able to process images and predict object bounding boxes in real-time. This makes it well-suited for use in live object detection applications, such as video surveillance and autonomous vehicles.
- There are several open-source implementations of YOLO available, including Darknet (the original implementation) and PyTorch YOLOv3. These implementations include pre-trained models and tools for training custom models on your own dataset.

- **Proposed System**

- There are several potential ways to improve the existing YOLO-based system for live object detection:
- Incorporate additional data augmentation techniques: By applying a wider range of data augmentation techniques during training, you can improve the robustness of your model and reduce the risk of overfitting.
- Use a more powerful CNN architecture: By using a more powerful CNN architecture, such as a deeper network or one that uses more advanced building blocks (e.g., residual connections), you may be able to improve the accuracy of your object detection system.

- Implement more efficient post-processing: You can improve the efficiency of your object detection system by implementing more efficient post-processing techniques, such as lightweight non-maximum suppression algorithms.
- Incorporate temporal information: In a live object detection system, you can leverage temporal information by incorporating information from previous frames in the video feed. This can help to improve the accuracy of your object detections, particularly for objects that are partially occluded or moving quickly.
- Use domain-specific knowledge: If you are working on a specific application (e.g., detecting pedestrians in a city), you can incorporate domain-specific knowledge into your model to improve its accuracy. This could include using additional training data that is relevant to your specific application or incorporating domain-specific features into your CNN architecture.

### 3.2 System Architecture

- **YOLO – You Only Look Once**

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image.

Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image. More recent approaches like R-CNN use region proposal

- Resize image.
- Run a convolutional network.
- Non-max suppression. Dog: 0.30 Person: 0.64 Horse: 0.28 Figure 1:

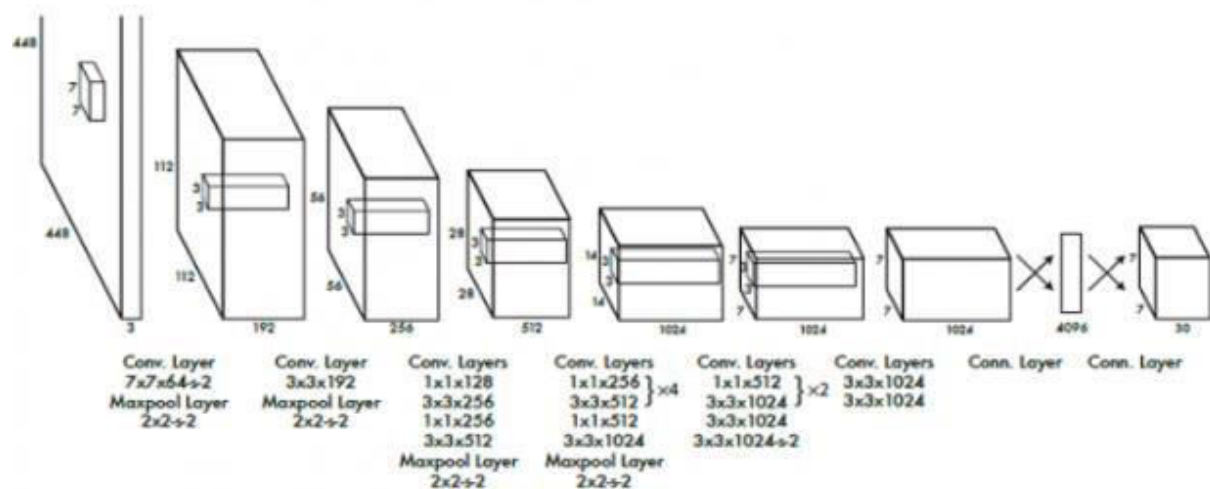


The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system

Re-sizes the input image to  $448 \times 448$ ,

Runs a single convolutional network on the image, and thresholds the resulting detections by the model's confidence. methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes.

After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene. These complex pipelines are slow and hard to optimize because each individual component must be trained separately. We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are. YOLO is refreshingly simple.



**Fig. 3.1 Yolo Predict**

A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object

detection. First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline.

We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems. For a demo of our system running in real-time on a webcam please.

Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

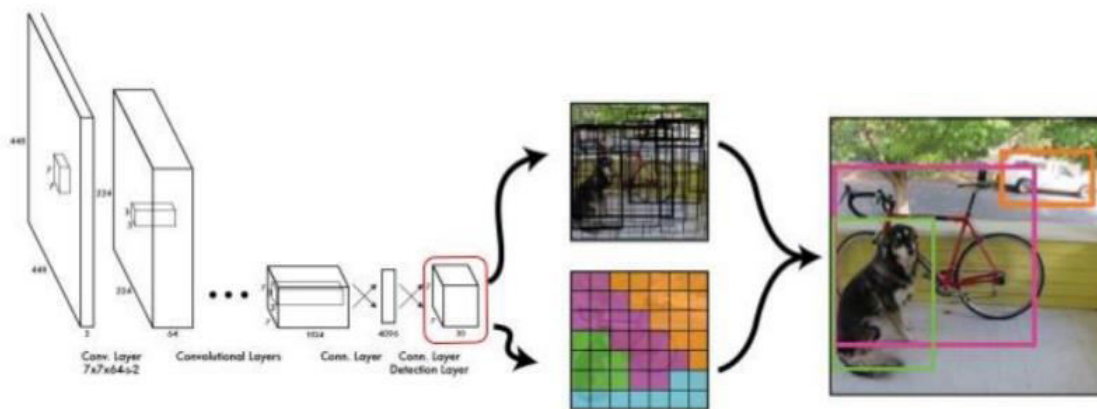
Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs. YOLO still lags behind state-of-the-art detection systems in accuracy. While it can only identify objects in images it struggles to precisely localize some objects, especially small ones. We examine these trade-offs further in our experiments.

### • **Unified Detection**

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision. Our system divides the input image into an  $S \times S$  grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts  $B$  bounding boxes and confidence scores for those boxes.

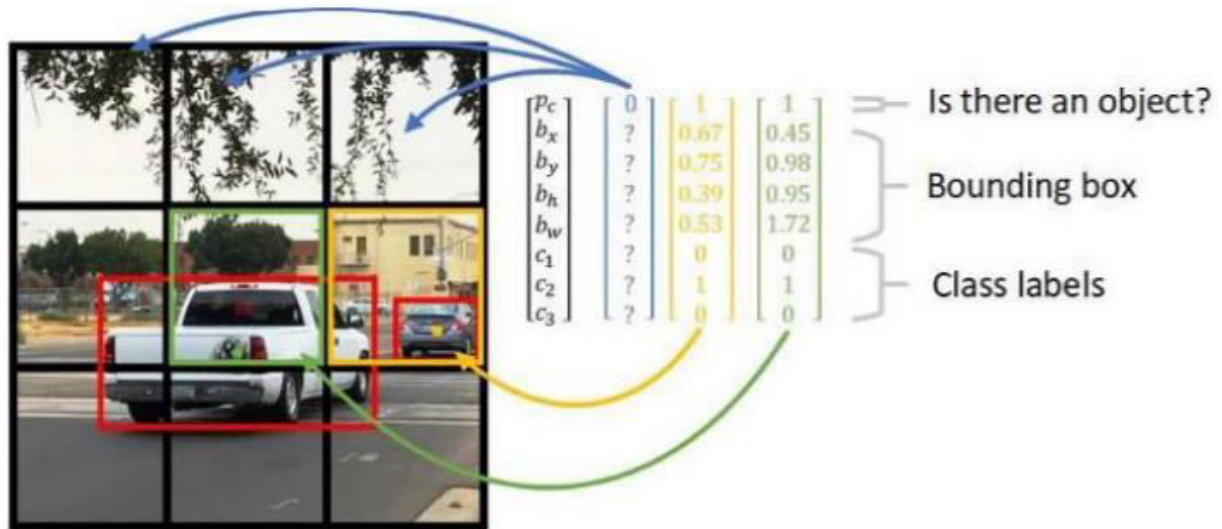
These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as  $\text{Pr}(\text{Object}) * \text{IOU truth pred.}$ . If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

### YOLO: You Only Look Once



**Fig. 3.2 Yolo Framework**

Each bounding box consists of 5 predictions:  $x$ ,  $y$ ,  $w$ ,  $h$ , and confidence. The  $(x, y)$  coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts  $C$  conditional class probabilities. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes  $B$ .



**Fig. 3.3 Unified Detection**

## • Network Design

We implement this model as a convolutional neural network and evaluate it on the PASCAL VOC detection dataset. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. Our network architecture is inspired by the Google-Net model for image classification. The network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by Google-Net, we simply use  $1 \times 1$  reduction layers followed by  $3 \times 3$  convolutional layers, similar to Lin et al. The full network is shown in Figure 3. We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO.

### 3.3 Functional Requirements

**Table No. 3.1 Functional Requirements**

<b>SL No.</b>	<b>Description</b>
FR1	Real-Time Object Detection: The system should be able to perform object detection in real-time, meaning it should process frames at a rate that appears instantaneous to the user.
FR2	Accurate Object Localization: The system should accurately localize objects in the input frames, providing precise bounding box coordinates that tightly enclose the objects of interest.
FR3	Object Classification: The system should classify the detected objects into appropriate classes or categories, providing the predicted class labels for each object.
FR4	Multiple Object Detection: The system should be able to detect and handle multiple objects simultaneously in a single frame, including cases where objects overlap or occlude each other.
FR5	Robustness to Scale and Aspect Ratio: The system should be able to handle objects of varying sizes and aspect ratios, adapting the object detection algorithm to work effectively across different scales.
FR6	Support for Various Object Classes: The system should be capable of detecting and classifying a wide range of object classes, including common objects found in everyday environments.
FR7	Customizable Object Detection Threshold: The system should allow users to customize the detection threshold, enabling them to control the sensitivity of the system and the trade-off between precision and recall.
FR8	Real-Time Visualization: The system should provide a real-time visualization of the detected objects in the input frames, overlaying bounding boxes and class labels to aid in understanding and analysis.
FR9	Compatibility with Different Input Sources: The system should be compatible with various input sources, such as live video feeds from cameras, pre-recorded video files, or image sequences, allowing flexibility in capturing and processing the input data.
FR10	Scalability and Performance: The system should be designed to handle large volumes of data and exhibit efficient performance, ensuring real-time object detection even with high-resolution frames or complex scenes.

### 3.4 Non Functional Requirements

#### 3.4.1 User Interface

#### OpenCV 3.4

**OpenCV** (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

#### 3.4.2 Software Requirements:

- |                   |   |                    |
|-------------------|---|--------------------|
| • Coding Language | : | Python 3           |
| • Environment     | : | Anaconda Framework |
| • Tools           | : | Open-CV            |
| • IDE             | : | PyCharm            |

- **Python**

**Python:** Python is a powerful scripting object-oriented programming language created by Guido Rossum in 1989. It is very useful for solving complex problems involving machine learning algorithms. It has various functions which help in preprocessing. Processing is fast and it is supported on almost all platforms. Integration with C++ and other image libraries is very easy, and it has in-built functions and libraries to store and manipulate data of all types. It provides interfaces to many OS system calls and libraries as per our need. Python has good code readability. Open CV and other libraries are integrated in python interpreter for using readymade optimized functions. Python language is used by us, as main coding language for our proposed system.

- **Open-CV**

Open-CV's application areas include

- 2D and 3D feature toolkits
- Emotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Augmented reality

- **Tensor-Flow**

Tensor-Flow is an open source software library for machine learning written in Python and C++, which is developed by Google Brain Team. Tensor-Flow is used as one of the main libraries for machine learning in our project. Using Tensor-Flow we can improve feature recognition tasks in images. Tensor-Flow corrected many issues found on other frameworks.

As a consequence, TF has achieved state of the art performance without compromising code readability. Moreover, Tensor-Flow gives flexibility to define different operations, especially neural network topologies.

- **Keras**

Keras is one of the most powerful and easy to use Python libraries for developing and evaluating deep learning models. The advantage of using Keras is mainly that, we can build neural networks in an easy way and efficient numerical computation is done on this developed neural network to achieve particular task.

There are two main types of models available in Keras:

- 1) Sequential model.
- 2) Functional model

The sequential Application programming Interface (API) allows us to create models layer by layer for most complex problems. It does not allow us to create models that share layers or have multiple inputs or output.

- **Pycharm**

PyCharm is an integrated development environment (IDE) specifically designed for Python development. It is developed by JetBrains and provides a comprehensive set of tools and features to streamline the coding process and enhance productivity. Here are some key aspects and features of PyCharm:

- **Code Editor:** PyCharm offers a powerful code editor with syntax highlighting, code completion, and code navigation features. It supports various versions of Python and provides intelligent code analysis to detect errors and offer suggestions for improvement.
- **Project Management:** PyCharm helps in managing Python projects efficiently. It allows you to create, open, and organize projects, keeping all the related files and resources organized in a structured manner. It provides project templates, version control system integration, and virtual environment support.
- **Debugging:** PyCharm includes a robust debugger that helps in identifying and fixing issues in Python code. It allows setting breakpoints, stepping through the code, inspecting variables, and monitoring program execution. It also supports remote debugging for debugging code running on remote machines or servers.
- **Testing and Profiling:** PyCharm integrates with popular testing frameworks like pytest and unit test, enabling developers to write and run tests within the IDE. It provides a test runner and offers tools for test result analysis. Additionally, it supports code profiling to identify performance bottlenecks and optimize code execution.
- **Intelligent Code Refactoring:** PyCharm includes a range of automated code refactoring



tools to improve code structure and maintainability. It offers features like code formatting, renaming variables and functions, extracting methods, and more. These tools help in reducing code duplication, improving readability, and ensuring adherence to coding standards.

