



**Department of Scientific Computing, Modelling and Simulation  
Savitribai Phule Pune University**

**Academic Year: 2024–2025**

**Title: Comparative Study on Object Detection  
Using CNN And YOLO**

Under the Guidance of  
**Dr. Kaveri Kale**

- Name: Bhushan Zade
- Roll No: MS2421
- Guide: Dr. Kaveri Kale
- Academic year: 2025–26
- Date: 10 oct 2025

## **DECLARATION**

I hereby declare that the project entitled “Comparative Study on Object Detection Using CNN and YOLO” is an original work carried out by me under the guidance of Dr. Kaveri Kale and has not been submitted previously to any university or institution .

All sources of information used in this project have been duly acknowledged.

Bhushan Zade

(Signature)

## **ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to my project guide Dr. Kaveri Kale for her valuable guidance, encouragement, and continuous support throughout the course of this project.

I am thankful to the faculty members of the Department of Scientific Computing , Savitribai Phule Pune University, for providing the necessary facilities and academic environment required for the successful completion of this work.

.

Finally, I express my gratitude to all those who directly or indirectly contributed to the successful completion of this project.

Bhushan Zade

## ABSTRACT

Object detection is a fundamental problem in computer vision that involves identifying and localizing objects within images. With the advancement of deep learning, Convolutional Neural Networks (CNNs) have significantly improved object detection performance. However, achieving a balance between detection accuracy and real-time performance remains a challenge.

This project presents a comparative study of object detection using CNN-based Faster R-CNN and YOLOv5, a single-stage object detection framework. Both models were trained and evaluated on the Pascal VOC 2007 dataset. Performance was analyzed using standard metrics such as Precision, Recall, Mean Average Precision (mAP), inference time, and Frames Per Second (FPS).

Experimental results show that Faster R-CNN provides high detection accuracy but suffers from slower inference speed. In contrast, YOLOv5 achieves competitive accuracy while delivering real-time performance, making it suitable for time-critical applications. The study highlights the trade-off between speed and accuracy and provides insights into selecting appropriate object detection models based on application requirements.

## 1. INTRODUCTION

### 1.1 Background

Object detection is one of the most fundamental and challenging problems in the field of computer vision and machine learning. Unlike image classification, where the task is to assign a single label to an image, object detection involves both identifying the presence of objects and localizing them within an image using bounding boxes. This dual requirement makes object detection computationally more complex and practically more significant.

With the rapid growth of applications such as autonomous vehicles, video surveillance, medical image analysis, robotics, and smart cities, efficient object detection systems have become essential. Early object detection techniques relied heavily on handcrafted features such as Haar features, Histogram of Oriented Gradients (HOG), and Scale-Invariant Feature Transform (SIFT), combined with traditional machine learning classifiers like Support Vector Machines (SVMs). While these methods achieved moderate success, they lacked robustness and scalability when applied to complex real-world scenarios.

The emergence of deep learning, particularly Convolutional Neural Networks (CNNs), has significantly transformed object detection. CNNs automatically learn hierarchical feature representations directly from data, eliminating the need for manual feature engineering. This advancement has led to the development of powerful object detection frameworks such as R-CNN, Fast R-CNN, Faster R-CNN, and YOLO.

---

### 1.2 Motivation

Although CNN-based object detectors provide high detection accuracy, many of them suffer from high computational cost and slow inference speed. Two-stage detectors like Faster R-CNN first generate region proposals and then classify each region, resulting in increased latency. This limitation makes them unsuitable for real-time applications such as live video processing and autonomous navigation.

On the other hand, single-stage detectors like YOLO (You Only Look Once) reformulate object detection as a regression problem and perform detection in a

single forward pass of the network. This architectural design significantly improves inference speed, making YOLO suitable for real-time object detection tasks.

The motivation behind this project is to experimentally analyze and compare CNN-based object detection using Faster R-CNN with YOLO-based object detection, focusing on both accuracy and speed. By conducting a comparative study using a standard benchmark dataset, this project aims to provide practical insights into the trade-offs between these two approaches.

---

### 1.3 Problem Statement

Despite significant advancements in object detection, selecting an appropriate detection framework for a given application remains challenging. High-accuracy models often suffer from slow inference times, while real-time models may compromise on detection precision.

The problem addressed in this project is:

*How do CNN-based two-stage object detectors compare with YOLO-based single-stage detectors in terms of accuracy and inference speed when trained on a standard benchmark dataset?*

---

### 1.4 Objectives of the Project

The primary objectives of this project are:

1. To study the fundamentals of CNN-based object detection.
2. To implement a CNN-based object detector using Faster R-CNN.
3. To implement a YOLO-based object detector using YOLOv5.
4. To preprocess and prepare the Pascal VOC dataset for both models.
5. To evaluate both models using standard metrics such as Precision, Recall, and mAP.

