"REAL TIME OBJECT DETECTION USING CONVOLUTION NEURAL NETWORK"

Project report submitted

in partial fulfilment of the requirement for the degree of

Bachelor of Technology

By

ISHA TYAGI
(213025017)

VANSH SALAR
(213025047)

TARUN KUMAR
(213025043)

SUPERVISOR
MR. KAPIL KUMAR

COMPUTER SCIENCE ENGINEERING DEPARTMENT

COLLEGE OF SMART COMPUTING, COER UNIVERSITY

(APRIL, 2025)

CERTIFICATE

It is certified that the work contained in the project report titled “Real-Time Object Detection using convolution Neural Network,” by “Isha Tyagi, Vansh Salar, Tarun Kumar” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree

Signature of Supervisor(s)

Mr. Kapil Kumar CSE

DEPARTMENT COER

UNIVERSITY

April, 2025

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

Isha Tyagi (213025017)

_____

Vansh Salar (213025047)

_____

Tarun Kumar (21302543)

_____

Date:

_____

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

I am grateful to my project guide Mr. Kaipil Kumar for the guidance, inspiration and constructive suggestions that helped me in the preparation of this project.

I am also thankful to my colleagues who have helped me in successful completion of the project.

ABSTRACT

The rapid advancements in artificial intelligence and computer vision have enabled machines to perceive and understand visual data similarly to humans. One such application is real-time object detection, a crucial aspect of intelligent systems like autonomous vehicles, surveillance cameras, and robotics. This project, titled "Real-Time Object Detection Using Neural Networks," explores the implementation of deep learning models to accurately detect and classify multiple objects within live video feeds.

The objective of this project is to design and deploy a neural network-based system capable of identifying and localizing objects in real-time with high precision and low latency. We utilize Convolutional Neural Networks (CNNs), particularly YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) models, for their balance between speed and accuracy. These models are trained on large datasets such as COCO or PASCAL VOC, which contain diverse object classes and varied environments, enhancing the generalization of the system.

The architecture is implemented using Python, with the aid of deep learning libraries like TensorFlow and OpenCV for real-time image processing and display. Key components of the system include video frame capture, pre-processing, model inference, post-processing (such as non-max suppression), and visualization of bounding boxes with class labels.

Testing and evaluation demonstrate that the model performs well under different lighting conditions and backgrounds, maintaining consistent detection speeds above 30 frames per second (FPS), making it suitable for real-world deployment. Furthermore, the system's modularity allows for easy integration with embedded systems or cloud platforms, enabling broader applications.

This project not only showcases the practical utility of neural networks in real-time scenarios but also provides a foundation for future enhancements, including tracking, multi-object recognition, and edge-based-deployment for IoT systems.

CHAPTER-1 INTRODUCTION

1.1 Project Objective:

The primary objective of this project is to design and develop an efficient real-time object detection system using Convolutional Neural Networks (CNNs) that can accurately identify and localize multiple objects within an image or video frame. The system aims to operate in real time, enabling practical deployment in areas such as surveillance, autonomous vehicles, robotics, smart city infrastructure, and industrial automation.

Specifically, the project seeks to:

1. **Leverage the capabilities of deep learning and CNN architectures** to automate the process of object detection, overcoming the limitations of traditional computer vision techniques which often rely on handcrafted features and are less adaptable to complex or dynamic environments.
2. **Build a robust model architecture** that is capable of:
    - Extracting high-level spatial features from raw input images.
    - Predicting both the **class label** and **bounding box coordinates** for every detected object within a single forward pass.
    - Handling varying object scales, occlusions, and complex backgrounds effectively.
3. **Implement a real-time detection pipeline** using state-of-the-art CNN-based models such as:
    - **YOLO (You Only Look Once)**
    - **SSD (Single Shot Multi-Box Detector)**
    - Or **Faster R-CNN**, depending on the required balance between accuracy and speed.
4. **Train the model on benchmark datasets** like COCO, PASCAL VOC, or a custom dataset relevant to the application, and perform preprocessing, augmentation, and annotation to ensure the model generalizes well to unseen data.
5. **Evaluate the performance** of the detection system using standard metrics such as:
    - Mean Average Precision (MAP)
    - Intersection over Union (IOU)
    - Precision-Recall Curves
    - Inference Time per frame
6. **Optimize the model** for real-time performance, including techniques like:
    - Quantization

- Pruning
- GPU acceleration
- Use of lightweight architectures like Tiny-YOLO

7. **Develop a user-friendly interface or visualization tool** (possibly using OpenCV or a web-based framework) to display real-time detection results with bounding boxes and labels overlaid on the video feed.
8. **Ensure scalability and flexibility** so the system can be extended to detect custom objects in different real-world domains with minimal retraining.

By achieving these objectives, the project contributes to advancing automated visual perception systems that are fast, accurate, and adaptable to various real-world applications requiring instant recognition and decision-making based on visual inputs.

1.2 Background:

In recent years, the field of computer vision has witnessed remarkable progress, primarily due to the advancements in deep learning and neural network architectures. One of the most significant applications of computer vision is object detection, which involves identifying and localizing objects within an image or video frame. Unlike image classification, which only labels an image as a whole, object detection provides precise information about what objects are present and where they are located in real time. This capability has wide-ranging applications in areas such as autonomous driving, video surveillance, industrial automation, medical imaging, and augmented reality.

Traditional object detection methods relied heavily on handcrafted features and conventional machine learning algorithms, which often suffered from low accuracy and poor scalability in complex environments. With the emergence of deep learning, particularly Convolutional Neural Networks (CNNs), object detection has become significantly more accurate and efficient. Neural networks can learn hierarchical features from raw pixel data, enabling systems to detect a variety of objects with minimal pre-processing.

This project aims to design and implement a system for real-time object detection using neural networks. The focus is on achieving a balance between accuracy and speed, which is essential for real-time applications. We explore state-of-the-art object detection models like YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), both of which offer fast inference times and can detect multiple objects per frame with high accuracy.

The system is built using Python and utilizes powerful libraries such as TensorFlow, Keras, and OpenCV. A standard webcam or video input serves as the real-time data source, and each frame is processed through the neural network to detect and label objects dynamically. To ensure real-time performance, optimizations like GPU acceleration and efficient image pre-processing techniques are incorporated.