- **Version Control Integration:** PyCharm integrates with version control systems like Git, Mercurial, and Subversion. It provides a seamless interface for managing version control operations within the IDE, including committing changes, viewing diffs, and resolving conflicts.
- **Built-in Terminal and Database Support:** PyCharm includes an integrated terminal for executing command-line tasks without leaving the IDE. It also offers database integration, allowing developers to connect to databases, execute SQL queries, and manage database schemas.
- **Extensibility and Plugin Ecosystem:** PyCharm supports the installation of third-party plugins, enabling developers to extend its functionality and customize their development environment. There is a rich ecosystem of plugins available, offering additional features and integrations.
- **Anaconda:**

Anaconda is a popular open-source distribution platform and management system for scientific computing and data science tasks. It provides a comprehensive environment for Python and R programming languages, along with a collection of pre-installed libraries and tools specifically curated for data analysis, machine learning, and scientific computing. Here are the key aspects of Anaconda:

- **Conda Package Manager:** Anaconda utilizes the Conda package manager, which allows users to easily install, update, and manage software packages and dependencies. Conda simplifies package management by resolving and handling complex dependency chains, ensuring compatibility across different packages and environments.

- **Anaconda Navigator:** Anaconda Navigator is a graphical user interface (GUI) that provides an intuitive way to manage and navigate through Anaconda environments. It allows users to create, manage, and switch between different environments, install packages, launch applications, and access documentation and tutorials.
- **Environments:** Anaconda allows you to create isolated environments, which are self-contained environments with their own set of packages and dependencies. Environments enable users to work on different projects with different package requirements without conflicts. They can be easily created, activated, and deactivated using the Conda command-line interface or the Anaconda Navigator.
- **Pre-installed Libraries:** Anaconda comes with a wide range of pre-installed libraries and tools that are commonly used in data science and scientific computing, such as NumPy, Pandas, Matplotlib, SciPy, Scikit-learn, Jupyter Notebook, and many more. These pre-installed libraries provide a foundation for data analysis, visualization, machine learning, and other scientific computing tasks.
- **Jupyter Notebook Integration:** Anaconda seamlessly integrates with Jupyter Notebook, a web-based interactive coding environment. Jupyter Notebook allows users to create and share documents that contain live code, visualizations, and explanatory text. It supports multiple programming languages, including Python, R, and Julia, making it a versatile tool for data exploration, prototyping, and analysis.
- **Platform and OS Support:** Anaconda is available for multiple platforms, including Windows, macOS, and Linux, ensuring cross-platform compatibility. It provides consistent package management and environment management across different operating systems.
- **Extensive Package Ecosystem:** Anaconda offers a vast ecosystem of additional packages and libraries that can be easily installed and managed using Conda. It provides access to thousands of community-contributed packages, allowing users to leverage a wide range of tools and functionalities for their specific data science and scientific computing needs.

### 3.4.3 Hardware Requirements:

- Operating system : Windows 8 / 10 (64 bit)
- System Processor : i5 / i7
- Hard Disk : 500 GB.
- Ram : 8 GB / 12 GB.
- Any desktop / Laptop system with above configuration or higher level.

### 3.5 System model

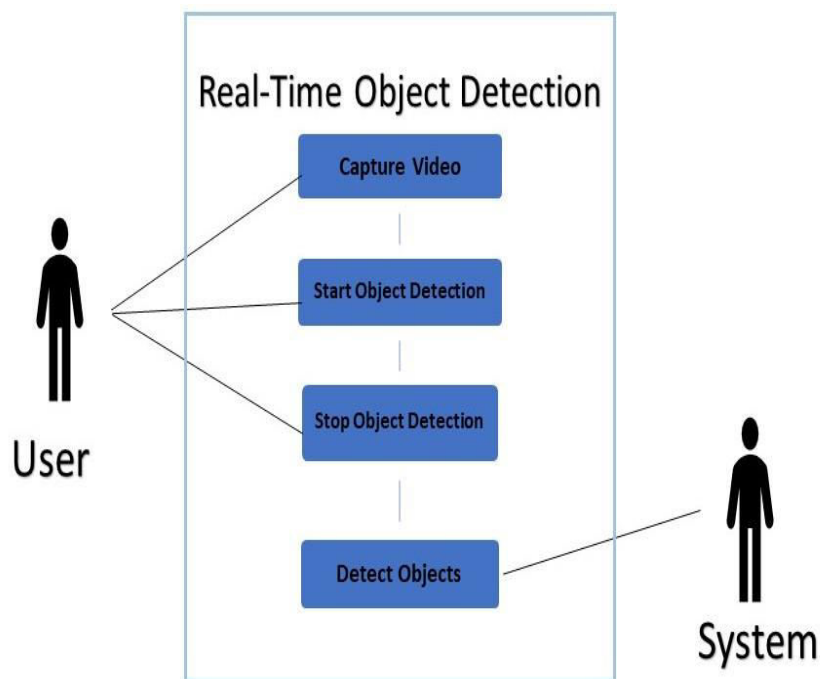
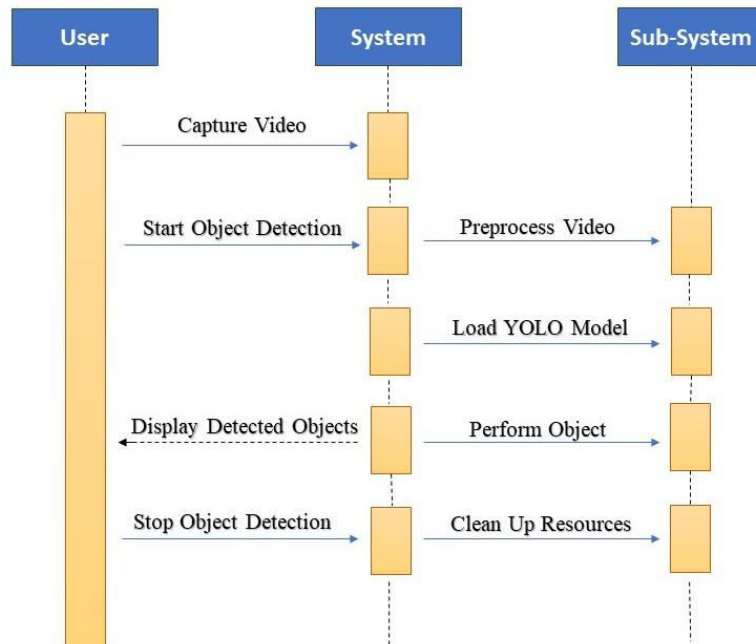


Fig 3.4 Use case for Real-time Object Detection



**Fig 3.5 Dataflow representation for Real-time Object Detection**

## **CHAPTER 4**

### **SYSTEM DESIGN**

#### **4.1 System Architecture and Decomposition**

The real-time object detection system using the YOLO (You Only Look Once) framework typically consists of the following components:

- **Input Stream:** The system takes an input stream, which can be a live video feed from a camera or a pre-recorded video file.
- **Preprocessing:** The input stream is preprocessed to ensure compatibility with the YOLO framework. This involves resizing the frames to a fixed size and normalizing the pixel values.
- **YOLO Model:** The core component of the system is the YOLO model, which is responsible for detecting objects in the input frames. YOLO utilizes a deep neural network architecture, usually based on convolutional neural networks (CNNs), to perform object detection. The model is trained on a large dataset with annotated images to learn to identify different object classes.
- **Object Detection:** The YOLO model performs object detection on each frame of the input stream. It divides the frame into a grid of cells and predicts bounding boxes and class probabilities for objects within each cell. The model outputs the coordinates of the bounding boxes, the associated class labels, and the confidence scores for each detected object.
- **Post-processing:** The output of the YOLO model goes through post-processing steps to refine the detections. This may involve applying non-maximum suppression to remove overlapping bounding boxes with lower confidence scores. The post-processing ensures that only the most relevant and accurate detections are retained.
- **Visualization/Output:** The final step involves visualizing the detected objects in the input frames. This can include drawing bounding boxes around the objects and labeling them with their respective class names. The system can display the annotated frames in real-time or save them for later analysis.

The real-time object detection system using the YOLO framework can be decomposed into several subtasks, each with its own responsibilities. Here's a decomposition of the system:

- **Input Management:** This component is responsible for handling the input stream, whether it's a live video feed or a pre-recorded video file. It handles tasks such as accessing the frames, managing frame buffering, and providing frames for processing.
- **Preprocessing:** This component takes the input frames and performs necessary preprocessing steps. It resizes the frames to a fixed size required by the YOLO model and normalizes the pixel values to a suitable range.
- **YOLO Model:** This component consists of the deep neural network architecture that performs object detection. It takes preprocessed frames as input and processes them through the network to generate predictions. The YOLO model may be implemented using a pre-trained model that is fine-tuned for specific object detection tasks.
- **Object Detection:** This component takes the output from the YOLO model and processes it to extract relevant information. It includes tasks such as decoding the predicted bounding box coordinates, applying class probabilities, and filtering out low-confidence detections.
- **Post-processing:** This component refines the output of the object detection step. It applies techniques like non-maximum suppression to remove overlapping bounding boxes and retain only the most confident detections. Post-processing ensures that the final detections are accurate and non-redundant.
- **Visualization/Output:** This component is responsible for visualizing the detections in the input frames. It draws bounding boxes around the detected objects and labels them with their respective class names. It may also handle tasks such as displaying the annotated frames in real-time or saving them for further analysis.

Each of these components can be further decomposed into specific modules or functions depending on the complexity of the system and the requirements of the application. The decomposition allows for modular design, easier maintenance, and flexibility in incorporating improvements or modifications to individual components.

### 4.2 Data Models

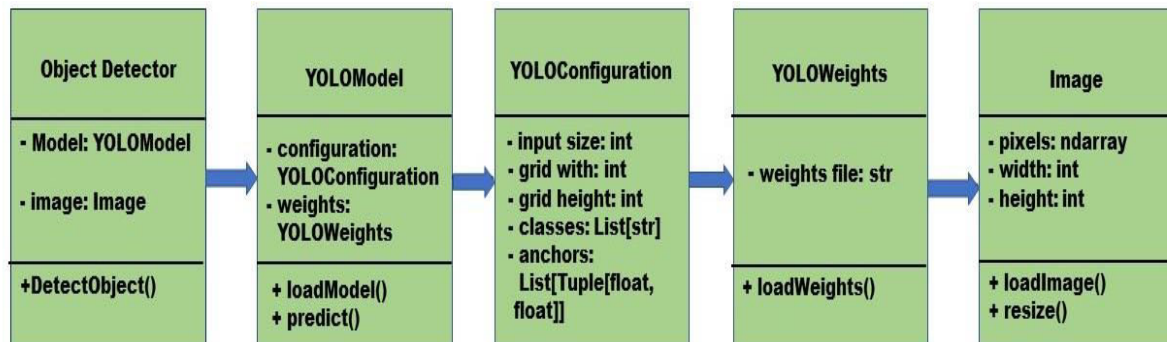


Fig 4.1 Class Model for YOLO

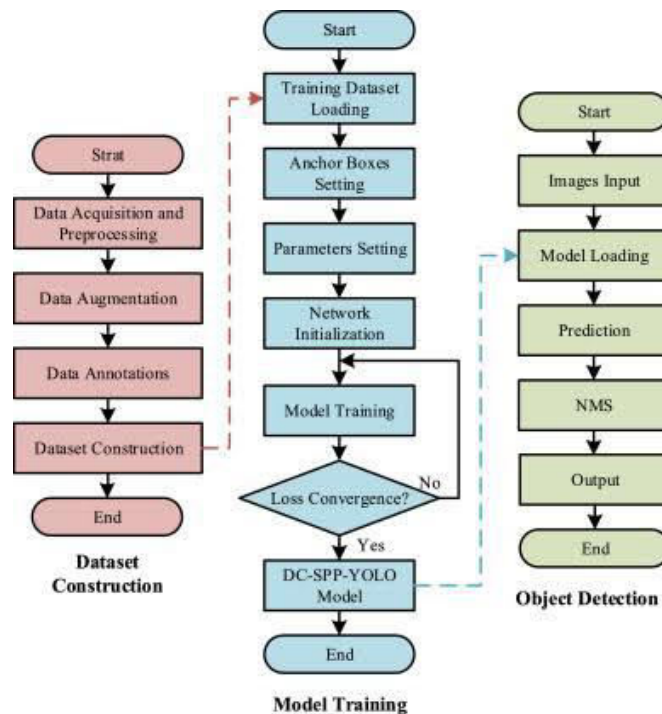


Fig 4.2 Dataflow representation for YOLO

### 4.3 Algorithmic Design & User Interface Design

#### CNN: Convolutional Neural Network

Convolutional neural networks are DL algorithms that are capable of feature detection on images that are given as input. CNNs can attribute weights to various features of an image to distinguish each one. CNNs work with lesser amounts of pre-processing. CNNs can learn filters and features by themselves, unlike manual teaching required for basic algorithms. This results in lesser pre-processing being needed.

- Collect and label a dataset of images containing the objects you want to detect.
- Train a convolutional neural network (CNN) on this dataset using the YOLO framework. This will allow the model to learn to recognize the objects in the images.
- Set up a camera or video feed as the input for your object detection system.
- Preprocess the video feed frames to prepare them for input into the CNN. This may include resizing, cropping, and normalizing the images.
- Use the trained CNN to predict the bounding boxes and class labels for the objects in each frame of the video feed.
- Overlay the bounding boxes and class labels on the original video feed frames to visualize the object detections.

You may also want to consider implementing some form of post-processing, such as non-maximum suppression, to improve the accuracy and efficiency of your object detection system.