6. To analyze inference time and frames per second (FPS) for both approaches.
7. To perform a comparative study highlighting the trade-offs between accuracy and speed.

## 1.5 Scope of the Project

The scope of this project includes:

- Training and evaluating object detection models on the Pascal VOC 2007 dataset.
- Comparison between two-stage and single-stage object detection frameworks.
- Analysis of performance metrics and inference speed.
- Visualization of detection results and training performance.

The project does not focus on deploying the models in production environments or optimizing them for edge devices.

---

## 1.6 Organization of the Report

The remainder of this report is organized as follows:

- Chapter 2 presents a detailed literature review on object detection techniques.
- Chapter 3 describes the dataset, tools, and technologies used.
- Chapter 4 explains the methodology and model architectures.
- Chapter 5 discusses the implementation details.
- Chapter 6 presents experimental results and evaluation.
- Chapter 7 provides a comparative analysis between Faster R-CNN and YOLO.
- Chapter 8 concludes the work and outlines future research directions.

## 2. LITERATURE REVIEW

### 2.1 Introduction to Object Detection

Object detection is a fundamental problem in computer vision that involves identifying objects of interest in an image and determining their precise locations. Unlike image classification, which assigns a single label to an entire image, object detection must solve two tasks simultaneously:

1. Classification – identifying the object category
2. Localization – predicting bounding box coordinates

Early object detection systems relied on handcrafted features and classical machine learning algorithms. However, these approaches struggled with complex backgrounds, varying object scales, occlusion, and illumination changes. The advent of deep learning, particularly Convolutional Neural Networks (CNNs), has revolutionized object detection by enabling end-to-end learning of features directly from data.

---

### 2.2 Traditional Object Detection Techniques

Before deep learning, object detection was primarily based on feature engineering. Popular methods included:

#### 2.2.1 Haar Feature-Based Detection

Viola and Jones introduced a real-time face detection framework using Haar-like features and AdaBoost classifiers. While effective for face detection, this approach was limited to specific object categories and lacked generalization.

#### 2.2.2 Histogram of Oriented Gradients (HOG)

The HOG descriptor captures gradient orientation information and was widely used for pedestrian detection. Combined with Support Vector Machines (SVMs), HOG achieved reasonable accuracy but required manual feature design and was computationally expensive for large datasets.

#### 2.2.3 Limitations of Traditional Methods

Traditional methods suffered from:

- Poor scalability
- High dependence on handcrafted features

- Limited robustness to real-world variations

These limitations motivated the transition to deep learning-based approaches.

---

## 2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are a class of deep learning models specifically designed for visual data. CNNs consist of convolutional layers, pooling layers, and fully connected layers that automatically learn hierarchical features from images.

Key advantages of CNNs include:

- Automatic feature extraction
- Translation invariance
- High accuracy on large datasets

CNNs laid the foundation for modern object detection architectures.

---

## 2.4 Region-Based CNN (R-CNN)

Girshick et al. introduced the Region-Based Convolutional Neural Network (R-CNN), which marked a major breakthrough in object detection. R-CNN works by:

1. Generating region proposals using selective search
2. Passing each region through a CNN
3. Classifying each region using SVMs

Limitations of R-CNN

- Extremely slow training and inference
  - High memory usage
  - Multiple independent models
- 

## 2.5 Fast R-CNN

Fast R-CNN improved upon R-CNN by sharing convolutional computations. Instead of running CNNs on each region proposal separately, Fast R-CNN processes the entire image once and extracts region features using ROI pooling.

## Improvements

- Faster training and inference
- End-to-end training
- Improved accuracy

However, Fast R-CNN still relied on selective search, which remained a computational bottleneck.

---

## 2.6 Faster R-CNN

Faster R-CNN introduced the Region Proposal Network (RPN) to generate region proposals directly from convolutional feature maps. This innovation eliminated selective search and significantly improved speed.

### Architecture Overview

- Backbone CNN (e.g., ResNet)
- Region Proposal Network
- ROI pooling
- Classification and bounding box regression

### Strengths

- High detection accuracy
- Precise localization
- Strong performance on benchmark datasets

### Limitations

- Two-stage detection increases inference time
- Not suitable for real-time applications

Faster R-CNN represents the CNN-based object detector used in this project.

---

## 2.7 Single-Stage Object Detectors

To overcome the speed limitations of two-stage detectors, researchers proposed single-stage detectors that directly predict bounding boxes and class probabilities.