Through this project, we demonstrate how neural networks can be applied effectively to solve real-world problems that require real-time responses. The implementation not only serves as a practical example of deep learning in action but also lays the groundwork for more advanced systems involving object tracking, scene understanding, and edge AI deployment

1.3 Problem Statement:

In the modern digital era, the ability of machines to visually perceive and interpret their surroundings in real-time has become critically important across a wide range of applications—from autonomous vehicles and security surveillance to robotics and industrial automation. Traditional object detection methods based on handcrafted features and classical image processing techniques often suffer from poor generalization, low accuracy, and high latency, especially when dealing with complex scenes containing multiple, overlapping, or occluded objects.

The primary challenge lies in developing a system that can not only detect and classify multiple objects within an image or video frame but also do so accurately and in real-time, under varying conditions such as different lighting, camera angles, and backgrounds. This necessitates a robust and efficient approach that can learn complex visual patterns and generalize effectively to unseen data.

Convolutional Neural Networks (CNNs) have demonstrated remarkable success in solving image classification and object recognition problems. However, implementing CNNs for real-time object detection involves addressing key issues such as:

- High computational cost,
- The trade-off between detection speed and accuracy,
- Efficient localization of multiple objects in a single pass,
- Scalability to diverse object classes and environmental conditions.

Therefore, the core problem this project aims to address is:

"How can we design and implement a real-time object detection system using convolutional neural networks that can accurately identify and localize multiple objects in diverse environments, while maintaining a balance between computational efficiency and detection precision?"

This problem demands a solution that combines the power of deep learning, optimized model architectures, and real-time processing capabilities to deliver a practical, scalable, and intelligent object detection system suitable for deployment in real-world scenarios

1.4 Aims and Objectives:

The aim of this project is to design and develop a real-time object detection system using Convolutional Neural Networks (CNNs) that can accurately identify, classify, and localize multiple objects within images or live video streams. The system should be capable of processing data efficiently with minimal latency, making it suitable for real-world applications such as surveillance, autonomous navigation, smart cities, and robotics.

This project seeks to achieve a balance between detection accuracy and processing speed by implementing and optimizing deep learning models such as YOLO, SSD, or Faster R-CNN, and deploying them in a way that enables real-time performance across a wide range of environments and object types.

CHAPTER-2 LITERATURE REVIEW

Real-time object detection has become a significant area of research in computer vision due to its wide range of practical applications including autonomous driving, video surveillance, robotics, and augmented reality. Over the years, advancements in deep learning, especially in Convolutional Neural Networks (CNNs), have revolutionized the way machines interpret visual data, moving from traditional methods of object detection to more intelligent, accurate, and real-time capable systems.

2.1 Traditional Approaches:

Early object detection techniques, such as Haar Cascades and Histogram of Oriented Gradients (HOG) with Support Vector Machines (SVMs), relied heavily on manual feature extraction. While effective to a certain extent, these methods were computationally expensive and lacked scalability, particularly in real-time applications or scenes with complex object variations.

2.2 Deep Learning-Based Detection Models:

The introduction of R-CNN (Region-based Convolutional Neural Network) by Girshick et al. marked a turning point in object detection. It significantly improved accuracy by applying CNNs to object proposals, but its processing time was too slow for real-time applications. This led to the development of Fast R-CNN and Faster R-CNN, which improved both speed and accuracy by integrating region proposal networks.

However, real-time detection required even faster models, giving rise to Single Shot Detectors (SSD) and You Only Look Once (YOLO) frameworks. YOLO, introduced by Redmon et al., redefined the approach by framing detection as a regression problem, enabling detection and classification in a single forward pass. YOLOv3, YOLOv4, and YOLOv5 further improved detection accuracy and inference speed, making them ideal for real-time applications. Similarly, SSD provided a good trade-off between speed and precision using multi-scale feature maps and default anchor boxes.

2.3 Real-Time Processing with Optimization:

To achieve true real-time performance, research has focused on model compression, quantization, and hardware-level optimizations. Frameworks like TensorRT, ONNX Runtime, and hardware accelerators like GPUs or TPUs are commonly used to meet real-time demands. Moreover, OpenCV has played a crucial role in handling real-time video streams and rendering outputs efficiently.

CHAPTER-3
MODAL IMPLEMENTATION AND ANALYSIS

3.1 INTRODUCTION:

The development of a real-time object detection system using Convolutional Neural Networks (CNNs) follows a structured and phased methodology, as outlined below:

3.1. Problem Understanding and Requirement Analysis

- Define the real-time object detection problem clearly.
- Identify use cases (e.g., surveillance, vehicle detection, etc.).
- Determine performance criteria such as detection speed (FPS), accuracy (mAP), and supported object classes.

3.2. Dataset Collection and Preprocessing

- **Data Sources**: Use public datasets like **COCO**, **PASCAL VOC**, or create a custom dataset using labeled images or video frames.
- **Annotation**: Label images with bounding boxes and class names using tools like **LabelImg** or **MakeSense.ai**.
- **Preprocessing Steps**:
    - Resize images to a uniform size.
    - Normalize pixel values.
    - Perform data augmentation (e.g., flipping, rotation, brightness adjustment) to improve generalization.

3.3. Model Selection and Architecture Design

- Select an appropriate CNN-based object detection model based on the trade-off between speed and accuracy:
    - **YOLOv5 or YOLOv8** (preferred for real-time applications).
    - **SSD (Single Shot Detector)**.
    - **Faster R-CNN** (more accurate but slower).
- Understand the chosen model's architecture:
    - **Backbone**: Feature extraction (e.g., Darknet, ResNet).
    - **Neck**: Feature fusion (e.g., FPN).
    - **Head**: Object classification and bounding box regression.

3.4. Model Training

- Split the dataset into training, validation, and testing sets (e.g., 70:20:10).
- Use a deep learning framework such as **TensorFlow**, **Keras**, or **PyTorch**.
- Set training parameters:
    - Epochs, batch size, learning rate, optimizer (e.g., Adam, SGD).
- Apply techniques like early stopping and learning rate decay to avoid overfitting.
- Monitor training using metrics like loss, mAP (mean Average Precision), and IoU (Intersection over Union).