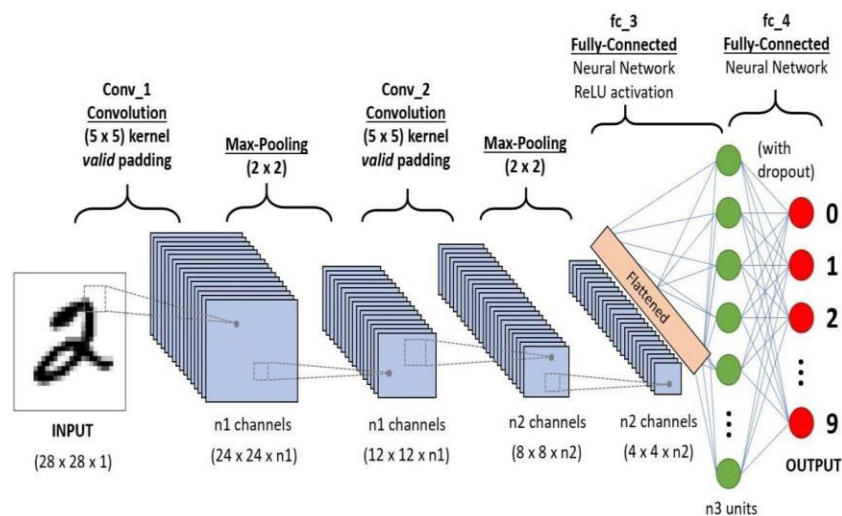


Fig. 4.3 Structure of a typical CNN



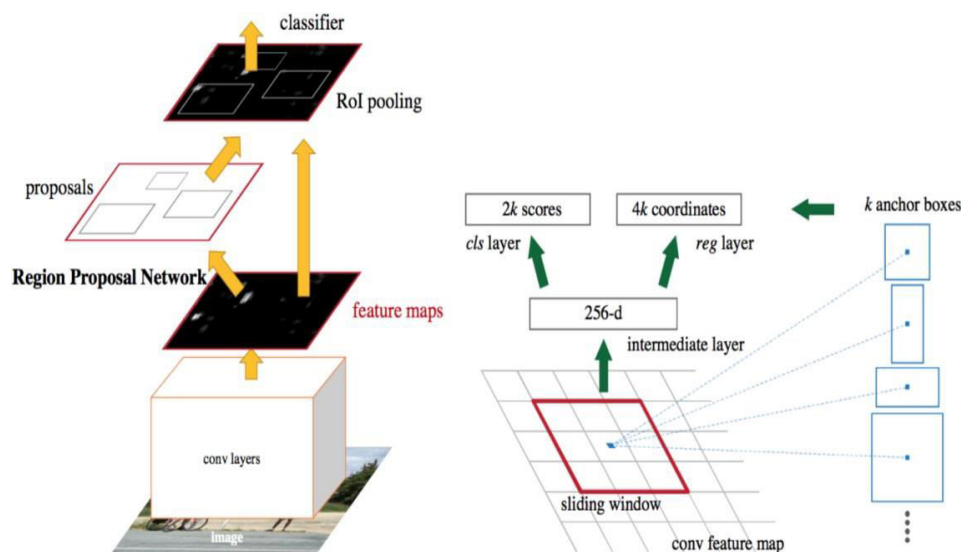
Since CNNs are basically neural networks, the sequencing is influenced by the structure of neurons found in the visual cortex of the brain. Response of unique neurons is conditional to a receptive field within which they operate and are triggered. These neurons are found as bundles, which form the entire area of the visual cortex.

Usually CNNs consist of

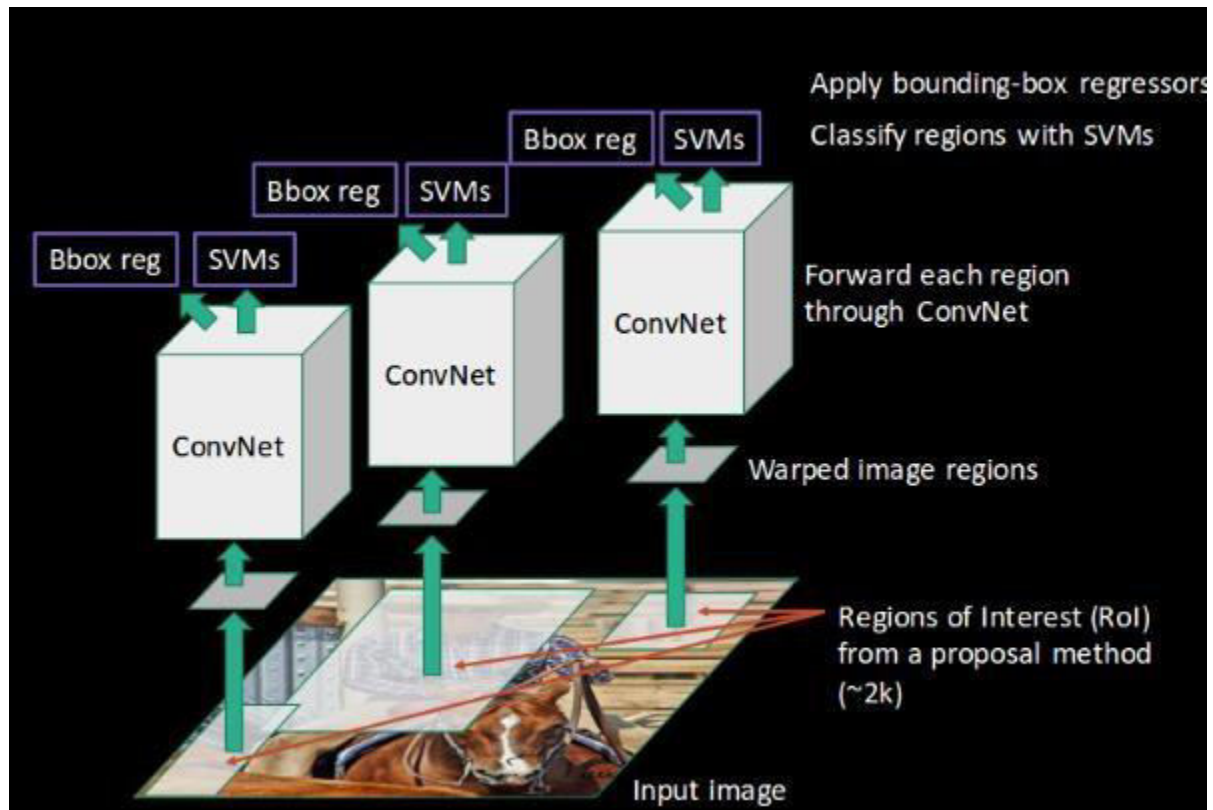
- i. Convolutional layers – the number of layers is dependent upon how deep the network is.
- ii. Activation layers – common activation layers such as ReLU, Leaky ReLU are used to make the math intact and prevent attaining a practically impossible outcome or random excitations.
- iii. Pooling layers – tensor size is reduced by the use of these.

### R-CNN

R-CNN stands for Region Specific Convolutional Neural Network. It uses a methodology known as Selective Search to look for regions with objects in an image. Selective search looks for patterns in an image and classifies objects based on that. Patterns are understood by the network by monitoring varying scales, colors, textures and enclosures. Initially the network takes in an image and selective search generates sub-segmentations based on these patterns. Similar regions are then associated to produce bigger regions



**Fig. 4.4 Image Processes involved in R-CNN**



**Fig. 4.5 Image data flow in R-CNN**

### Limitations of R-CNN

Training an R-CNN model is expensive and slow due to the following aspects of the network,

- Extracting 2000 regions particular to an image on the basis of selective searching
- Feature extraction for each image would become  $N \times 2000$  If  $N$  is the average number of features for each image or rather  $N$  should be the number of features extraction methods to be learnt by the network.

The entire process of object detection has 3 models wrapped in it:

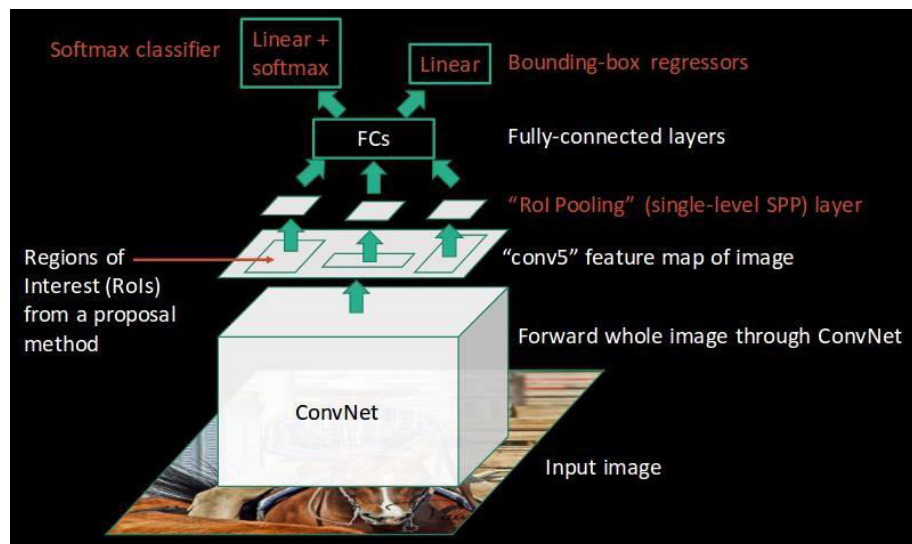
- i. CNNs for learning features to be extracted from the image
- ii. SVM classifier for object classification
- iii. Regressor model for narrowing down bounding boxes

The complexity of the processes involved in object detection renders R-CNN really slow and also computationally expensive to train models. It typically takes 40 – 50 seconds to perform the object detection.

### FAST R-CNN

This is an improvisation of R-CNN to make it faster. Instead of running the CNNs 2000 times per image, for all the 2000 regions, the CNN was run one time to generate a feature map for all the 2000 regions. Using these convolutional region maps, different regions of consideration are extracted and a ROI pool layer is used to remake the extracted region-specific feature maps and it is fed into the Fully connected layer.

Instead of 3 models like R-CNN, Fast R-CNN uses only one model which performs extraction of characteristics from different regions, after which division into classes is done and then it returns the bounding boxes for distinguished classes simultaneously.



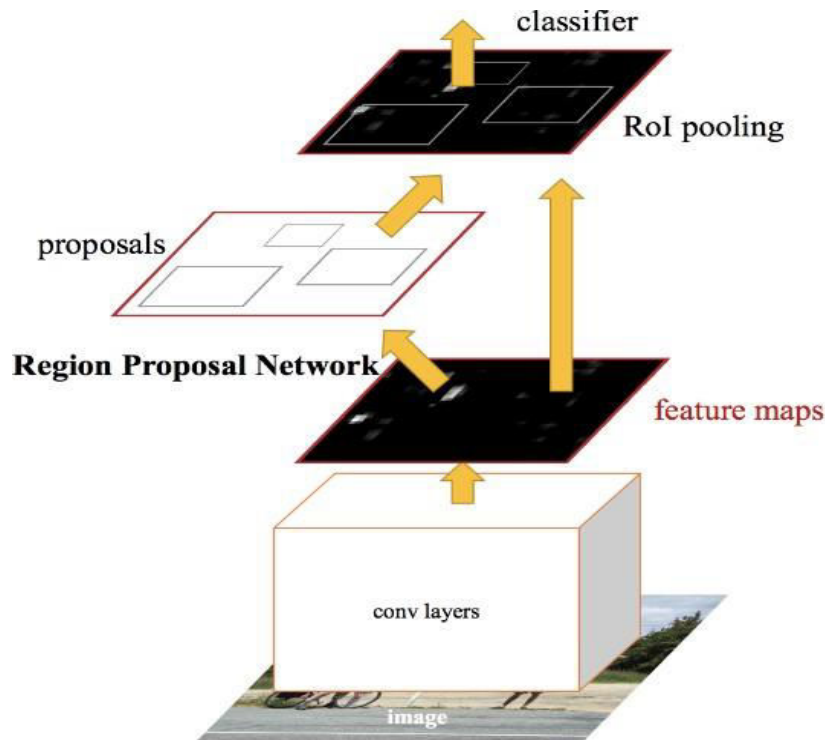
**Fig. 4.6 Image data flow in Fast R-CNN**

### Limitations of Fast R-CNN

Even though it's faster than R-CNN, it uses region proposal methods like selective search which makes the process of detecting objects slow, in other words not fast enough to stand as a robust Object detection network. Fast R-CNN still takes about 2 seconds to make predictions about the objects in the image.