Examples include:

- SSD (Single Shot Detector)
- YOLO (You Only Look Once)

These models trade a small amount of accuracy for significant gains in speed.

---

## 2.8 YOLO (You Only Look Once)

YOLO reformulates object detection as a regression problem. Instead of generating region proposals, YOLO divides the image into a grid and predicts bounding boxes and class probabilities in a single forward pass.

### Key Characteristics

- Single-stage architecture
- End-to-end training
- Real-time inference

### Advantages

- Extremely fast detection
- Global context awareness
- Lower false positives in background regions

### Limitations

- Difficulty detecting small objects
  - Slightly lower localization accuracy compared to two-stage detectors
- 

## 2.9 Evolution of YOLO Models

YOLO has evolved through multiple versions:

- YOLOv1: Introduced unified detection framework
- YOLOv2: Improved accuracy and anchor boxes
- YOLOv3: Multi-scale detection
- YOLOv5: Lightweight, efficient, PyTorch-based

### **3. DATASET AND TOOLS**

#### **3.1 Introduction**

The performance of any machine learning or deep learning model depends heavily on the quality of the dataset and the tools used for implementation. For object detection tasks, a well-annotated dataset with diverse object categories is essential to train robust and generalizable models.

In this project, the Pascal Visual Object Classes (VOC) 2007 dataset was used as the benchmark dataset. The models were implemented and trained using modern deep learning frameworks and GPU acceleration to ensure efficient experimentation and evaluation.

#### **3.2 Pascal VOC Dataset**

##### **3.2.1 Overview of Pascal VOC**

The Pascal Visual Object Classes (VOC) dataset is one of the most widely used benchmark datasets in computer vision research. It was introduced to standardize the evaluation of object detection and classification algorithms.

Key characteristics of the Pascal VOC dataset include:

- Real-world images
- Multiple object categories per image
- Accurate bounding box annotations
- Standardized train, validation, and test splits

In this project, the Pascal VOC 2007 dataset was used.

##### **3.2.2 Dataset Statistics**

The Pascal VOC 2007 dataset contains:

- Total images: 9,963
- Training set: 2,501 images
- Validation set: 2,510 images
- Test set: 4,952 images

- Object categories: 20

The dataset includes images with varying resolutions, lighting conditions, object scales, and backgrounds, making it suitable for evaluating object detection models under realistic conditions.

---

### 3.2.3 Object Categories

The Pascal VOC dataset contains the following 20 object classes:

1. Aeroplane
2. Bicycle
3. Bird
4. Boat
5. Bottle
6. Bus
7. Car
8. Cat
9. Chair
10. Cow
11. Dining Table
12. Dog
13. Horse
14. Motorbike
15. Person
16. Potted Plant
17. Sheep
18. Sofa
19. Train
20. TV/Monitor

These categories represent a diverse set of objects commonly encountered in everyday scenes.

---

### 3.3 Dataset Structure

#### 3.3.1 Original VOC Dataset Structure

The Pascal VOC dataset is organized into the following directory structure:

- Annotations/  
Contains XML files with bounding box annotations for each image.
- JPEGImages/  
Contains the original image files.
- ImageSets/  
Contains text files specifying training, validation, and test splits.
- SegmentationClass/ and SegmentationObject/  
Contain segmentation masks (not used in this project).

Each XML annotation file includes:

- Object class labels
  - Bounding box coordinates (xmin, ymin, xmax, ymax)
- 

#### 3.3.2 Dataset Preparation for Faster R-CNN

For the CNN-based object detector (Faster R-CNN), the dataset was used directly in its original VOC format. The PyTorch `torchvision.datasets.VOCDetection` class was used to load images and annotations.

This allowed:

- Direct use of XML annotations
  - Automatic parsing of bounding boxes and labels
  - Seamless integration with PyTorch models
- 

#### 3.3.3 Dataset Preparation for YOLO

YOLO requires a different annotation format compared to Faster R-CNN. Therefore, the Pascal VOC annotations were converted from XML format to YOLO text format.

In YOLO format:

- Each image has a corresponding .txt file
- Each line represents one object
- Format:  
`<class_id> <x_center> <y_center> <width> <height>`
- All values are normalized between 0 and 1

The dataset was reorganized into the following structure:

VOC\_YOLO/

```
├── images/
|   ├── train/
|   └── val/
└── labels/
    ├── train/
    └── val/
```

This preprocessing step ensured compatibility with the YOLO training pipeline.

---

### 3.4 Hardware Requirements

#### 3.4.1 Computing Environment

The experiments were conducted using GPU-accelerated environments to reduce training time and improve performance.