3.5. Model Evaluation

- Evaluate model performance on the test set using:

    - **Precision**, **Recall**, **F1-Score**.
    - **IoU**: Accuracy of predicted bounding boxes.
    - **mAP**: Average precision across all classes.
- Test inference time (FPS) to ensure real-time capability.

3.6. Optimization for Real-Time Performance

- Techniques used:
    - **Model pruning** and **quantization** to reduce model size.
    - Use of **GPU acceleration** (via CUDA) or **TensorRT** for deployment.
    - Implement **lighter versions** like Tiny-YOLO if needed for edge devices.

3.7. System Integration and Visualization

- Integrate the model into a real-time application using:
    - **OpenCV** for video stream handling.
    - **Flask** or **Streamlit** for web-based interface (optional).
- Display:
    - Live bounding boxes on detected objects.
    - Object class and confidence scores.

3.8. Testing and Validation

- Test the system in different environments (lighting, object types, distance).
- Validate system performance in real-time scenarios.

3.9. Documentation and Report Preparation

- Document each phase, including code, architecture diagrams, and testing results.
- Prepare visual samples (input frames with output bounding boxes) and performance graphs

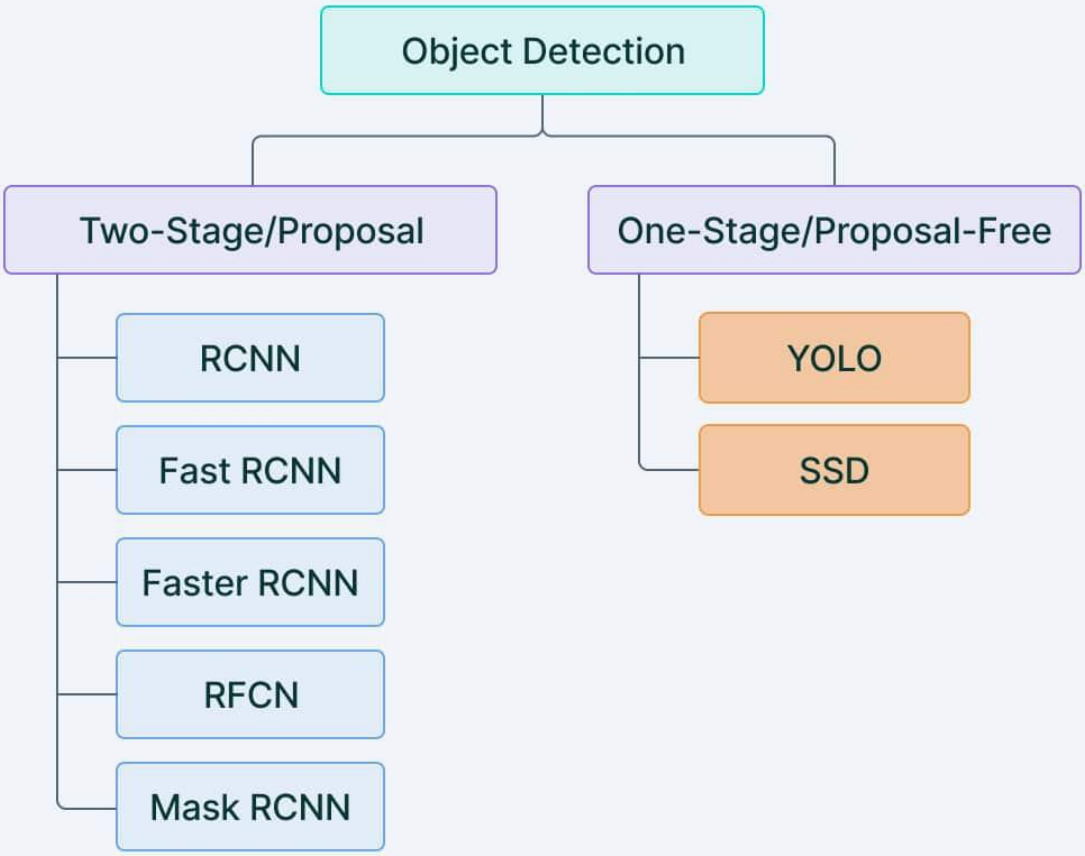
The main goal of this project is to develop a system capable of detecting multiple objects in real time from video input or camera feed, utilizing deep learning-based CNN models. The system aims to deliver high accuracy, low latency, and efficient processing suitable for real-world applications such as surveillance, autonomous vehicles, and smart cities.

In the rapidly evolving field of computer vision, real-time object detection has emerged as a crucial application with widespread use in areas such as surveillance, autonomous vehicles, robotics, and healthcare. Object detection involves identifying and locating objects within an image or video frame. Traditional object detection methods relied heavily on manual feature extraction and classical machine learning algorithms, which often struggled with accuracy, adaptability, and processing speed.

With the advancement of deep learning, particularly Convolutional Neural Networks (CNNs), object detection systems have become significantly more efficient and reliable. CNNs automatically extract and learn relevant features from raw image data, enabling models to accurately detect multiple objects of varying sizes and shapes in real-time.

This project focuses on analyzing the need for a robust object detection system capable of processing live video streams or camera feeds using CNN-based models. The analysis aims to evaluate the limitations of existing systems and establish the foundation for a proposed system that offers real-time performance, higher accuracy, and greater scalability for practical implementation

One and two stage detectors



V7 Labs

Description:

Traditional object detection systems use classic machine learning approaches and manual feature extraction techniques such as:

- Haar Cascades
- Histogram of Oriented Gradients (HOG) + SVM
- Viola-Jones Algorithm

These systems were designed primarily for static image detection and lack the robustness required for real-time applications.