### FASTER R-CNN

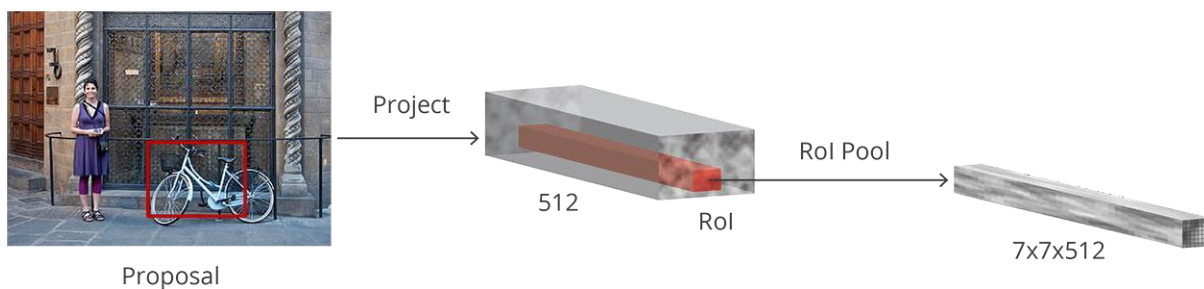
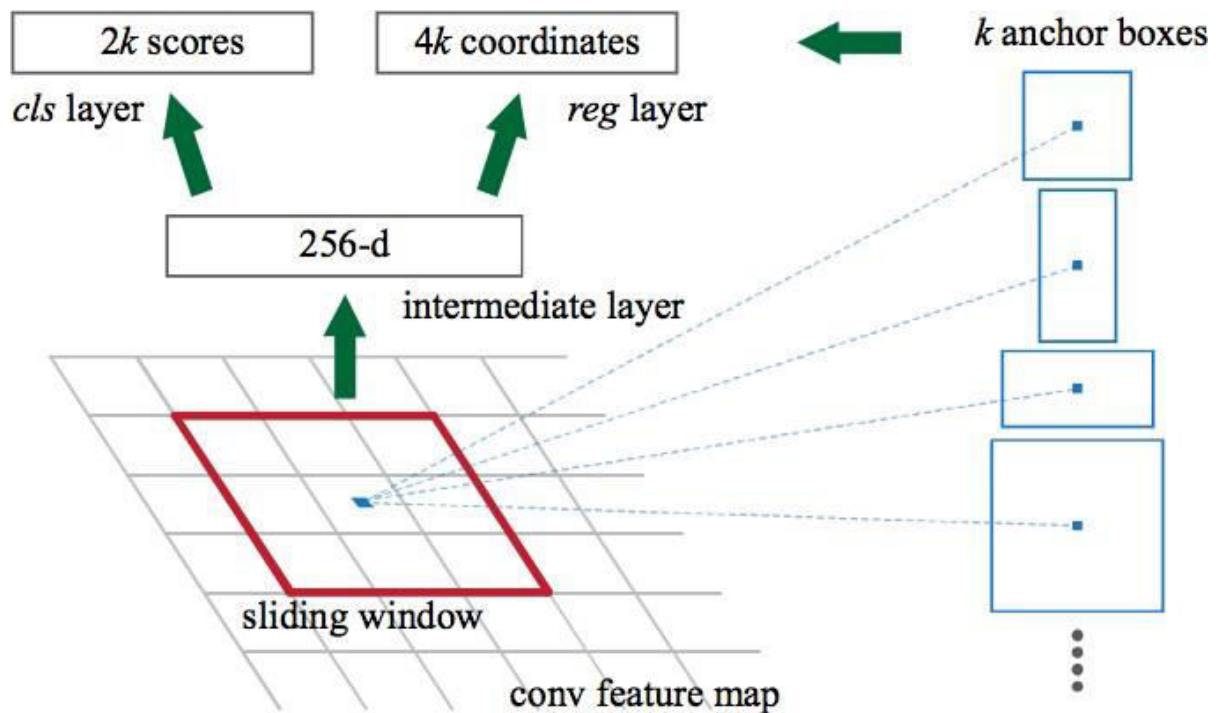
Faster R-CNN is the modified version of Fast R-CNN, where generation of ROIs is done using Region Proposal Network. This algorithm acquires input in the form of feature maps. The result is obtained as object-ness scores.



**Fig. 4.7 Image data flow in Faster R-CNN**

- The Faster R-CNN takes in an input image, passes it to the CNN and allows it to generate an image's feature map.
  - RPN is utilized on those maps, returning the object suggestion and the object-ness score.
- The RPN uses sliding windows over these object maps and 'k' Anchor boxes of different dimensions are generated. For each anchor box, two things are predicted,
- The probability of the anchor holding objects (it doesn't classify the objects at this step)
  - A bounding box regressor to better adjust the anchor to fit the objects better inside the box.

- Next, the proposals are taken in, cropped so that each proposal contains an object. An ROI pool layer is applied to all these suggestions to bring them down to same size. ROI pooling layers extract a fixed size pooling map for each anchor.
- Ultimately, the resized proposals pass to the fully connected layer containing the 'SoftMax' layer. Linear regression layer is present on the top, to classify and output the bounding boxes.



**Fig. 4.8 ROI Pooling Layer Working in Faster R-CNN**

### Limitations of Faster R-CNN

All of R-CNN's versions (including Faster R-CNN) look at different regions of a particular image sequentially as opposed to taking a look at the whole image. The result of this are two complications:

- Several passes are required through an image to extract all objects.
- The performance of all the systems in the network depends upon the performance of the previous systems, thereby if an error occurs due to improper generation of feature maps, it messes up the detection process after that.

### YOLO: YOU ONLY LOOK ONCE

This is a system which is focused on real-time determination of several objects and instances. This works in a similar structure to that of a FCNN algorithm, unlike RPNs where same regions in an image are processed several times, where the image gets passed once and the output is obtained. [input-  $N \times N$ ; output-  $S \times S$ ]. The image is divided into a grid of size  $S \times S$ . Every grid is assigned to predict a unique object. A certain number of boundary boxes is assigned to all grids. The drawback of YOLO is that since each grid can detect only a unique object, the proximity of the objects determines whether they are detected or not.

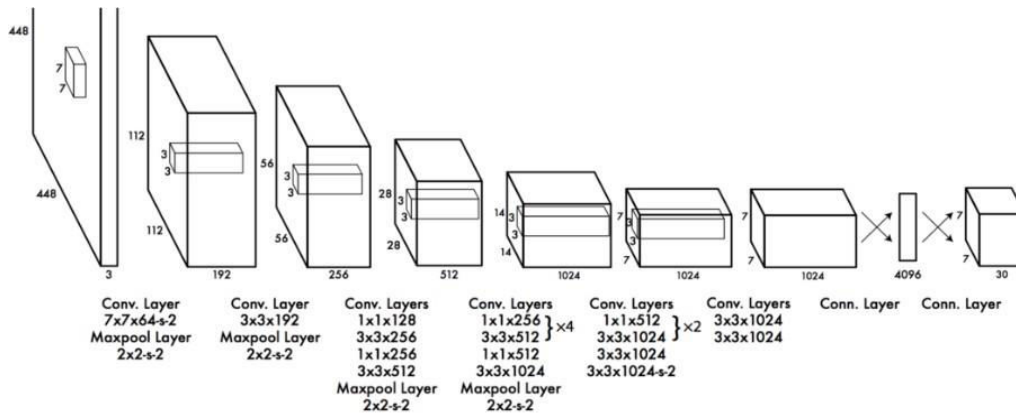
For every individual cell in the grid,

- '**B**' number of boundary boxes are predicted. every box has a **box confidence score**,
- Regardless of the no. of B, only one object can be detected
- '**C**' **conditional class probabilities** are predicted.

Grid size:  $7 \times 7$  [ $S \times S$ ]; Boundary boxes: 2; No. of classes: 20

Contents of a boundary box:  $x, y, w, h$ ; box confidence score. The "objectness" is reflected based on the confidence score. This shows the likeliness that a box encompasses the object and the accuracy of the box. The dimensions of the bounding box ( $w, h$ ) are standardized with those of the image and  $x, y$  are cell offset values (note that  $x, y, w, h$  lie within 0 – 1). Conditional class probability: 20 per cell; probability of an object pertaining to a certain class.

Prediction shape for YOLO: (S, S, (Bx5) +C) = (7, 7, (2x5) +20) = (7, 7,30).



Source

**Fig. 4.9 Structure of YOLO**

YOLO is set on the idea of building a CNN to predict a tensor of size 7,7,30. The spatial dimension reduces to 7x7, with every location having 1024 o/p channels. Two FC layers are used to make 7x7x2 boundary box on which linear regression is performed. Those with confidence scores above 0.25 are used to make predictions finally.

class confidence score is calculated as (per pred.box):

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

Classification and localization, both are needed to calculate confidence.

$$\begin{aligned}\text{box confidence score} &\equiv P_r(\text{object}) \cdot \text{IoU} \\ \text{conditional class probability} &\equiv P_r(\text{class}_i | \text{object}) \\ \text{class confidence score} &\equiv P_r(\text{class}_i) \cdot \text{IoU} \\ &= \text{box confidence score} \times \text{conditional class probability}\end{aligned}$$

where

$P_r(\text{object})$  is the probability the box contains an object.

$\text{IoU}$  is the IoU (intersection over union) between the predicted box and the ground truth.

$P_r(\text{class}_i | \text{object})$  is the probability the object belongs to  $\text{class}_i$  given an object is presence.

$P_r(\text{class}_i)$  is the probability the object belongs to  $\text{class}_i$



### 4.4 Design of the network

YOLO consists of: 24 convolutional layers, 2 FC layers. In certain layers, 1x1 reduction layers are utilized to reduce depth of feature maps. Tensor shape of the last layer is (7,7,1024), and this tensor is flattened. 2 FC layers are used as linear regressions to give a 7x7x30 o/p, which is reshaped to (7,7,30) (2 bound. box predictions at each location)

Fast YOLO: 9 convolutional layers; more shallow feature maps

### Loss function

YOLO works by predicting several bounding boxes for every cell. This would result in the generation of false positives. To counter this, we require only one bounding box to correspond to the object. The largest IoU is chosen with ground truth, for this reason, thus making every prediction more effective at guessing specific sizes, aspect ratios, etc.

Sum squared error is utilized between predictions and ground truth as a means to compute loss.

Loss function comprises of: classification, localization and confidence losses.

### Classification loss

It is the square of error of class cond. probabilities per class:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

where

$\mathbb{1}_i^{\text{obj}} = 1$  if an object appears in cell  $i$ , otherwise 0.

$\hat{p}_i(c)$  denotes the conditional class probability for class  $c$  in cell  $i$ .

### Localization loss

Error in predicted bound. box locations and dimensions is measured, where the box corresponding to the object is only counted.



$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

where

$\mathbb{1}_{ij}^{\text{obj}} = 1$  if the  $j$ th boundary box in cell  $i$  is responsible for detecting the object, otherwise 0.

$\lambda_{\text{coord}}$  increase the weight for the loss in the boundary box coordinates.

Different weights should be given to boxes of different sizes, since it may be the same for a small or a large box, if say a 3-pixel error occurs. This is partly compensated since YOLO uses sq. root of width and height in the place of the absolute values. Loss multiplied by  $\lambda_{\text{coord}}$  emphasizes the bound. box accuracy further (5 is the default).

### Confidence loss

Measurement of objectness of box (if object has been detected)

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2$$

where

$\hat{C}_i$  is the box confidence score of the box  $j$  in cell  $i$ .

$\mathbb{1}_{ij}^{\text{obj}} = 1$  if the  $j$ th boundary box in cell  $i$  is responsible for detecting the object, otherwise 0.

Measurement of object-ness of box (if object not detected)

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

where

$\mathbb{1}_{ij}^{\text{noobj}}$  is the complement of  $\mathbb{1}_{ij}^{\text{obj}}$ .

$\hat{C}_i$  is the box confidence score of the box  $j$  in cell  $i$ .

$\lambda_{\text{noobj}}$  weights down the loss when detecting background.

Class imbalances are created since most boxes don't contain objects. Thus, the model is trained such that the background is detected more often compared to the objects.  $\lambda_{\text{noobj}}$  is used as a factor to weigh down loss, to counter this (0.5 is the standard)

### **Conglomerated Loss Function**

The final loss sums up the others

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

### **YOLOv3**

YOLOv2 used a custom CNN known as Darknet-19 with 19 layers from the original network and 11 additional layers to make it a 30 layers' network for object detection. However ever with a 30 layers' architecture, YOLOv2 struggled with small objects detection and this was attributed to loss of fine-grained features as the input passed through each pooling layer. Identity mapping and concatenating characteristics were utilized from preceding layers to determine low lever features, in order to compensate for this.

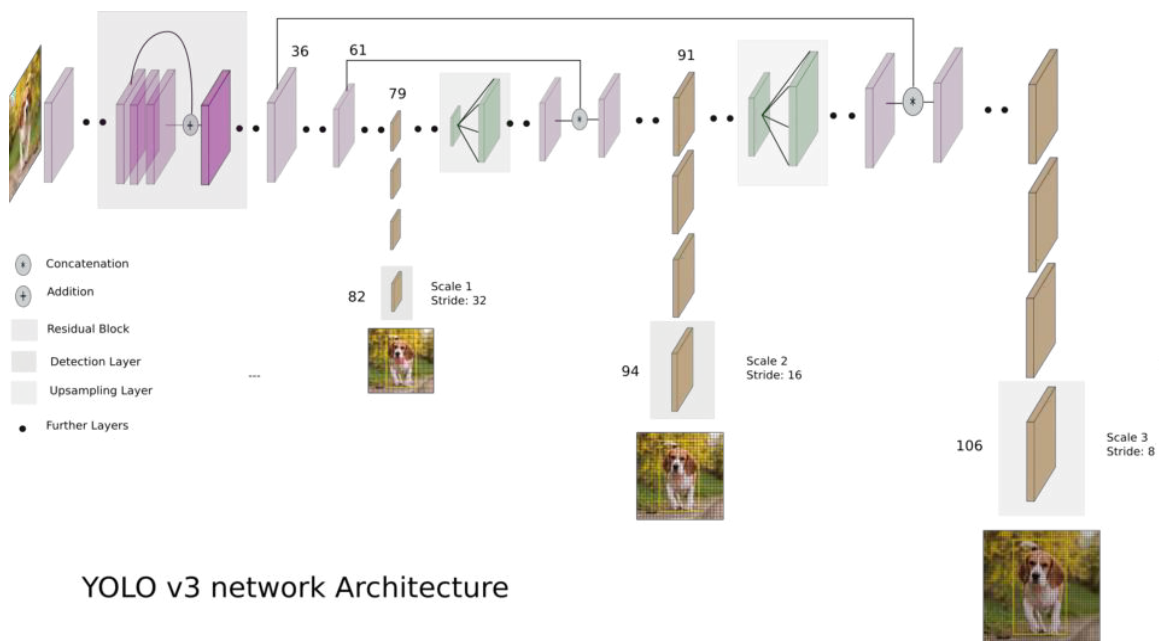
Ever after all this, it lacked several important aspects of a object detection algorithm which made it stable such as residual blocks, skip connections and up sampling layers.

These corrections were made and a new version of YOLO was born and that is known as YOLOv3.