The hardware configuration used includes:

- GPU: NVIDIA Tesla T4
- GPU Memory: 15 GB
- CPU: Multi-core processor (cloud-based)
- RAM: Sufficient for large-scale training

GPU acceleration was essential for training deep convolutional networks efficiently.

---

### 3.5 Software Tools and Frameworks

#### 3.5.1 Programming Language

- Python was used as the primary programming language due to its extensive support for machine learning and deep learning libraries.
- 

#### 3.5.2 Deep Learning Frameworks

The following frameworks were used:

- PyTorch  
Used for implementing Faster R-CNN and handling model training and inference.
  - Torchvision  
Provided pre-trained models, datasets, and utilities for object detection.
  - YOLOv5 (PyTorch-based)  
Used for implementing and training the YOLO object detection model.
- 

#### 3.5.3 Supporting Libraries

Additional libraries used in the project include:

- NumPy – numerical operations
  - Matplotlib – visualization of results
  - OpenCV – image processing
  - Pillow (PIL) – image handling
  - Pandas – result analysis and metrics storage
- 

### 3.6 Development Platform

#### 3.6.1 Kaggle Notebook Environment

## **4. METHODOLOGY**

### **4.1 Introduction**

Methodology defines the systematic approach followed to design, implement, and evaluate the object detection models in this project. The primary objective of this methodology is to ensure a fair and consistent comparison between CNN-based object detection using Faster R-CNN and single-stage object detection using YOLO.

The methodology consists of dataset preparation, model selection, training strategy, evaluation metrics, and result analysis. Both models were trained and evaluated on the same dataset to ensure comparability.

---

### **4.2 Overall System Architecture**

The overall workflow of the project is shown below:

1. Dataset selection and analysis
2. Dataset preprocessing
3. Model selection
4. Model training
5. Model evaluation
6. Comparative analysis

Both Faster R-CNN and YOLO models follow a similar high-level pipeline but differ significantly in their internal architectures and detection strategies.

---

### **4.3 Data Preprocessing Methodology**

#### **4.3.1 Image Preprocessing**

Before feeding images into the models, basic preprocessing steps were applied:

- Image resizing
- Conversion to tensor format
- Normalization of pixel values

For Faster R-CNN, preprocessing was handled internally by the PyTorch framework. For YOLO, images were resized to a fixed input size of  $640 \times 640$  during training.

---

#### 4.3.2 Annotation Preprocessing for Faster R-CNN

Faster R-CNN uses bounding box annotations in the following format:

- (xmin, ymin, xmax, ymax)

The annotations were parsed directly from Pascal VOC XML files using the VOCDetection dataset loader provided by torchvision. This allowed seamless integration of the dataset into the training pipeline without manual conversion.

---

#### 4.3.3 Annotation Conversion for YOLO

YOLO requires annotations in a normalized text-based format. Therefore, Pascal VOC XML annotations were converted into YOLO format using a custom preprocessing script.

The YOLO annotation format is defined as:

<class\_id> <x\_center> <y\_center> <width> <height>

All coordinates were normalized with respect to the image dimensions. This conversion ensured compatibility with the YOLOv5 training framework.

---

### 4.4 CNN-Based Object Detection Using Faster R-CNN

#### 4.4.1 Faster R-CNN Architecture

Faster R-CNN is a two-stage object detection framework consisting of:

1. Backbone Network
2. Region Proposal Network (RPN)
3. ROI Pooling Layer
4. Classification and Bounding Box Regression Head

In this project, a pre-trained ResNet backbone was used to extract deep feature representations from input images.

---

#### 4.4.2 Region Proposal Network (RPN)

The RPN generates candidate object regions by sliding small convolutional filters over the feature maps. For each spatial location, the RPN predicts:

- Objectness score
- Bounding box offsets

Anchors of multiple scales and aspect ratios are used to detect objects of varying sizes.

---

#### 4.4.3 ROI Pooling and Classification

The proposed regions are then passed through an ROI pooling layer, which converts variable-sized proposals into fixed-sized feature maps. These features are fed into fully connected layers for:

- Object classification
  - Bounding box refinement
- 

#### 4.4.4 Faster R-CNN Training Strategy

The Faster R-CNN model was trained using:

- Stochastic Gradient Descent (SGD)
- Pre-trained backbone weights
- Batch-wise training on GPU

Loss functions used include:

- Classification loss
  - Bounding box regression loss
- 

### 4.5 YOLO-Based Object Detection Methodology

#### 4.5.1 YOLO Detection Principle

YOLO (You Only Look Once) formulates object detection as a regression problem. Instead of generating region proposals, YOLO predicts bounding boxes and class probabilities directly from the full image in a single forward pass.