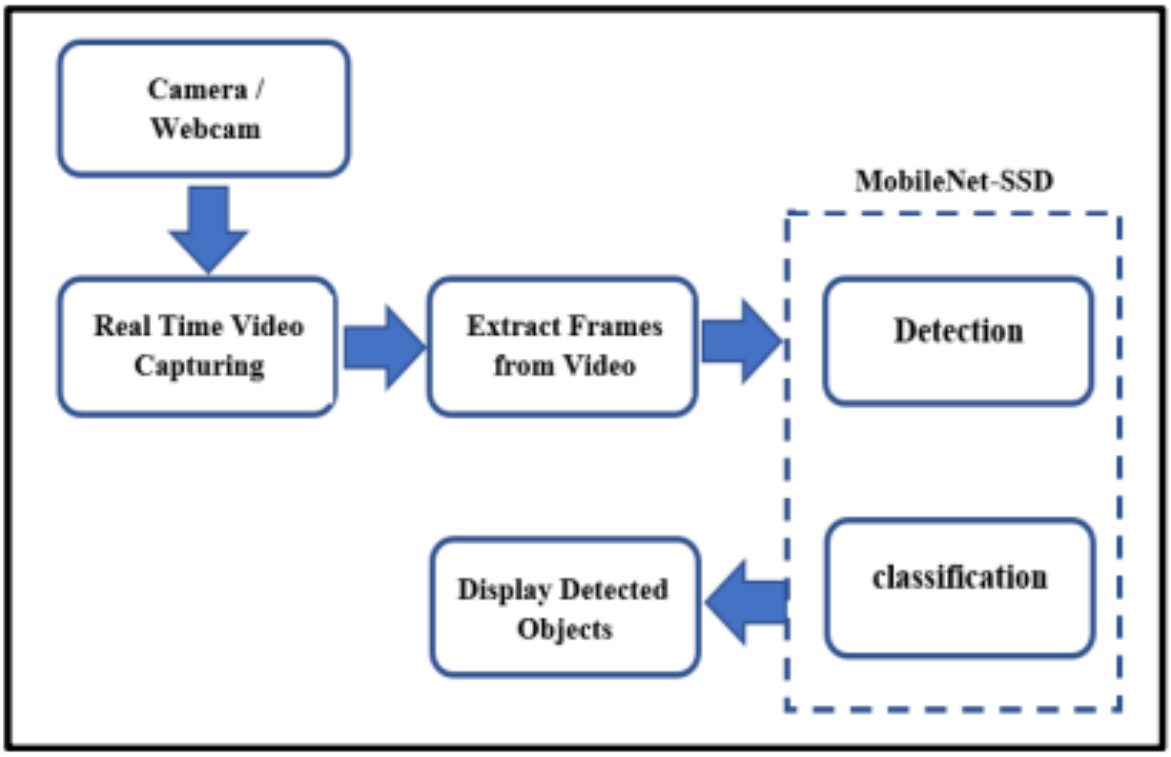
The existing methods for object detection using CNN. This may include deep learning algorithms like R-CNN and YOLO R-CNN: RCNN stands for “Region-based Convolutional Neural Network”. It's a kind of deep learning model for identifying objects in pictures. It first generates region proposals. RCNN, region proposals are generated using a selective search algorithm. This algorithm analyzes the image and identifies potential object regions based on similarities in color, texture, and other visual features.

These proposed regions are then passed through the convolutional neural network for further analysis and classification. It's a popular approach in computer vision. YOLO: YOLO stands for “YOU ONLY LOOK ONCE”. It is a real-time object detection system that uses a single neural network to process the entire image, segmenting it into areas and predicting possibilities and bounding boxes for each one. This indicates that Yolo can recognize several objects in a video. Volume 9, Issue 4, April – 2024 International Journal of Innovative Science and Research Technology.

Drawbacks of Existing Systems:

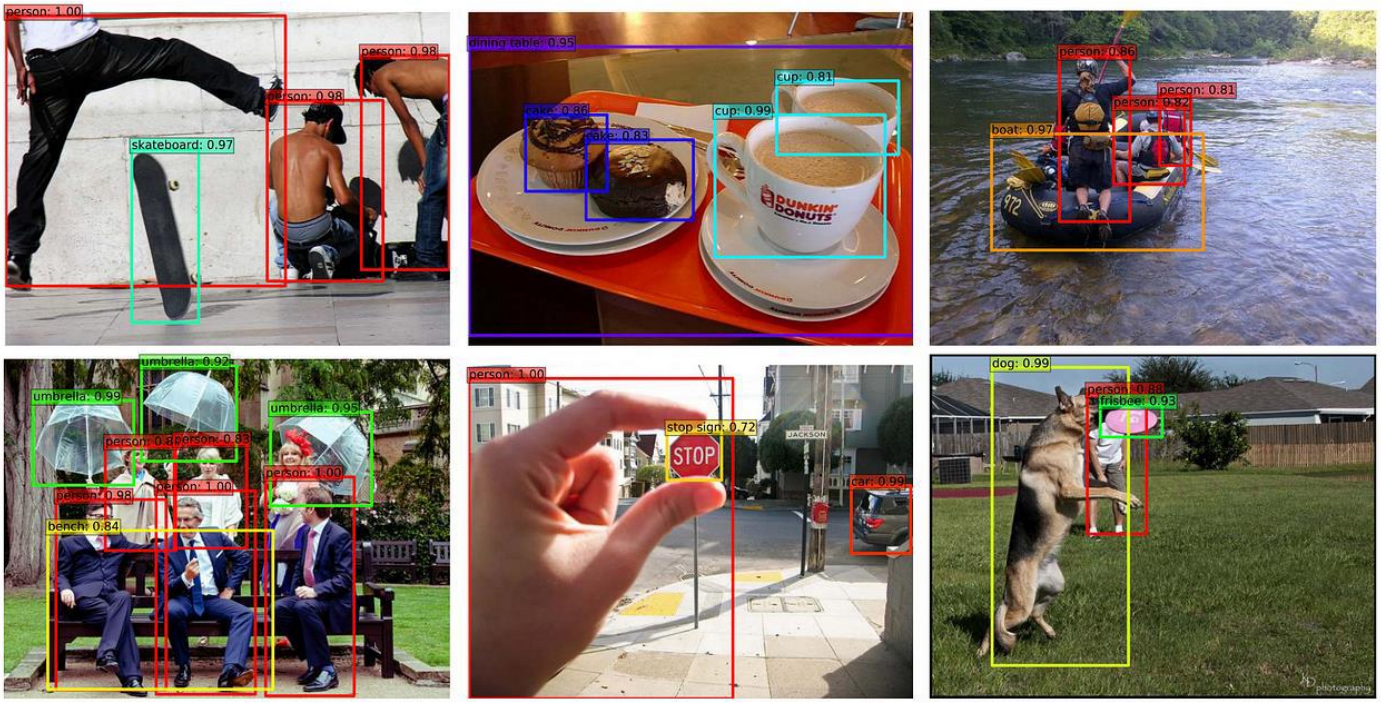
R-CNN couldn’t pushed real time speed though its system is updated and new versions of it are deployed. One of the main drawback of YOLO is it struggle to detect objects grouped close together, especially smaller ones.