	Type	Filters	Size	Output
	Convolutional	32	$3 \times 3$	$256 \times 256$
	Convolutional	64	$3 \times 3 / 2$	$128 \times 128$
1x	Convolutional	32	$1 \times 1$	
	Convolutional	64	$3 \times 3$	
	Residual			$128 \times 128$
	Convolutional	128	$3 \times 3 / 2$	$64 \times 64$
2x	Convolutional	64	$1 \times 1$	
	Convolutional	128	$3 \times 3$	
	Residual			$64 \times 64$
	Convolutional	256	$3 \times 3 / 2$	$32 \times 32$
8x	Convolutional	128	$1 \times 1$	
	Convolutional	256	$3 \times 3$	
	Residual			$32 \times 32$
	Convolutional	512	$3 \times 3 / 2$	$16 \times 16$
8x	Convolutional	256	$1 \times 1$	
	Convolutional	512	$3 \times 3$	
	Residual			$16 \times 16$
	Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$
4x	Convolutional	512	$1 \times 1$	
	Convolutional	1024	$3 \times 3$	
	Residual			$8 \times 8$
	Avgpool		Global	
	Connected		1000	
	Softmax			

**Fig. 4.10 YOLO Configuration**

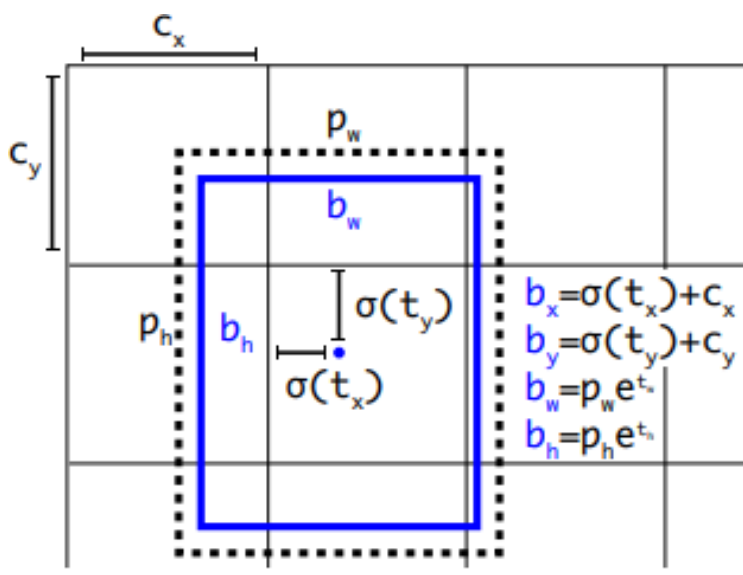
YOLOv3 also uses a variant of Darknet-53 which has 53 convolutional layers trained on Imagenet for the purpose of classification with an additional of 53 more layers stacked onto it to make it a full-fledged network to perform classification and detection. As a result, YOLOv3 is slower than the second version but a lot more accurate than its predecessors.



**Fig. 4.11 Structure of YOLOv3**

The main difference between YOLOv3 and its predecessors is that it makes predictions at 3 different scales. The initial detection is performed in the 82nd layer. If an input image of 416x416 is fed into the network, the feature map thus acquired would be of size 13x13. The other two scales at which detections happen are at the 94th layer yielding a feature map of dimensions 26x26x255 and the final detection happens at the 106th layer, resulting in a feature map with dimension 52x52x255. The detection which happens at the 82nd layer, is liable for the detection of large objects and the detections which happen at the 106th layer is liable for detecting small objects with the 94th layer, staying in-between these 2 with a dimension of 26x26, detecting medium size objects.

This kind of varied detection scale renders YOLOv3 good at detecting small objects than its predecessors.



**Fig. 4.12 Anchor Box Detection**

YOLOv3 uses 9 anchor boxes to localize objects with 3 for each detection scale. The anchor boxes are assigned in the descending order with the largest 3 boxes of all for first detection layer which is used to detect large objects, the next 3 for the medium sized objects detection layer and the final 3 for the small objects' detection layer.

YOLOv3 uses 10x the number of bounding boxes used by YOLOv2 since YOLOv3 detects at 3 different scales. For instance, for an image of input size 416x416, YOLOv2 would predict

$13 \times 13 \times 5 = 845$  boxes whereas YOLOv3 would go for  $13 \times 13 \times 5 + 26 \times 26 \times 5 + 52 \times 52 \times 5 = A$  whopping 10,647 boxes.

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

The loss function of YOLOv3 was modified

This is the loss function of YOLOv2 where the last 3 terms in this image correspond to the function which penalizes for the objectness score predicted by the model for the bounding boxes responsible for predicting objects. The second last term is responsible for bounding boxes having no objects and the final one penalizes the model for the class prediction score for the bounding box which predicts objects.

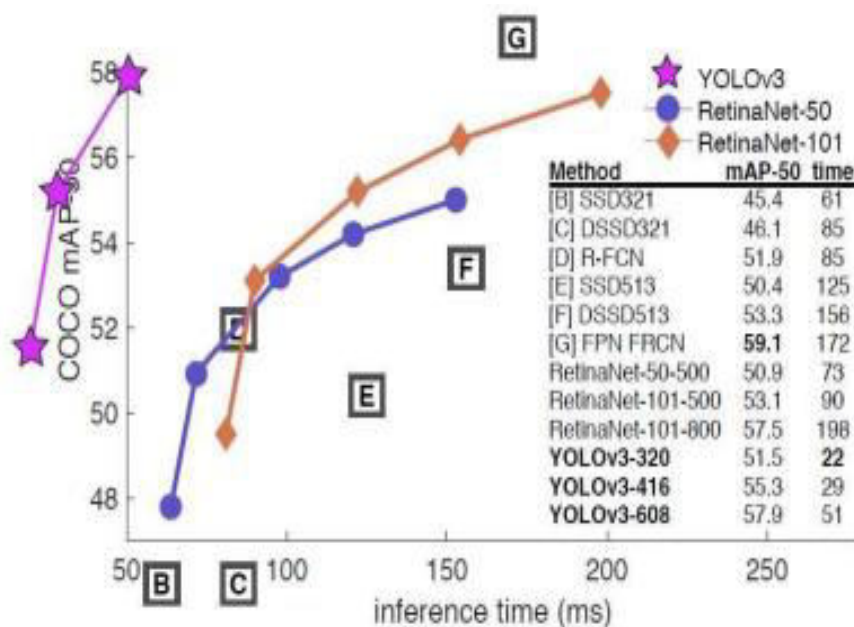
The terms involved in the calculation of loss function in YOLOv2 were calculated using Mean Squared Error method while in YOLOv3, it was modified to Logistic Regression since this model offer a better fit than the previous one.

YOLOv3 executes classification of multiple labels for objects that are detected in images and videos. In YOLOv2, softmaxing is performed on all the class scores and whichever has the maximum class score is assigned to that object. This rests on the assumption that if one object belongs to one class, it can't be a part of another class. For instance, it is not necessary that an object belonging to the class Car would not belong to the class Vehicle. An alternative approach to this would be using logistic regression to predict class scores of objects and setting a threshold

for predicting multiple labels. Classes that have scores greater than the threshold score are then assigned to the box

YOLOv3 was benchmarked against popular state of the art detectors like RetinaNet50 and RetinaNet101 with the COCO mAP 50 benchmark where 50 stands for the accuracy of the model, how well the predicted bounding boxes align with the Ground Truth bounding boxes. This metric of evaluating CNNs is known as IOU, Intersection Over Union. 50 over here corresponds to 0.5 on the IOU scale of the evaluation. If the prediction made by the model is less than 0.5, it is classified as a mislocalisation and classified as a false positive.

YOLOv3 is really fast and accurate. When measured at 50 mAP, it is on par with the RetinaNet50 and RetinaNet101 but it is almost 4x faster than those two models. In benchmarks where the accuracy metric is higher (COCO 75), the boxes need to be more aligned with the Ground Truth label boxes and here is where RetinaNet zooms past YOLO in terms of accuracy.



**Fig. 4.13 Benchmark scores of YOLOv3 and other networks against COCO mAP 50**

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<i>Two-stage methods</i>							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	<b>52.1</b>
<i>One-stage methods</i>							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	<b>40.8</b>	<b>61.1</b>	<b>44.1</b>	<b>24.1</b>	<b>44.2</b>	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

**Fig. 4.14 Benchmark scores of object detection networks against COCO Dataset**

These are the metrics for different models with different benchmarks and it is quite observable that the mAP (mean Average Precision) for YOLOv3 is 57.9 on COCO 50 benchmark and 34.4 on COCO 75 benchmark. RetinaNet is better at detecting small objects but YOLOv3 is so much faster than versions of RetinaNet.

YOLO uses a technique known as Non-Maximal Suppression (NMS) to eliminate duplicates of the same object being detected twice or more than that. It essentially retains on the bounding box with the highest confidence score. The initial step is to discard all the bounding boxes which have a confidence score lesser than the input of the threshold set for detected objects. If the threshold is set to 0.55, it retains bounding boxes with confidence scores more than or equal to 0.55

YOLO also uses an Intersection Over Union metric to grade the algorithm's accuracy. IOU is a simple ratio of area of intersection between the predicted box and the Ground truth box to the Area of the Union of the predicted box with ground truth box. After removing bounding boxes with the detection probability lesser than the NMS threshold, YOLO discards all the boxes for objects with IOU scores lesser than the IOU threshold to eliminate duplicate detections further.



## CHAPTER 5

# IMPLEMENTATION

### 5.1 IMPLEMENTATION OF YOLOv8

To implement the pre-trained YOLOv8 network, all that is required from the library is the config file of YOLOv8 which defines the layers and other essential specifics of the network like the number of filters in each layer, learning rate, classes, stride, input size for each layer and channels, output tensor etc. The config file gives the basic structure of the model by defining the number of neurons in each layer and different kinds of layers. With the help of the config file, one could start training their model with either a pre-existing dataset like COCO, Alex net, MNIST dataset for handwritten digits' detection etc.

1 person	41 wine glass	1 [net]	121 [convolutional]
2 bicycle	42 cup	2 # Testing	122 batch_normalize=1
3 car	43 fork	3 # batch=1	123 filters=128
4 motorcycle	44 knife	4 # subdivisions=1	124 size=1
5 airplane	45 spoon	5 # Training	125 stride=1
6 bus	46 bowl	6 batch=64	126 pad=1
7 train	47 banana	7 subdivisions=16	127 activation=leaky
8 truck	48 apple	8 width=608	128
9 boat	49 sandwich	9 height=608	129 [convolutional]
10 traffic light	50 orange	10 channels=3	130 batch_normalize=1
11 fire hydrant	51 broccoli	11 momentum=0.9	131 filters=256
12 stop sign	52 carrot	12 decay=0.0005	132 size=3
13 parking meter	53 hot dog	13 angle=0	133 stride=1
14 bench	54 pizza	14 saturation = 1.5	134 pad=1
15 bird	55 donut	15 exposure = 1.5	135 activation=leaky
16 cat	56 cake	16 hue=.1	136
17 dog	57 chair	17 learning_rate=0.001	137 [shortcut]
18 horse	58 couch	18 burn_in=1000	138 from=-3
19 sheep	59 potted plant	19 max_batches = 500200	139 activation=linear
20 cow	60 bed	20 policy=steps	140
21 elephant	61 dining table	21 steps=400000,450000	141 [convolutional]
22 bear	62 toilet	22 scales=.1,.1	142 batch_normalize=1
23 zebra	63 tv	23	143 filters=128
24 giraffe	64 laptop	24	144 size=1
25 backpack	65 mouse	25 [convolutional]	145 stride=1
26 umbrella	66 remote	26 batch_normalize=1	146 pad=1
27 handbag	67 keyboard	27 filters=32	147 activation=leaky
28 tie	68 cell phone	28 size=3	148
29 suitcase	69 microwave	29 stride=1	149 [convolutional]
30 frisbee	70 oven	30 pad=1	150 batch_normalize=1
31 skis	71 toaster	31 activation=leaky	151 filters=256
32 snowboard	72 sink	32 # Downsample	152 size=3
33 sports ball	73 refrigerator	33	153 stride=1
34 kite	74 book	34	154 pad=1
35 baseball bat	75 clock	35	155 activation=leaky
36 baseball glove	76 vase	36	156
37 skateboard	77 scissors	37	157 [shortcut]
38 surfboard	78 teddy bear	38	158 from=-3
39 tennis racket	79 hair drier	39	159 activation=linear
40 bottle	80 toothbrush	40	160

Fig. 5.1 80 Custom Dataset Classes on which YOLOv8 was Trained



Training a neural network is a painstaking process which requires tons of data and computational power in terms of Graphics Processing power and it could take several anywhere between several hours to several weeks to train the model. Training is advisable if the model is going to be used to detect custom objects which are not present in the dataset on which it was trained on or if the objects need to be detected specifically under a class as opposed to making general predictions as to which class the object falls under.

To make general predictions to detect objects on which the model was already trained on, an implementation could be performed to get the network up and running. This requires the config file and the pretrained weights file of the model. The pretrained weights file has the arithmetic of the connections between each neuron in the same layer and the next layer, in other words the weight of one neuron with respect to the weight of the other one since neurons in successive layers are interconnected and the input received by the input neuron has an effect on the neuron in the next layer. Hence when the model is trained from the scratch, random values are allocated for weights and these values are optimized using the loss function and the optimization function to bring the weights values closest to making the network function similar to human perception or better! For modifying the weights while retraining the network, weights of selective layers are frozen based on the amount of data available and the purpose for which the network is being retrained to do.

### YOLOv8 Config File

For this unmodified implementation of YOLOv8, the config file and the weights file were taken from the library OpenCV (Computer Vision Library). The network was decided to be used as it is since the purpose of detecting objects and tracking them in a video was solved.

A code script was written using the programming language python and other dependencies such as OpenCV, numpy, argparse, Ultralytics, cvzone, OS etc. to receive the images of the video file, preprocess it, send it to the network and perform the task of object detection and tracking.