This single-stage detection approach significantly reduces inference time.

---

#### 4.5.2 YOLOv5 Architecture

YOLOv5 consists of three main components:

1. Backbone – CSPDarknet
2. Neck – PANet for feature aggregation
3. Head – Detection layers for multi-scale prediction

The architecture enables detection of objects at different scales and improves accuracy for both small and large objects.

---

#### 4.5.3 YOLO Training Pipeline

The YOLOv5 training pipeline includes:

- Automatic anchor box optimization
- Data augmentation techniques
- Multi-scale training
- End-to-end backpropagation

The model was initialized using pre-trained weights (yolov5s.pt) to accelerate convergence and improve performance.

---

#### 4.5.4 YOLO Loss Function

YOLO uses a composite loss function consisting of:

- Bounding box loss
- Objectness loss
- Classification loss

These loss components jointly optimize localization accuracy and classification performance.

## 4.6 Training Configuration and Hyperparameters

The following hyperparameters were used for YOLO training:

- Image size:  $640 \times 640$
- Batch size: 16
- Epochs: 30
- Optimizer: SGD
- Learning rate: Default YOLOv5 settings

GPU acceleration was used throughout training to reduce computation time.

---

## 4.7 Evaluation Methodology

To evaluate model performance, the following metrics were used:

- Precision
- Recall
- Mean Average Precision (mAP@0.5)
- Mean Average Precision (mAP@0.5:0.95)
- Inference time
- Frames Per Second (FPS)

These metrics provide a comprehensive evaluation of both detection accuracy and computational efficiency.

---

## 4.8 Experimental Workflow

The experimental workflow followed these steps:

1. Dataset preparation and preprocessing
2. Training Faster R-CNN model
3. Training YOLOv5 model
4. Validation and evaluation
5. Result visualization
6. Comparative analysis

## **5. IMPLEMENTATION DETAILS**

### **5.1 Introduction**

Implementation details translate the theoretical methodology into a working system. This chapter focuses on how Faster R-CNN and YOLOv5 were implemented using PyTorch, how the dataset was processed, and how experiments were executed in a GPU-enabled environment. Emphasis is placed on reproducibility, clarity, and alignment with the experimental goals of the project.

---

### **5.2 Development Environment Setup**

#### **5.2.1 Platform Selection**

The project was implemented using the Kaggle Notebook environment, which provides free GPU access and pre-installed deep learning libraries. This platform was chosen due to its ease of use, reproducibility, and ability to handle large-scale deep learning experiments without local hardware limitations.

---

#### **5.2.2 Hardware Configuration**

The experiments were executed on the following hardware configuration:

- GPU: NVIDIA Tesla T4
- GPU Memory: 15 GB
- CPU: Cloud-based multi-core processor
- RAM: Adequate for deep learning workloads

GPU acceleration was critical for reducing training and inference time.

---

#### **5.2.3 Software Environment**

The software environment consisted of:

- Python 3.x
- PyTorch (for deep learning model implementation)
- Torchvision (for Faster R-CNN and dataset utilities)

- YOLOv5 (PyTorch-based implementation)

Additional libraries such as NumPy, Pandas, Matplotlib, OpenCV, and Pillow were used for data processing and visualization.

---

### 5.3 Dataset Implementation

#### 5.3.1 Loading Pascal VOC Dataset for Faster R-CNN

The Pascal VOC 2007 dataset was loaded using the `VOCDetection` class provided by the `torchvision.datasets` module. This class automatically parses XML annotations and returns images along with bounding box and label information.

The dataset was split into training and validation sets according to the standard Pascal VOC protocol. Minimal preprocessing was required, as Faster R-CNN internally handles image normalization and resizing.

---

#### 5.3.2 Dataset Conversion for YOLO

YOLO requires a different annotation format compared to Faster R-CNN. Therefore, a custom preprocessing script was implemented to convert Pascal VOC XML annotations into YOLO-compatible text files.

Key steps in the conversion process include:

- Parsing XML annotation files
- Extracting bounding box coordinates
- Normalizing bounding box values
- Assigning class indices
- Organizing images and labels into YOLO directory structure

This conversion step was essential to ensure compatibility with the YOLOv5 training pipeline.

---

### 5.4 Faster R-CNN Model Implementation

#### 5.4.1 Model Initialization

The Faster R-CNN model was implemented using the `torchvision.models.detection` module. A pre-trained Faster R-CNN model with a

ResNet backbone was selected to leverage transfer learning and accelerate convergence.