The proposed system aims to overcome the limitations of traditional object detection mechanisms by integrating deep learning models based on Convolutional Neural Networks (CNNs). This system is designed to detect and classify multiple objects in real-time from a live video feed or camera stream. The core idea revolves around using CNN-based object detectors such as YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), and Faster R-CNN, which have demonstrated superior accuracy and speed.



Process of Proposed System

These models eliminate the need for manual feature extraction by learning spatial hierarchies of features directly from input images. By doing so, the system is capable of understanding and interpreting scenes at multiple scales and complexities, making it highly suitable for dynamic environments such as surveillance, traffic monitoring, autonomous navigation, and industrial automation



System Workflow

The proposed system comprises the following stages:

Input Capture: Real-time video stream or static images are fed into the system via a webcam or IP camera.
Preprocessing: The input frames are resized, normalized, and sometimes enhanced (e.g., noise reduction, sharpening) to match the input specifications of the CNN model.

Model Inference: A pre-trained or custom-trained CNN model processes the frame and identifies objects within it by drawing bounding boxes and classifying each region.

Post-Processing: Techniques like Non-Maximum Suppression (NMS) are applied to refine detections by removing overlapping boxes.

Output Display: Detected objects are displayed on the screen with bounding boxes and labels, often with real-time confidence scores.

Features of the Proposed System

End-to-End Detection Pipeline: Fully automated, requiring no manual feature engineering or separate classification.
Real-Time Processing Capability: Models like YOLOv4 and YOLOv5 process 30–60 FPS on standard GPUs.

Multi-Class Detection: Can detect and classify hundreds of object types simultaneously.
Scalability: Easily upgradable to support additional object classes or use cases with minimal retraining.
Hardware Compatibility: Optimized for use on GPUs, TPUs, and even edge devices like Raspberry Pi and Jetson Nano.

Model Architecture Overview
Most modern real-time object detectors follow a unified architecture:

- **Backbone**: A CNN-based feature extractor (e.g., ResNet, CSPDarknet, VGG).
- **Neck**: Enhances features using modules like FPN (Feature Pyramid Network) or PANet.
- Head: Outputs bounding boxes, object confidence scores, and class probabilities.

Each layer learns increasingly abstract features — from edges and textures to object parts and entire objects — enabling the network to identify and locate objects with remarkable accuracy.

Deep Learning Models Considered

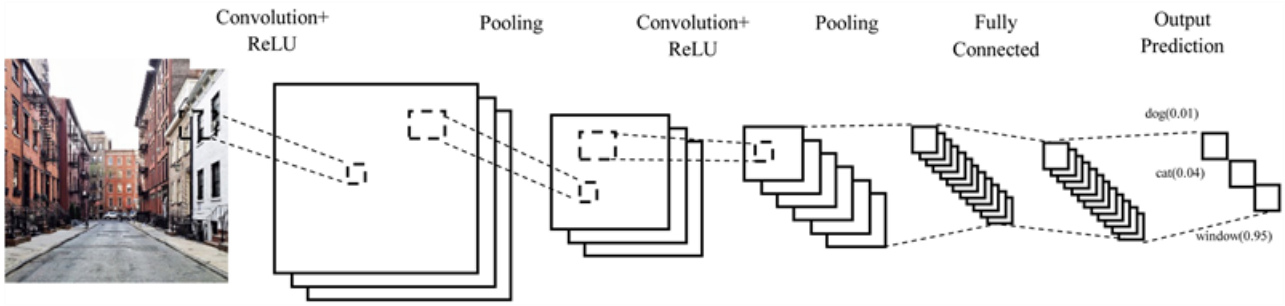
- **YOLO (You Only Look Once)**: Divides the image into grids and predicts bounding boxes and class probabilities directly.
- **SSD (Single Shot Detector)**: Detects objects at multiple scales and aspect ratios using feature maps at different stages.
- **Faster R-CNN**: Combines region proposal networks (RPN) with CNN-based classification, known for high accuracy albeit slower than YOLO.

Advantages Over Traditional Systems Parameter	**Traditional System**	**Proposed CNN-Based System**
Feature Extraction	Manual	Learned automatically
Processing Speed	Slow	Real-Time (30+ FPS)
Accuracy	Low to Medium	High (up to 80–90% mAP)
Adaptability	Low	High (transfer learning supported)
Multi-Object Support	Limited	Full multi-class object detection
Robustness	Sensitive to environment	Stable across lighting and angles

System Design

Convolutional Layer

The convolutional layer is the core layer to construct a convolutional neural network, which generates most of the calculations in the network. Note that the amount of calculation is not the number of parameters. The convolution operation can effectively reduce the training complexity of the network model and reduce the network connection and parameter weights, which makes it easier to train than a fully connected network of the same scale. Common convolution operations are as follows: ordinary convolution, transposed convolution, hole convolution, and depth separable convolution.