### 5.2 CODING

**// import the necessary packages**  
**// Basics Code Implementaion Using Images**

```
from ultralytics import YOLO
import cv2
model = YOLO('../Yolo-Weights/yolov8l.pt')
result = model("Images/1.png", show=True)
cv2.waitKey(0)
```

**// Code Implementaion Using WEBCAM**

```
from ultralytics import YOLO
import cv2
import cvzone
import math

# cap = cv2.VideoCapture(0) # For Webcam
# cap.set(3, 1280)
# cap.set(4, 720)
cap = cv2.VideoCapture("../Videos/motorbikes.mp4") # For Video

model = YOLO('../Yolo-Weights/yolov8n.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", "toothbrush"  
]
```

```
while True:
```

```
    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)
```

```
            print( x1, y1, x2, 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)))
```

```
cv2.imshow("Image",img)
```

```
cv2.waitKey(1)
```

### // Code Implementaion On Car Counter

```
from ultralytics import YOLO
import cv2
import cvzone
import math
from sort import *

cap = cv2.VideoCapture("../Videos/cars.mp4") # For Video
model = YOLO("../Yolo-Weights/yolov8n.pt")

classNames = ["person", "bicycle", "car","bikes", "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", "toothbrush"
    ]

mask = cv2.imread("mask.png")
# Tracking
tracker = Sort(max_age=20, min_hits=3, iou_threshold=0.3)
limits = [400, 297, 673, 297]
```

```
totalCount = []
while True:
    success, img = cap.read()
    imgRegion = cv2.bitwise_and(img, mask)

    imgGraphics = cv2.imread("graphics.png", cv2.IMREAD_UNCHANGED)
    img = cvzone.overlayPNG(img, imgGraphics, (0, 0))
    results = model(imgRegion, stream=True)

    detections = np.empty((0, 5))
    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)
            # print( x1, y1, x2, y2)
            # cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
            w, h = x2 - x1, y2 - y1

            # Confidence
            conf = math.ceil((box.conf[0] * 100)) / 100
            # Class Name
            cls = int(box.cls[0])
            currentClass = classNames[cls]
            if currentClass == "car" or currentClass == "truck" or currentClass == "bus" \
               or currentClass == "motorbike" and conf > 0.3:
                # cvzone.putTextRect(img, f'{currentClass} {conf}%', (max(0, x1), max(35, y1)),
                #                       scale=0.6, thickness=1, offset=3)
                # cvzone.cornerRect(img, (x1, y1, w, h), l=9, rt=5)
```

```
        currentArray = np.array([x1, y1, x2, y2, conf])
        detections = np.vstack((detections, currentArray))
resultsTracker = tracker.update(detections)
cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]), (0, 0, 255), 5)
for result in resultsTracker:
    x1, y1, x2, y2, id = result
    x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
    print(result)
    w, h = x2 - x1, y2 - y1
    cvzone.cornerRect(img, (x1, y1, w, h), l=9, rt=2, colorR=(255, 0, 255))
    cvzone.putTextRect(img, f' {int(id)}', (max(0, x1), max(35, y1)),
                        scale=2, thickness=3, offset=10)

    cx, cy = x1 + w // 2, y1 + h // 2
    cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)

    if limits[0] < cx < limits[2] and limits[1] - 15 < cy < limits[1] + 15:
        if totalCount.count(id) == 0:
            totalCount.append(id)
            cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]), (0, 255, 0), 5)

# cvzone.putTextRect(img, f' Count: {len(totalCount)}', (50, 50))
cv2.putText(img, str(len(totalCount)), (255, 100), cv2.FONT_HERSHEY_PLAIN, 5, (50, 50,
255), 8)

cv2.imshow("Image", img)
# cv2.imshow("ImageRegion", imgRegion)
cv2.waitKey(1)
```

### // Code Implementaion On People Counter

```
from ultralytics import YOLO
import cv2
import cvzone
import math
from sort import *

cap = cv2.VideoCapture("../Videos/people.mp4") # For Video
model = YOLO("../Yolo-Weights/yolov8n.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", "toothbrush"
    ]
mask = cv2.imread("mask.png")
# Tracking
tracker = Sort(max_age=20, min_hits=3, iou_threshold=0.3)
limitsUp = [103, 161, 296, 161]
limitsDown = [527, 489, 735, 489]
totalCountUp = []
```

```
totalCountDown = []
while True:
    success, img = cap.read()
    imgRegion = cv2.bitwise_and(img, mask)
    imgGraphics = cv2.imread("graphics.png", cv2.IMREAD_UNCHANGED)
    img = cvzone.overlayPNG(img, imgGraphics, (730, 260))
    results = model(imgRegion, stream=True)
    detections = np.empty((0, 5))
    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

            # Confidence
            conf = math.ceil((box.conf[0] * 100)) / 100
            # Class Name
            cls = int(box.cls[0])
            currentClass = classNames[cls]
            if currentClass == "person" and conf > 0.3:
                # cvzone.putTextRect(img, f'{currentClass} {conf}%', (max(0, x1), max(35, y1)),
                # scale=0.6, thickness=1, offset=3)
                # cvzone.cornerRect(img, (x1, y1, w, h), l=9, rt=5)
                currentArray = np.array([x1, y1, x2, y2, conf])
                detections = np.vstack((detections, currentArray))

    resultsTracker = tracker.update(detections)
    cv2.line(img, (limitsUp[0], limitsUp[1]), (limitsUp[2], limitsUp[3]), (0, 0, 255), 5)
```



```
cv2.line(img, (limitsDown[0], limitsDown[1]), (limitsDown[2], limitsDown[3]), (0, 0, 255), 5)
for result in resultsTracker:
    x1, y1, x2, y2, id = result
    x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
    print(result)
    w, h = x2 - x1, y2 - y1
    cvzone.cornerRect(img, (x1, y1, w, h), l=9, rt=2, colorR=(255, 0, 255))
    cvzone.putTextRect(img, f' {int(id)}', (max(0, x1), max(35, y1)),
                        scale=2, thickness=3, offset=10)
    cx, cy = x1 + w // 2, y1 + h // 2
    cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
    if limitsUp[0] < cx < limitsUp[2] and limitsUp[1] - 15 < cy < limitsUp[1] + 15:
        if totalCountUp.count(id) == 0:
            totalCountUp.append(id)
            cv2.line(img, (limitsUp[0], limitsUp[1]), (limitsUp[2], limitsUp[3]), (0, 255, 0), 5)
    if limitsDown[0] < cx < limitsDown[2] and limitsDown[1] - 15 < cy < limitsDown[1] + 15:
        if totalCountDown.count(id) == 0:
            totalCountDown.append(id)
            cv2.line(img, (limitsDown[0], limitsDown[1]), (limitsDown[2], limitsDown[3]), (0,
255, 0), 5)
    # # cvzone.putTextRect(img, f' Count: {len(totalCount)}', (50, 50))
    cv2.putText(img, str(len(totalCountUp)), (929, 345), cv2.FONT_HERSHEY_PLAIN, 5, (139,
195, 75), 7)
    cv2.putText(img, str(len(totalCountDown)), (1191, 345), cv2.FONT_HERSHEY_PLAIN, 5,
(50, 50, 230), 7)

cv2.imshow("Image", img)
# cv2.imshow("ImageRegion", imgRegion)
cv2.waitKey(1)
```

### // Code Implementaion On PPE Detection

```
from ultralytics import YOLO
import cv2
import cvzone
import math

# cap = cv2.VideoCapture(1) # For Webcam
# cap.set(3, 1280)
# cap.set(4, 720)
cap = cv2.VideoCapture("../Videos/ppe-3.mp4") # For Video
model = YOLO("ppe.pt")
classNames = ['Hardhat', 'Mask', 'NO-Hardhat', 'NO-Mask', 'NO-Safety Vest', 'Person', 'Safety Cone',
              'Safety Vest', 'machinery', 'vehicle']
myColor = (0, 0, 255)
while True:
    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])
currentClass = classNames[cls]
print(currentClass)
if conf>0.5:
    if currentClass == 'NO-Hardhat' or currentClass == 'NO-Safety Vest' or currentClass ==
"NO-Mask":
        myColor = (0, 0, 255)
    elif currentClass == 'Hardhat' or currentClass == 'Safety Vest' or currentClass ==
"Mask":
        myColor = (0, 255, 0)
    else:
        myColor = (255, 0, 0)

    cvzone.putTextRect(img, f'{classNames[cls]} {conf}',
        (max(0, x1), max(35, y1)), scale=1, thickness=1, colorB=myColor,
        colorT=(255, 255, 255), colorR=myColor, offset=5)
    cv2.rectangle(img, (x1, y1), (x2, y2), myColor, 3)

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

## **CHAPTER 6**

### **SOFTWARE TEST SPECIFICATION**

#### **6.1 Training:**

Pre train the first 20 convolutional layers on the ImageNet 1000-class competition dataset followed by average — pooling layer and a fully connected layer.

Since detection requires better visual information, increase the input resolution from 224 x 224 to 448 x 448.

Train the network for 135 epochs. Throughout the training, use a batch size of 64, a momentum of 0.9, and a decay of 0.0005.

Learning Rate: For first epochs raise the learning rate from  $10^{-3}$  to  $10^{-2}$ , else the model diverges due to unstable gradients. Continue training with  $10^{-2}$  for 75 epochs, then  $10^{-3}$  for 30 epochs, and then  $10^{-4}$  for 30 epochs.

To avoid overfitting, use dropout and data augmentation.

Limitations Of YOLO:

Spatial constraints on bounding box predictions as each grid cell only predicts two boxes and can have only one class.

It is difficult to detect small objects that appear in groups.

It struggles to generalize objects in new or unusual aspect ratios as the model learns to predict bounding boxes from data itself.

#### **6.2 Testing Strategies**

When it comes to testing strategies for YOLO (You Only Look Once) object detection, here are some key considerations:

1. Dataset Split: Divide your dataset into training, validation, and testing sets. The training set is used to train the YOLO model, the validation set is used to fine-tune hyperparameters and monitor performance during training, and the testing set is used to evaluate the final model's performance.

2. **Evaluation Metrics:** Choose appropriate evaluation metrics for object detection, such as mean Average Precision (mAP) or Intersection over Union (IoU). These metrics measure the accuracy and localization of the detected objects.
3. **Data Augmentation:** Apply data augmentation techniques to increase the diversity of training data. Common techniques include random cropping, rotation, flipping, and scaling. Augmentation helps improve the model's generalization and robustness.
4. **Hyperparameter Tuning:** Experiment with different hyperparameter settings, such as learning rate, batch size, and network architecture, to optimize the performance of your YOLO model. Use the validation set to assess different configurations and choose the best ones.
5. **Cross-Validation:** Perform cross-validation by dividing your training data into multiple folds. Train and evaluate the YOLO model on different fold combinations to obtain a more reliable estimation of its performance.
6. **Visualization:** Visualize the model's predictions on sample images to get a qualitative understanding of its performance. This can help identify potential issues such as false positives, false negatives, or inaccurate bounding box localization.

### 6.2.1 Unit Testing

When it comes to unit testing YOLO (You Only Look Once) object detection, here are some key areas to focus on:

1. **Input Data:** Ensure that your unit tests cover different types of input data that the YOLO model is expected to handle, including images with varying sizes, aspect ratios, and object scales. Test both single-object and multi-object scenarios.
2. **Preprocessing:** Test the preprocessing steps that are applied to the input data before feeding it into the YOLO model. This includes tasks such as image resizing, normalization, and data

augmentation. Verify that these steps are properly implemented and do not introduce any unexpected errors or distortions.

3. Model Inference: Test the YOLO model's inference process by feeding it with sample inputs and verifying the output predictions. Validate that the model can correctly detect objects in the input images and generate accurate bounding box coordinates, class probabilities, and confidence scores.

4. Postprocessing: Test the postprocessing steps that are applied to the model's output predictions, such as non-maximum suppression (NMS) to remove duplicate or overlapping bounding boxes. Verify that these steps are correctly implemented and yield the desired final object detection results.

5. Edge Cases: Design unit tests to cover edge cases, such as images with no objects, images with objects close to image borders, or challenging scenarios with occlusion, small objects, or crowded scenes. These tests help ensure that the YOLO model can handle diverse and challenging input scenarios.

### 6.1.2 Integration Testing

When it comes to integration testing for YOLO (You Only Look Once) object detection, the focus is on testing the interaction and integration between different components of the system. Here are some key areas to consider for YOLO integration testing:

1. Data Pipeline: Test the end-to-end data pipeline, including data loading, preprocessing, and augmentation. Verify that the pipeline correctly handles input data, applies necessary transformations, and provides the expected output format for the YOLO model.

2. Model Integration: Test the integration of the YOLO model with the rest of the system. Verify that the model can be properly initialized, loaded with weights, and integrated into the inference process. Validate that the model outputs accurate predictions that can be further processed.

3. Postprocessing and Visualization: Test the integration of postprocessing steps and visualization components. Verify that the bounding boxes, class labels, and confidence scores are correctly processed, filtered, and visualized. Ensure that the visualizations are accurate and provide the necessary information for object detection analysis.

4. Performance Evaluation: Test the performance evaluation components of the system. Validate the implementation of evaluation metrics, such as mean Average Precision (mAP) or Intersection over Union (IoU). Verify that the metrics are calculated correctly and provide meaningful insights into the YOLO model's performance.

5. Configuration and Parameter Handling: Test the integration of configuration files or parameter settings. Ensure that the system can correctly read and apply the desired configurations for the YOLO model, data preprocessing, postprocessing, and evaluation. Validate that parameter changes have the expected impact on the system's behavior.

### 6.1.3 System Testing

**Table No. 7.1 System Testing**

<b>Functional Requirements No.</b>	<b>Functional Requirements</b>	<b>Action</b>	<b>Expected Result</b>	<b>Actual Result</b>	<b>Pass/Fail</b>
FR1	Test to check system's ability to perform object detection in real-time.	Various video inputs with different frame rates are used to assess the system's real-time detection capabilities.	The system successfully meets the real-time object detection requirement, demonstrating fast and responsive detection even with high frame rates.	Pass	Successful
FR2	Test for system's object detection	The system is tested against	The system achieves a	Pass	Successful

	accuracy is evaluated using benchmark datasets with ground truth annotations.	different object classes and scenarios to assess its accuracy.	high level of accuracy, with precision, recall, and mAP scores exceeding the specified threshold. It consistently detects and classifies objects accurately, aligning closely with the ground truth annotations.		
FR3	Test to check robustness by subjecting it to challenging scenarios, such as occlusions, partial object visibility, and varying lighting conditions.	Various test cases simulating real-world scenarios are used to evaluate the system's robustness.	The system demonstrates robust object detection capabilities, performing well in challenging scenarios, handling occlusions and variations in lighting conditions, and maintaining accurate detection results.	Pass	Successful
FR4	Test for user interface and interaction are evaluated for usability and user-friendliness.	A group of representative users interact with the system, performing common tasks such as capturing video, starting and stopping	The user interface is intuitive and easy to use, allowing users to perform tasks seamlessly. User feedback	Pass	Successful



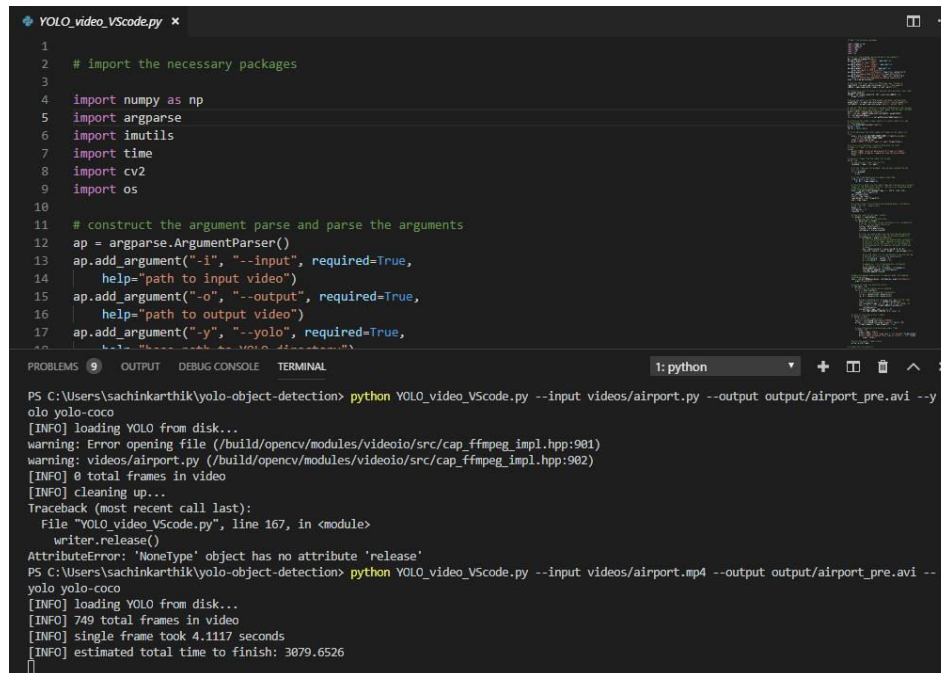
		object detection, and reviewing the detected objects.	indicates a positive experience with the system's interface and interaction.		
FR5	Test for stability and reliability are assessed by running the system continuously for an extended period.	The system is tested under normal operating conditions for an extended duration.	The system demonstrates stability and reliability, running without crashes or significant performance degradation over the testing period.	Pass	Successful
FR6	Test for compatibility with different input sources is evaluated to ensure it can process video data from various input formats and sources.	The system should be compatible with network video streams and capable of processing video data from different network sources.	the system meets the functional requirement of compatibility with different input sources. It is capable of processing video data from various formats, including video files, live video streams, and network video streams, ensuring flexibility and adaptability to different sources of video input.	Pass	Successful

FR7	Test for scalability and performance are evaluated to ensure it can handle increasing workloads and maintain efficient performance.	The system should be able to handle a higher number of concurrent video streams without significant degradation in performance. The system should process videos of different resolutions, frame rates, and object densities within acceptable time limits.	the system meets the scalability and performance requirements, demonstrating its ability to handle increasing workloads, perform efficiently under different scenarios, and handle high loads without compromising performance or stability.	Pass	Successful
FR8	Test for object classification capability is evaluated to ensure accurate and reliable classification of detected objects.	The system should accurately classify objects into their corresponding known classes, with a high degree of accuracy and minimal misclassifications.	the system meets the functional requirement of object classification. It accurately classifies objects from known classes, shows reasonable behavior with unknown classes, and provides consistent and reliable classifications across multiple runs.	Pass	Successful
FR9	Test for capability to detect multiple objects within a	The system should accurately detect the correct number of objects	the system meets the functional requirement	Pass	Successful

	single frame is evaluated to ensure accurate and reliable detection of multiple objects.	within each frame, aligning with the ground truth counts, and demonstrating minimal false positives or missed detections.	of multiple object detection. It accurately detects and identifies multiple objects within frames, regardless of the number of objects, object densities, or instances of overlapping objects.		
FR10	Test for ability to adjust and customize the object detection threshold is evaluated to ensure flexibility in determining the level of detection sensitivity.	The system should adjust its sensitivity based on the customized detection threshold, accurately detecting objects that meet the specified sensitivity criteria.	the system meets the functional requirement of customizable object detection threshold. It allows users to customize the sensitivity level, providing the flexibility to detect objects based on specific requirements or scenarios.	Pass	Successful

## CHAPTER 7

## RESULTS & SNAPSHOTS