The final classification layer was modified to match the number of object classes in the Pascal VOC dataset.

---

#### 5.4.2 Training Procedure

The Faster R-CNN model was trained using:

- Mini-batch gradient descent
- Pre-trained backbone weights
- GPU acceleration

During training, the model optimized both classification loss and bounding box regression loss. Training was performed for a limited number of epochs due to the high computational cost of two-stage detectors.

---

#### 5.4.3 Inference and Timing Measurement

Inference time for Faster R-CNN was measured by passing validation images through the trained model and recording the average processing time per image. Frames Per Second (FPS) was calculated as the reciprocal of inference time.

This measurement provided a baseline for comparison with YOLO.

---

### 5.5 YOLOv5 Model Implementation

#### 5.5.1 YOLOv5 Repository Setup

The YOLOv5 implementation was obtained from the official Ultralytics repository. The repository includes scripts for training, validation, and inference, along with pre-trained weights.

The required dependencies were installed, and the repository was configured to run on GPU.

---

### 5.5.2 Dataset Configuration File

A dataset configuration file (voc.yaml) was created to specify:

- Dataset paths
- Number of classes
- Class names

This configuration file enabled YOLOv5 to locate training and validation data correctly.

---

### 5.5.3 Training Configuration

YOLOv5 was trained using the following configuration:

- Image size:  $640 \times 640$
- Batch size: 16
- Number of epochs: 30
- Pre-trained weights: yolov5s.pt

Training logs, metrics, and model checkpoints were automatically saved during training.

---

### 5.5.4 Validation and Best Model Selection

YOLOv5 automatically performs validation at the end of each epoch. The model with the highest validation mAP was saved as the best model, which was later used for evaluation and inference.

---

## 5.6 Result Storage and Logging

All training results were stored in the runs/train/exp directory. This directory contains:

- Training logs
- Model weights
- Metric plots
- Confusion matrix

## 6. RESULTS AND EVALUATION

### 6.1 Introduction

This chapter presents the experimental results and performance evaluation of the object detection models implemented in this project. The performance of Faster R-CNN and YOLOv5 is evaluated using standard object detection metrics such as Precision, Recall, Mean Average Precision (mAP), inference time, and Frames Per Second (FPS).

The evaluation is performed on the Pascal VOC 2007 validation dataset to ensure consistency and fair comparison.

---

### 6.2 Evaluation Metrics

To assess the performance of object detection models, the following metrics were used:

#### 6.2.1 Precision

Precision measures the proportion of correctly detected objects among all predicted objects.

$$\text{Precision} = \frac{TP}{TP + FP}$$

High precision indicates fewer false positives.

---

#### 6.2.2 Recall

Recall measures the proportion of actual objects that were correctly detected.

$$\text{Recall} = \frac{TP}{TP + FN}$$

High recall indicates fewer missed detections.

---

#### 6.2.3 Mean Average Precision (mAP)

Mean Average Precision is the most commonly used metric in object detection. Two variants were used:

- mAP@0.5: Intersection over Union (IoU) threshold of 0.5
- mAP@0.5:0.95: Average mAP across IoU thresholds from 0.5 to 0.95

These metrics provide a comprehensive evaluation of both localization and classification accuracy.

---

#### 6.2.4 Inference Time and FPS

Inference time measures the average time taken by the model to process a single image. Frames Per Second (FPS) is calculated as the inverse of inference time and indicates real-time performance capability.

---

### 6.3 Faster R-CNN Results

#### 6.3.1 Detection Performance

Faster R-CNN demonstrated strong object detection performance due to its two-stage architecture. The model effectively localized objects and produced accurate bounding boxes, particularly for medium and large-sized objects.

---

#### 6.3.2 Inference Speed Analysis

The measured inference performance for Faster R-CNN is as follows:

- Average inference time per image: 0.136 seconds
- Frames per second (FPS): 7.35

This indicates that Faster R-CNN is computationally intensive and not suitable for real-time object detection tasks.

---

#### 6.3.3 Observations

- High localization accuracy
  - Slower inference speed
  - Suitable for offline analysis and accuracy-critical applications
- 

### 6.4 YOLOv5 Results

#### 6.4.1 Quantitative Detection Metrics

The YOLOv5 model achieved the following performance on the Pascal VOC validation set:

- Precision: 0.768
- Recall: 0.648
- mAP@0.5: 0.717
- mAP@0.5:0.95: 0.453

These results demonstrate that YOLOv5 achieves competitive detection accuracy despite its lightweight, single-stage architecture.