Architecture of CNN framework

Common Object Detection Network Model

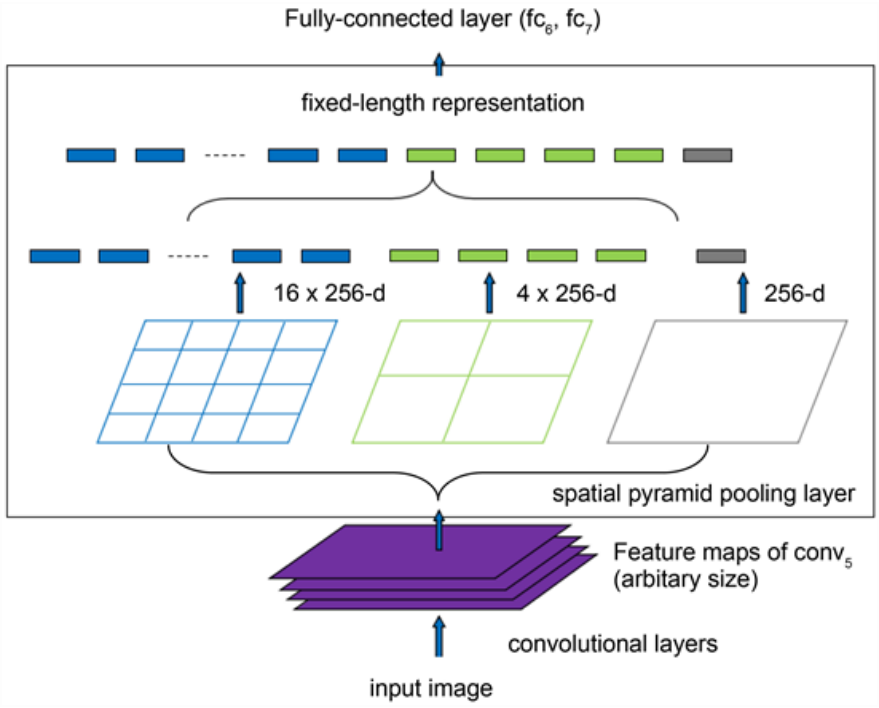
SPPNet

In 2015, SPPNet was published on IEEE. In R-CNN, to generate a vector of equal dimensions for all candidate regions, the candidate regions are forcibly scaled, which will destroy the

proportional relationship of the image, which is not good for feature

extraction, and this extraction process is quite time-consuming, so SPPNet is optimized

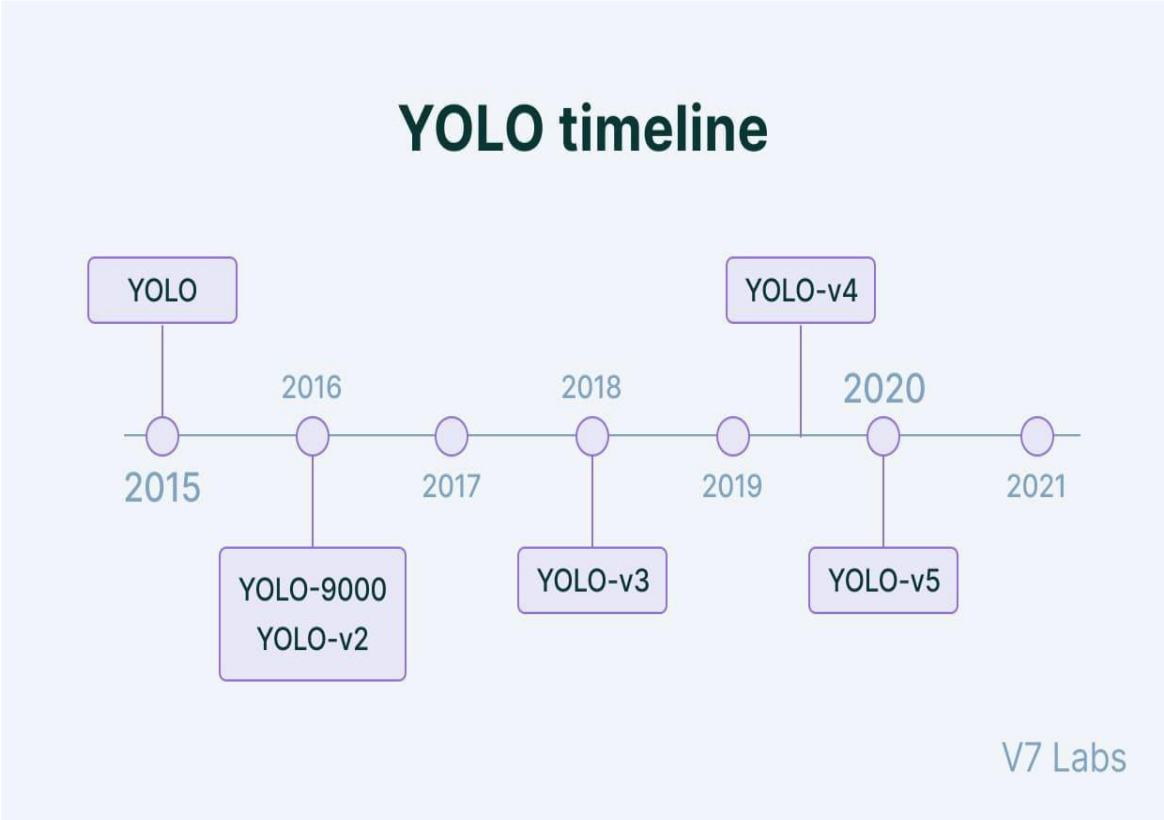
here, using spatial pyramid pooling



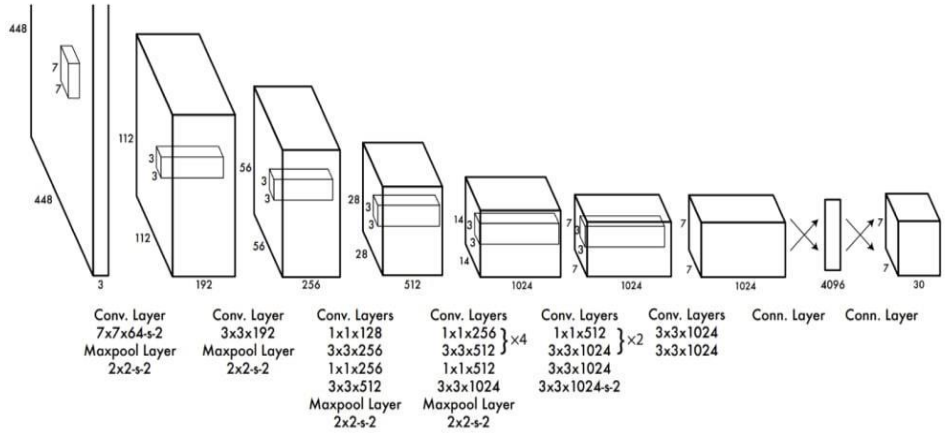
The architecture of the SPPNet framework.