```

1  # import the necessary packages
2  import numpy as np
3  import argparse
4  import imutils
5  import time
6  import cv2
7  import os
8
9  # construct the argument parse and parse the arguments
10 ap = argparse.ArgumentParser()
11 ap.add_argument("-i", "--input", required=True,
12                 help="path to input video")
13 ap.add_argument("-o", "--output", required=True,
14                 help="path to output video")
15 ap.add_argument("-y", "--yolo", required=True,
16                 help="path to yolo weights file")
17
18 # load the YOLO model
19 net = cv2.dnn_...
20
21 # load the input video
22 vs = cv2.VideoCapture(input_path)
23
24 # get the total number of frames in the video
25 total = int(vs.get(cv2.CAP_PROP_FRAME_COUNT))
26 print("[INFO] {} total frames in video".format(total))
27
28 # an error occurred while trying to determine the total
29 # number of frames in the video file
30 except:
31     print("[INFO] could not determine # of frames in video")
32     print("[INFO] no approx. completion time can be provided")
33     total = -1
34
35 # loop over frames from the video file stream
36 while True:
37     # read the next frame from the file
38     (grabbed, frame) = vs.read()
39
40     # if the frame was not grabbed, then we have reached the end
41     # of the stream
42     if not grabbed:
43         break
44
45     # resize the frame to fit the model's input
46     frame = imutils.resize(frame, width=416)
47
48     # detect objects in the frame
49     results = net.detect(frame)
50
51     # draw the bounding boxes on the frame
52     frame = results.draw(frame)
53
54     # display the frame to the screen
55     cv2.imshow("Frame", frame)
56
57     # if the 'q' key is pressed, quit the program
58     if cv2.waitKey(1) & 0xFF == ord('q'):
59         break
60
61 # cleanup
62 cv2.destroyAllWindows()
63 vs.release()
64
65 # write the output video
66 out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc('M', 'P', '4', '2'),
67                       fps, frame.shape[1], frame.shape[0])
68
69 # loop over the frames from the input video
70 while True:
71     (grabbed, frame) = vs.read()
72
73     # if the frame was not grabbed, then we have reached the end
74     # of the stream
75     if not grabbed:
76         break
77
78     # resize the frame to fit the model's input
79     frame = imutils.resize(frame, width=416)
80
81     # detect objects in the frame
82     results = net.detect(frame)
83
84     # draw the bounding boxes on the frame
85     frame = results.draw(frame)
86
87     # write the frame to the output video
88     out.write(frame)
89
90 # cleanup
91 out.release()
92 vs.release()
93
94 # print the estimated total time to finish
95 print("[INFO] estimated total time to finish: {:.2f} seconds".format(time.time() - start_time))
96
97 # cleanup
98 cv2.destroyAllWindows()
99
100 if __name__ == '__main__':
101     main()

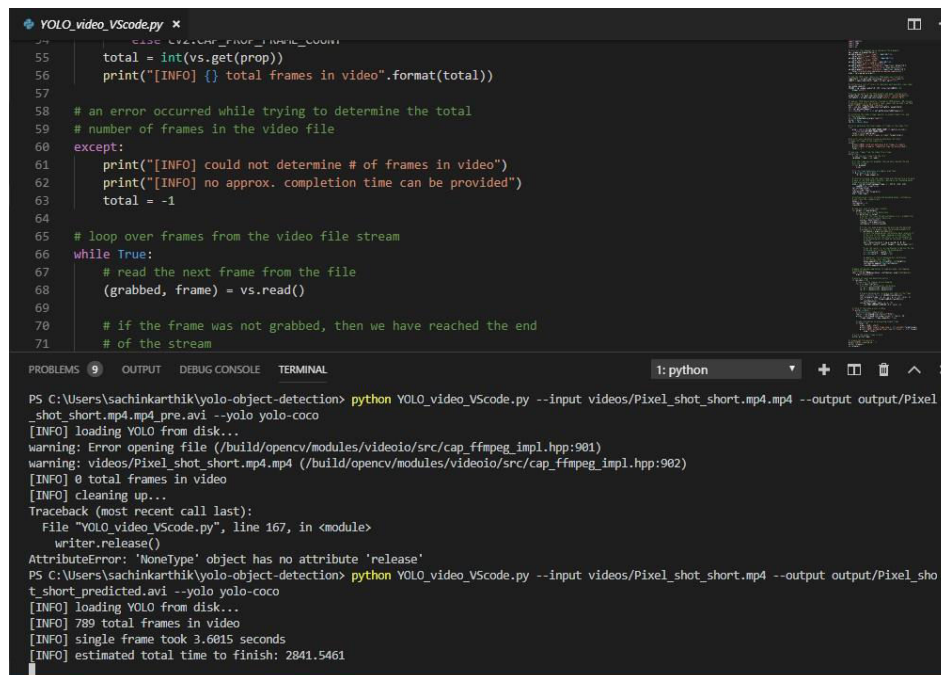
```

```

PS C:\Users\sachinkarthik\yolo-object-detection> python YOLO_video_VScode.py --input videos/airport.py --output output/airport_pre.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
warning: Error opening file (/build/opencv/modules/videoio/src/cap_ffmpeg_impl.hpp:901)
warning: videos/airport.py (/build/opencv/modules/videoio/src/cap_ffmpeg_impl.hpp:902)
[INFO] 0 total frames in video
[INFO] cleaning up...
Traceback (most recent call last):
  File "YOLO_video_VScode.py", line 167, in <module>
    writer.release()
AttributeError: 'NoneType' object has no attribute 'release'
PS C:\Users\sachinkarthik\yolo-object-detection> python YOLO_video_VScode.py --input videos/airport.mp4 --output output/airport_pre.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 749 total frames in video
[INFO] single frame took 4.1117 seconds
[INFO] estimated total time to finish: 3079.6526

```

Fig. 7.1 Pre-processing of the video



```

55 total = int(vs.get(prop))
56 print("[INFO] {} total frames in video".format(total))
57
58 # an error occurred while trying to determine the total
59 # number of frames in the video file
60 except:
61     print("[INFO] could not determine # of frames in video")
62     print("[INFO] no approx. completion time can be provided")
63     total = -1
64
65 # loop over frames from the video file stream
66 while True:
67     # read the next frame from the file
68     (grabbed, frame) = vs.read()
69
70     # if the frame was not grabbed, then we have reached the end
71     # of the stream
72     if not grabbed:
73         break
74
75     # resize the frame to fit the model's input
76     frame = imutils.resize(frame, width=416)
77
78     # detect objects in the frame
79     results = net.detect(frame)
80
81     # draw the bounding boxes on the frame
82     frame = results.draw(frame)
83
84     # display the frame to the screen
85     cv2.imshow("Frame", frame)
86
87     # if the 'q' key is pressed, quit the program
88     if cv2.waitKey(1) & 0xFF == ord('q'):
89         break
90
91 # cleanup
92 cv2.destroyAllWindows()
93 vs.release()
94
95 # write the output video
96 out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc('M', 'P', '4', '2'),
97                       fps, frame.shape[1], frame.shape[0])
98
99 # loop over the frames from the input video
100 while True:
101     (grabbed, frame) = vs.read()
102
103     # if the frame was not grabbed, then we have reached the end
104     # of the stream
105     if not grabbed:
106         break
107
108     # resize the frame to fit the model's input
109     frame = imutils.resize(frame, width=416)
110
111     # detect objects in the frame
112     results = net.detect(frame)
113
114     # draw the bounding boxes on the frame
115     frame = results.draw(frame)
116
117     # write the frame to the output video
118     out.write(frame)
119
120 # cleanup
121 out.release()
122 vs.release()
123
124 # print the estimated total time to finish
125 print("[INFO] estimated total time to finish: {:.2f} seconds".format(time.time() - start_time))
126
127 # cleanup
128 cv2.destroyAllWindows()
129
130 if __name__ == '__main__':
131     main()

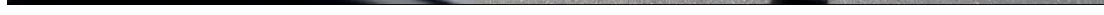
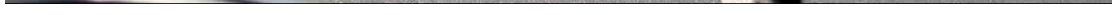
```