---

#### 6.4.2 Inference Speed Analysis

YOLOv5 demonstrated significantly faster inference compared to Faster R-CNN:

- Average inference time per image: 0.045 seconds
- Frames per second (FPS): 22.15

This confirms that YOLOv5 supports real-time object detection.

---

#### 6.4.3 Class-wise Performance Analysis

YOLOv5 performed particularly well on frequently occurring and rigid objects such as:

- Person
- Car
- Train
- Horse

Lower performance was observed for smaller objects such as bottle and potted plant, which is consistent with known limitations of single-stage detectors.

---

### 6.5 Visualization of Results

#### 6.5.1 Detection Output Visualization

YOLOv5 detection results were visualized on validation images. The model successfully detected multiple objects within a single image and produced accurate bounding boxes along with confidence scores.

These visualizations demonstrate the practical effectiveness of YOLOv5 in real-world scenarios.

---

### 6.5.2 Training and Validation Curves

Training and validation performance curves generated during YOLO training show:

- Decreasing loss values
- Increasing precision and mAP
- Stable convergence without overfitting

These observations confirm effective model learning.

---

### 6.5.3 Confusion Matrix Analysis

The confusion matrix illustrates strong class-wise prediction accuracy for common object categories while highlighting confusion among visually similar or small objects.

---

## 6.6 Comparative Performance Summary

| Metric               | Faster R-CNN YOLOv5 |       |
|----------------------|---------------------|-------|
| Precision            | High                | 0.768 |
| Recall               | High                | 0.648 |
| mAP@0.5              | High                | 0.717 |
| Inference Time (sec) | 0.136               | 0.045 |
| FPS                  | 7.35                | 22.15 |
| Real-Time Capability | No                  | Yes   |

---

## 6.7 Discussion

The results clearly indicate a trade-off between detection accuracy and inference speed. Faster R-CNN prioritizes accuracy and precise localization, whereas YOLOv5 emphasizes real-time performance with competitive accuracy.

The choice of object detection model should therefore depend on application requirements

---

## 7. COMPARATIVE ANALYSIS

### 7.1 Introduction

The purpose of this chapter is to perform a detailed comparative analysis of CNN-based object detection using Faster R-CNN and single-stage object detection using YOLOv5. While both models aim to detect and localize objects within images, their internal architectures and detection strategies differ significantly. This chapter highlights these differences and analyzes their impact on accuracy, speed, and practical usability.

---

### 7.2 Architectural Comparison

#### 7.2.1 Faster R-CNN Architecture

Faster R-CNN follows a two-stage detection pipeline. In the first stage, the Region Proposal Network (RPN) generates candidate object regions. In the second stage, these regions are classified and refined using fully connected layers. This separation of proposal generation and classification allows the model to achieve high localization accuracy.

However, this architecture increases computational complexity and inference time, as multiple processing steps are required for a single image.

---

## 7.2.2 YOLOv5 Architecture

YOLOv5 follows a single-stage detection approach. The model processes the entire image in a single forward pass and directly predicts bounding boxes and class probabilities. By eliminating the region proposal stage, YOLO significantly reduces computation time and achieves real-time performance.

The use of multi-scale detection layers enables YOLOv5 to detect objects of varying sizes efficiently.

---

## 7.3 Accuracy Comparison

### 7.3.1 Detection Accuracy

Faster R-CNN generally achieves strong detection accuracy due to its two-stage design and precise region proposals. This makes it particularly effective in scenarios requiring accurate localization.

YOLOv5, while slightly less precise in localization, achieved competitive accuracy in this project, with a precision of 76.8% and an mAP@0.5 of 71.7%. These values indicate that YOLOv5 performs well on standard object detection benchmarks despite its lightweight design.

---

### 7.3.2 Class-wise Performance

Both models performed well on commonly occurring object categories such as person and car. YOLOv5 showed reduced performance for small or visually complex objects such as bottle and potted plant, which aligns with known limitations of single-stage detectors.

---

## 7.4 Speed and Computational Efficiency

### 7.4.1 Inference Time Comparison

The inference time analysis clearly demonstrates the efficiency difference between the two models:

- Faster R-CNN: 0.136 seconds per image

- YOLOv5: 0.045 seconds per image

YOLOv5 is approximately three times faster than Faster R-CNN.

---

#### 7.4.2 Frames Per Second (FPS)

- Faster R-CNN: 7.35 FPS
- YOLOv5: 22.15 FPS

This difference highlights YOLOv5's ability to support real-time object detection applications such as video surveillance and autonomous systems.