You Only Look Once (YOLO)

YOLO falls under single-stage object detection models and widely used for real-time object detection task and introduced by Redmon et al. It generates the bounding boxes and class predictions in single evaluation. It is widely known as unified network and very fast compared to Faster R-CNN and runs using single convolutional neural network. The CNN used in YOLO is based on GoogLeNet model originally and the updated version is called DarkNet based on VGG. It splits the input image into a grid of cells, where each cell directly classifies the object and predicts a bounding box. As a result, there are large numbers of bounding boxes generated that are integrated to a final prediction.



YOLO ARCHITECTURE



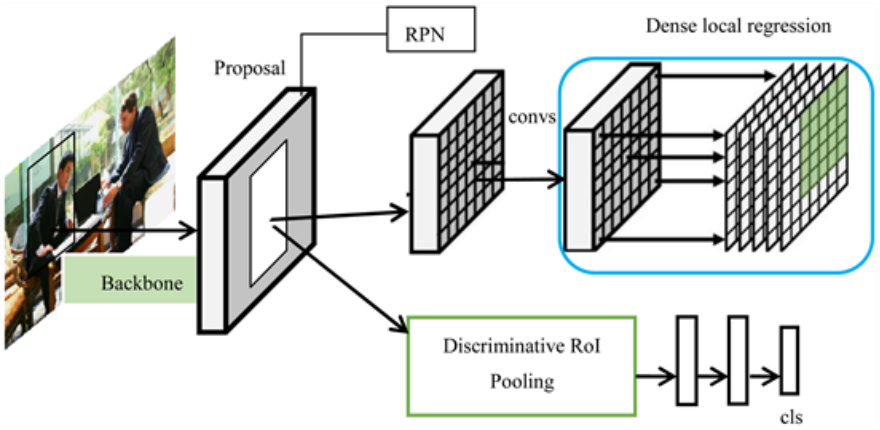
The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

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 how accurate it thinks the predicted box is.

The variations in YOLO are YOLOv1, YOLOv2, and YOLOv3, where YOLOv3 is the latest version. YOLO is a fast and good for real-time object detection tasks. It is possible to train it end-to-end for accuracy improvisation as it uses a single CNN for prediction. It is more generalized and performs well with generalization of natural and artwork images.

YOLOv1

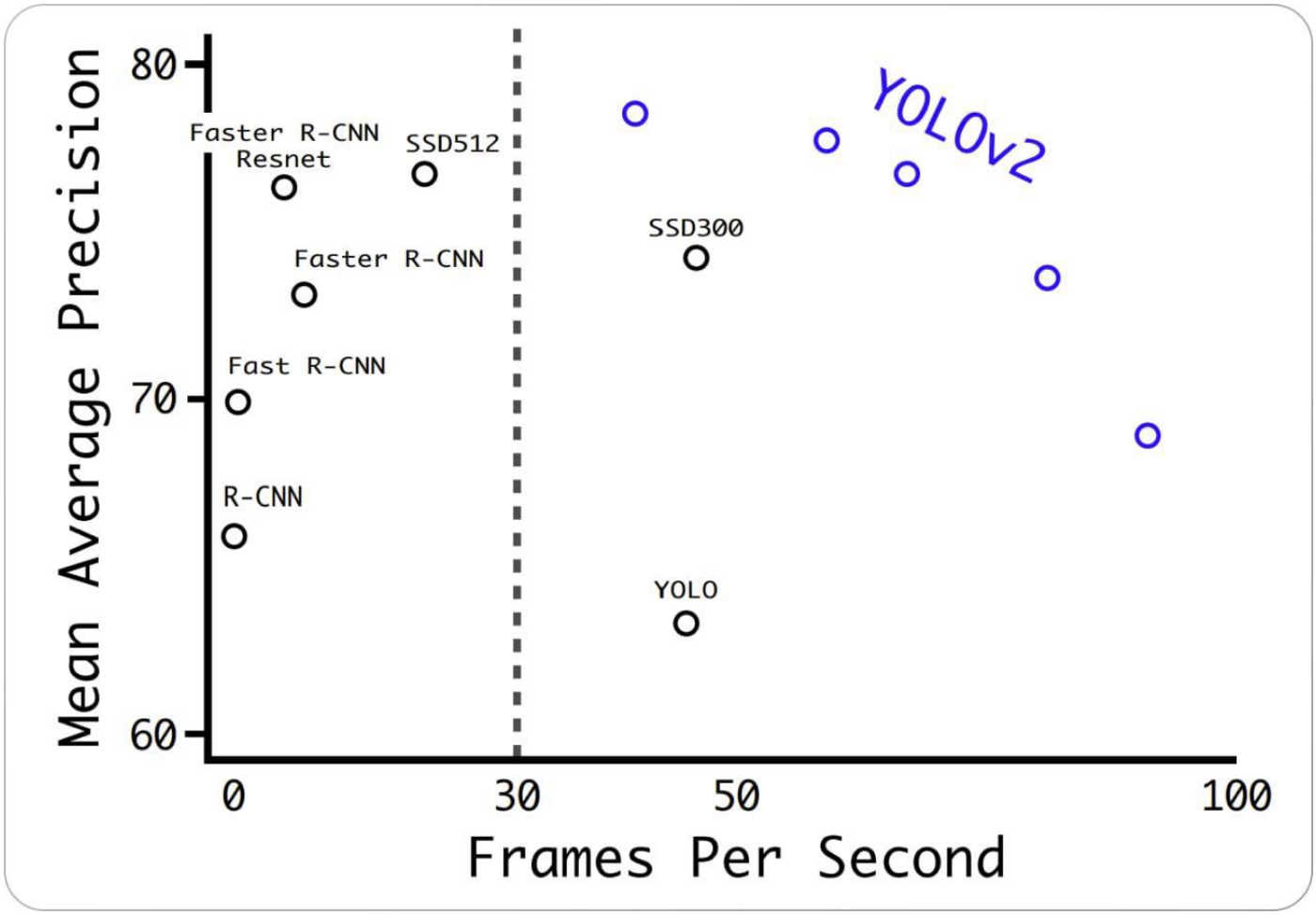
The first step of YOLO is to divide the picture. It divides the picture into grids, and the size of each grid is equal. The core idea of YOLO is to turn object detection into a regression problem, using the entire image as the input of the network, and only going through a neural network to get the location of the bounding box and its category. Its detection speed is extremely fast, the generalization ability is strong, the speed is provided, and the accuracy is reduced. The disadvantage is that for small objects, overlapping objects cannot be detected.