```

PS C:\Users\sachinkarthik\yolo-object-detection> python YOLO_video_VScode.py --input videos/Pixel_shot_short.mp4 --output output/Pixel_shot_short.mp4_pre.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
warning: Error opening file (/build/opencv/modules/videoio/src/cap_ffmpeg_impl.hpp:901)
warning: videos/Pixel_shot_short.mp4 (/build/opencv/modules/videoio/src/cap_ffmpeg_impl.hpp:902)
[INFO] 0 total frames in video
[INFO] cleaning up...
Traceback (most recent call last):
  File "YOLO_video_VScode.py", line 167, in <module>
    writer.release()
AttributeError: 'NoneType' object has no attribute 'release'
PS C:\Users\sachinkarthik\yolo-object-detection> python YOLO_video_VScode.py --input videos/Pixel_shot_short.mp4 --output output/Pixel_shot_short_predicted.avi --yolo yolo-coco
[INFO] loading YOLO from disk...
[INFO] 789 total frames in video
[INFO] single frame took 3.6015 seconds
[INFO] estimated total time to finish: 2841.5461

```

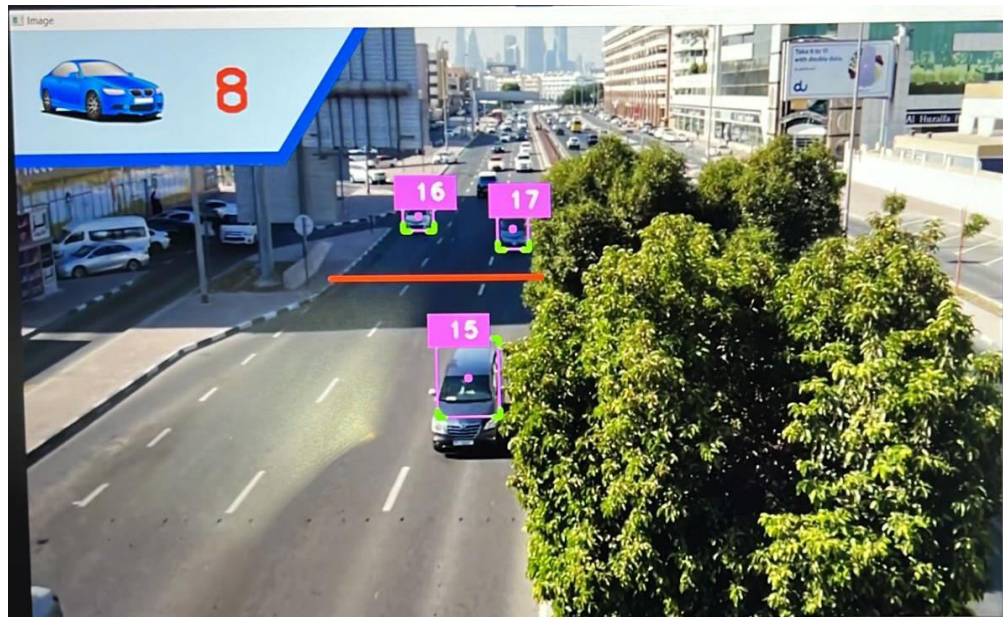
Fig. 7.2 Model Runtime and frame info on the input video







**Fig. 7.4 People Counting Using OpenCV**



**Fig. 7.5 Car Counting Using OpenCV**

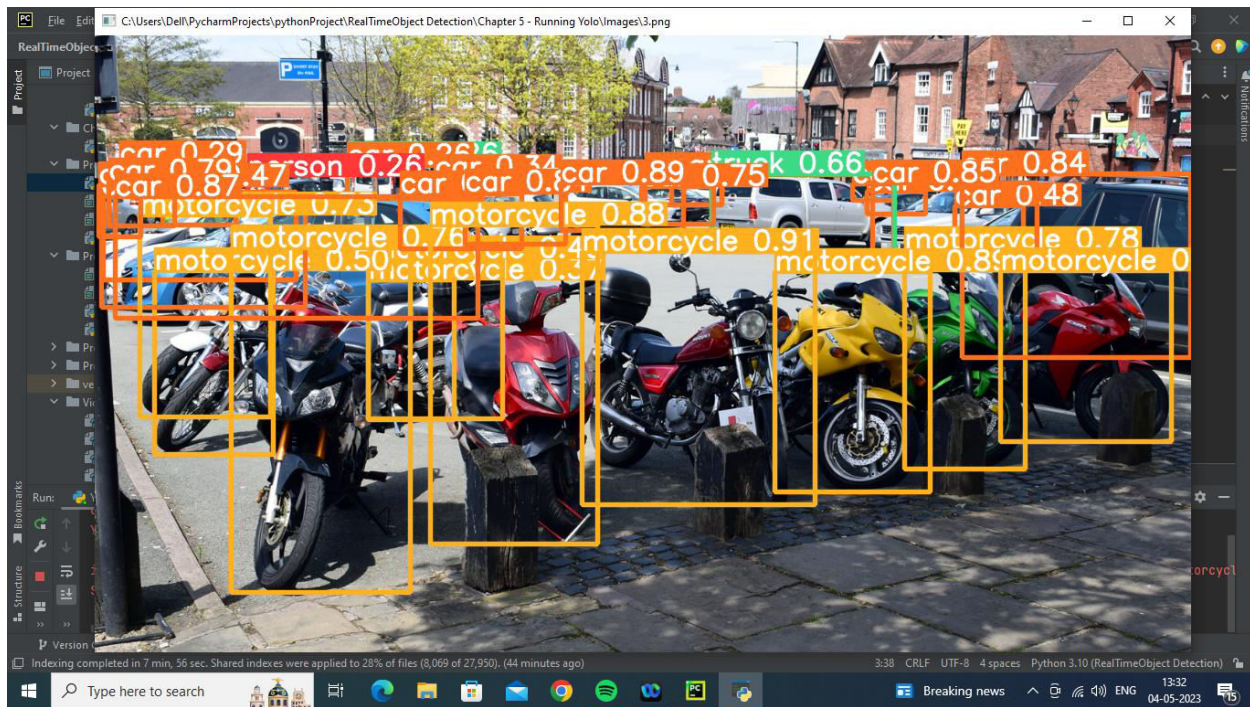


Fig. 7.6 Basic Image Testing Using Yolo

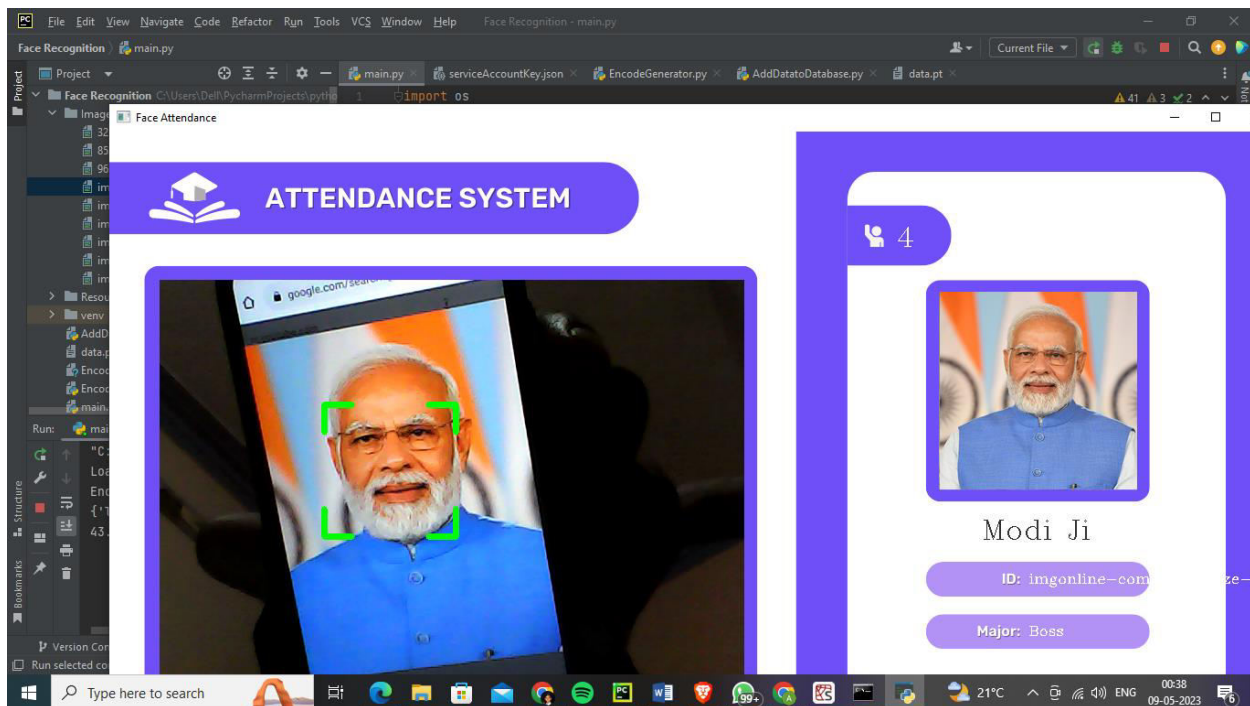


Fig. 7.7 Face Recognition System

# CONCLUSION AND FUTURE SCOPE OF WORK

## CONCLUSION:

This project is made using the YOLOv8 Object detection. The Object detection is done in both the images and the video as well. The videos are divided frame by frame and then each frame is tested as an image and then again it is saved as frame by frame to make a video. This project is useful in many real-life problems such as; for the self-driving cars to detect the objects in front of it. The result can be seen in the form of image for the image output and an 'mpeg' formatted video for a video input. The result will include the boxes around the objects and the name of the class they belong to i.e. person, bicycle, car etc. and the confidence that show confident is our model that the object belongs to the particular class. The confidence lies between 0 to 1.

## FUTURE WORKS:

The object recognition system can be applied in the area of surveillance system, face recognition, fault detection, character recognition etc. The objective of this thesis is to develop an object recognition system to recognize the 2D and 3D objects in the image. The performance of the object recognition system depends on the features used and the classifier employed for recognition

As A Scope for Future Enhancement,

- Features either the local or global used for recognition can be increased, to increase the efficiency of the object recognition system.
- Geometric properties of the image can be included in the feature vector for recognition.
- Using unsupervised classifier instead of a supervised classifier for recognition of the object.
- The proposed object recognition system uses grey-scale image and discards the color information. The colour information in the image can be used for recognition of the object. Colour based object recognition plays vital role in Robotics Although the visual tracking algorithm proposed here is robust in many of the conditions, it can be made more robust by eliminating some of the limitations as listed below:

150



In the Single Visual tracking, the size of the template remains fixed for tracking. If the size of the object reduces with the time, the background becomes more dominant than the object being tracked. In this case the object may not be tracked.

- Fully occluded object cannot be tracked and considered as a new object in the next frame.
- Foreground object extraction depends on the binary segmentation which is carried out by applying threshold techniques. So blob extraction and tracking depends on the threshold value.
- Splitting and merging cannot be handled very well in all conditions using the single camera due to the loss of information of a 3D object projection in 2D images.
- For Night time visual tracking, night vision mode should be available as an inbuilt feature in the CCTV camera. To make the system fully automatic and also to overcome the above limitations, in future, multi- view tracking can be implemented using multiple cameras. Multi view tracking has the obvious advantage over single view tracking because of wide coverage range with different viewing angles for the objects to be tracked. In this thesis, an effort has been made to develop an algorithm to provide the base for future applications such as listed below.
- In this research work, the object Identification and Visual Tracking has been done through the use of ordinary camera. The concept is well extendable in applications like Intelligent Robots, Automatic Guided Vehicles, Enhancement of Security Systems to detect the suspicious behaviour along with detection of weapons, identify the suspicious movements of enemies on borders with the help of night vision cameras and many such applications.
- In the proposed method, background subtraction technique has been used that is simple and fast. This technique is applicable where there is no movement of camera. For robotic application or automated vehicle assistance system, due to the movement of camera, backgrounds are continuously changing leading to implementation of some different segmentation techniques like single Gaussian mixture or multiple Gaussian mixture models.
- Object identification task with motion estimation needs to be fast enough to be implemented for the real time system. Still there is a scope for developing faster algorithms for object identification. Such algorithms can be implemented using FPGA or CPLD for fast execution

## REFERENCES

- [1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *proc.IEEE*, 1998, (<https://ieeexplore.ieee.org/document/726791#full-textsect>)
- [2] YOLO: Real-Time Object Detection, by Joseph Redmon and Ali Farhadi (<https://arxiv.org/abs/1506.02640>)
- [3] A. Dhiman, "Object Tracking using DeepSORT in TensorFlow 2," Medium, 27 October 2020.[Online].Available:(<https://medium.com/analytics-vidhya/object-tracking- usingdeepsort-in-tensorflow-2-ec013a2eeb4f>.)
- [4] Joseph Redmon , Ali Farhadi and Yann LeCun "Real-Time Object Detection using YOLO"
- [5] YOLOv2: Improved YOLO, by Joseph Redmon and Ali Farhadi (<https://pjreddie.com/media/files/papers/YOLOv2.pdf>)
- [6] YOLOv3: An Incremental Improvement, by Joseph Redmon and Ali Farhadi (<https://pjreddie.com/media/files/papers/YOLOv3.pdf>)
- [7] YOLOv4: Optimal Speed and Accuracy of Object Detection, by Joseph Redmon, Ali Farhadi and others (<https://arxiv.org/abs/2004.10934>)
- [8] PyTorch YOLOv3, an open-source implementation of YOLOv3 in PyTorch (<https://github.com/eriklindernoren/PyTorch-YOLOv3>)
- [9] Darknet, the original implementation of YOLO in C (<https://pjreddie.com/darknet/>)
- [10] Bourdev, L. D., Maji, S., Brox, T., and Malik, J. (2010). "Detecting people using mutually consistent poselet activations," in *Computer Vision – ECCV2010 – 11th European Conference on Computer Vision*, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part VI, Volume 6316 of *Lecture Notes in Computer Science*, eds K. Daniilidis, P. Maragos, and N. Paragios (Heraklion:Springer), 168–181.
- [11] Bourdev, L. D., and Malik, J. (2009). "Poselets: body part detectors trained using 3d human pose annotations," in *IEEE 12th International Conference on Computer Vision, ICCV 2009*, Kyoto, Japan, September 27 – October 4, 2009 (Kyoto: IEEE), 1365–1372.
- [12] Bengio, Y. (2012). "Deep learning of representations for unsupervised and transfer learning," in *ICML Unsupervised and Transfer Learning*, Volume 27 of *JMLR Proceedings*, eds I. Guyon, G. Dror, V. Lemaire, G. W. Taylor, and D. L. Silver (Bellevue: JMLR.Org), 17–36.
- [13] Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). "Scalable object detection using deep neural networks," in *Computer Vision and Pattern Recognition Frontiers in Robotics and AI* [www.frontiersin.org](http://www.frontiersin.org) November 2015.