---

### 7.5 Trade-off Analysis

The comparison reveals a clear trade-off between accuracy and speed:

- Faster R-CNN prioritizes accuracy and precise localization but incurs higher inference latency.
- YOLOv5 sacrifices a small amount of localization accuracy in exchange for significantly faster detection.

This trade-off is a critical consideration when selecting an object detection framework for a specific application.

---

### 7.6 Practical Application Perspective

From a practical standpoint:

- Faster R-CNN is suitable for offline applications where accuracy is more important than speed, such as image analysis and research tasks.
- YOLOv5 is better suited for real-time applications, including video processing, surveillance systems, and robotics.
- 

The experimental results from this project support these application-driven choices.

## **8. CONCLUSION AND FUTURE SCOPE**

### **8.1 Conclusion**

This project presented a comprehensive comparative study of object detection techniques using CNN-based Faster R-CNN and single-stage YOLOv5 models. The primary objective was to analyze the trade-off between detection accuracy and inference speed using a standard benchmark dataset, namely Pascal VOC 2007.

A complete object detection pipeline was designed and implemented, starting from dataset preprocessing to model training, evaluation, and result visualization. Faster R-CNN was implemented as a representative two-stage detector, while YOLOv5 was selected as a modern single-stage detector optimized for real-time performance. Both models were trained and evaluated under similar conditions to ensure a fair comparison.

The experimental results demonstrate that Faster R-CNN provides strong localization accuracy due to its region proposal mechanism and two-stage architecture. However, this accuracy comes at the cost of high computational complexity and slower inference speed. With an average inference time of 0.136 seconds per image and a frame rate of 7.35 FPS, Faster R-CNN is not suitable for real-time applications.

On the other hand, YOLOv5 achieved a strong balance between speed and accuracy. The model attained a precision of 76.8%, recall of 64.8%, and an mAP@0.5 of 71.7% on the Pascal VOC validation set. Most importantly, YOLOv5 demonstrated real-time performance with an inference time of 0.045 seconds per image and a frame rate of 22.15 FPS. These results confirm that YOLOv5 is well-suited for time-critical object detection tasks.

The comparative analysis clearly highlights the speed–accuracy trade-off inherent in object detection models. While Faster R-CNN is preferable for accuracy-critical and offline analysis tasks, YOLOv5 is a more practical choice for real-time systems such as video surveillance, autonomous navigation, and robotics.

Overall, the project successfully achieved its objectives and provided practical insights into the strengths and limitations of CNN-based and YOLO-based object detection frameworks.

## 8.2 Contributions of the Project

The major contributions of this project are summarized as follows:

1. A detailed study of object detection techniques using CNN-based and YOLO-based models.
  2. Implementation of Faster R-CNN and YOLOv5 using the Pascal VOC 2007 dataset.
  3. Conversion of Pascal VOC annotations to YOLO format and preparation of a custom training pipeline.
  4. Comprehensive evaluation using standard metrics such as Precision, Recall, mAP, inference time, and FPS.
  5. Experimental validation of the speed–accuracy trade-off between two-stage and single-stage detectors.
  6. Visualization of detection results, training curves, and confusion matrices to support quantitative findings.
- 

## 8.3 Limitations of the Project

Despite successful implementation and evaluation, the project has certain limitations:

- The experiments were conducted on a single dataset (Pascal VOC 2007), which limits generalization to more complex datasets.
- Only one variant of Faster R-CNN and YOLO was considered.
- The models were not optimized for deployment on edge or mobile devices.
- Hyperparameter tuning was limited due to computational constraints.

These limitations provide opportunities for further improvement and exploration.

---

## 8.4 Future Scope

The scope of this project can be extended in several directions:

1. Use of Advanced YOLO Versions

Newer versions such as YOLOv7 and YOLOv8 can be explored to further improve accuracy and inference speed.

2. Training on Larger Datasets

Training models on larger datasets such as MS COCO can improve generalization and robustness.

3. Integration of Transformer-Based Models

Incorporating transformer-based object detection frameworks like DETR may enhance performance, especially for complex scenes.

4. Small Object Detection Enhancement

Techniques such as attention mechanisms and feature pyramid optimization can be applied to improve detection of small objects.

5. Real-Time Video and Edge Deployment

The trained models can be deployed on real-time video streams and edge devices such as NVIDIA Jetson platforms.

6. Model Optimization

Techniques like pruning, quantization, and knowledge distillation can be used to reduce model size and inference latency.

---

## 8.5 Final Remarks

This project provides a practical and experimental understanding of modern object detection systems. By systematically comparing Faster R-CNN and YOLOv5, the study highlights how architectural design choices impact both performance and usability. The findings of this project can serve as a strong foundation for further research and development in computer vision and deep learning.