YOLOv2

Compared with YOLOv1, which uses the fully connected layer to directly predict the coordinates of the Bounding Box, YOLOv2 draws on the idea of Faster R-CNN and introduces the Anchor mechanism. The K-means clustering method is used to cluster and calculate a better Anchor template in the training set, which greatly improves the recall rate of the algorithm. At the same time, combining the fine-grained features of the image, the shallow features are connected with the deep features, which is helpful for the detection of small-scale targets.

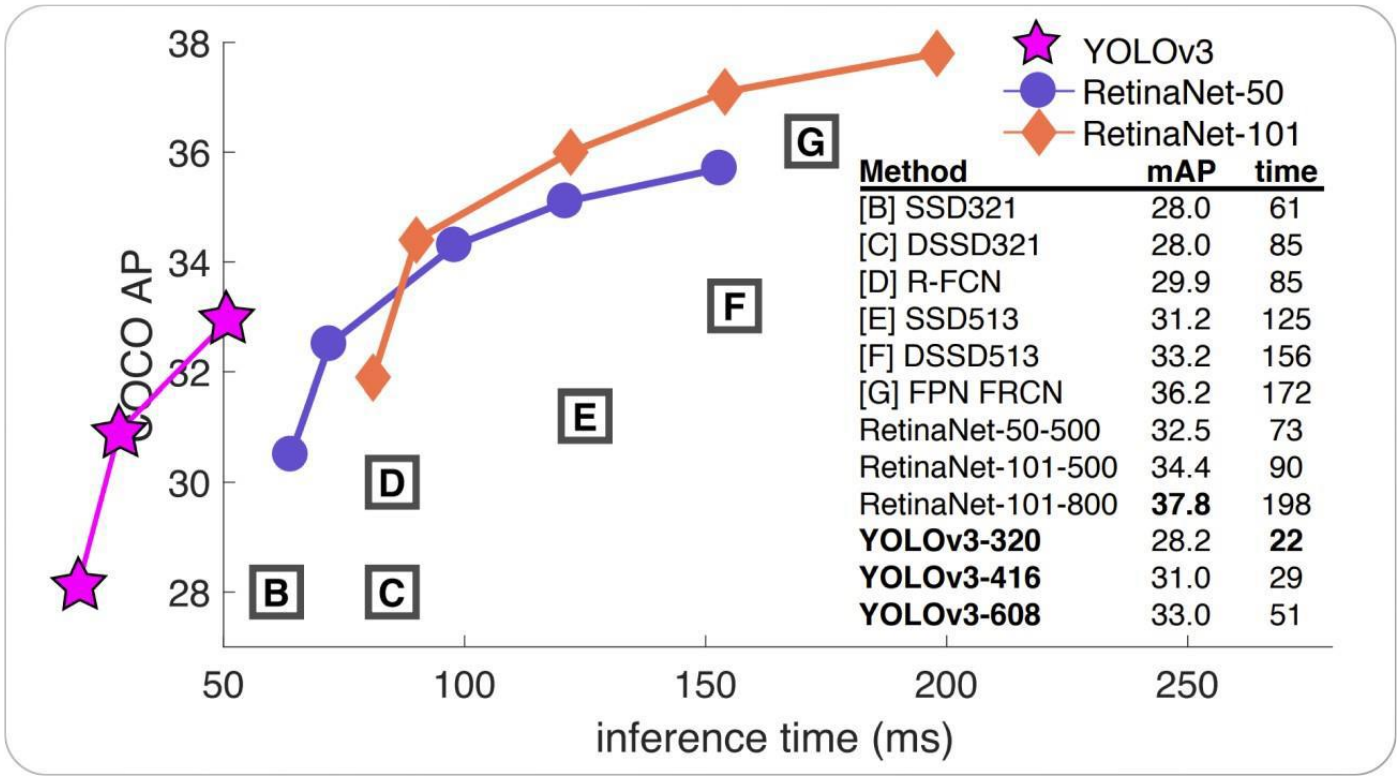
The article proposes a new training method—a joint training algorithm. This algorithm can mix these two data sets. Use a hierarchical view to classify objects, and use a huge amount of classification data set data to expand the detection data set, thereby mixing two different data sets.



YOLOv3

YOLOv3's prior detection system reuses the classifier or locator to perform detection tasks. They apply the model to multiple locations and scales of the image. Those areas with higher scores can be regarded as the test results. In addition, compared to other object detection methods, they use a completely different method. They apply a single neural network to the entire image. The network divides the image into different regions and predicts the bounding box and probability of each region.

These bounding boxes are weighted by the predicted probability. The model has some advantages over classifier-based systems. It looks at the entire image during the test, so its prediction uses the global information in the image. Unlike R-CNN, which requires thousands of single target images, it makes predictions through a single network evaluation. This makes YOLOv3 very fast, generally, it is 1000 times faster than R-CNN and 100 times faster than Fast R-CNN.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

YOLOv4

In 2020, Bochkovskiy and others launched YOLOv4. YOLOv4 conducted a lot of tests on some commonly used Tricks in deep learning and finally selected these useful Tricks: WRC, CSP, CmBN, SAT, Mish activation, Mosaic data augmentation, CmBN, DropBlock regularization, and CIoU loss. YOLOv4 adds these practical skills based on traditional YOLO to achieve the best trade-off between detection speed and accuracy.

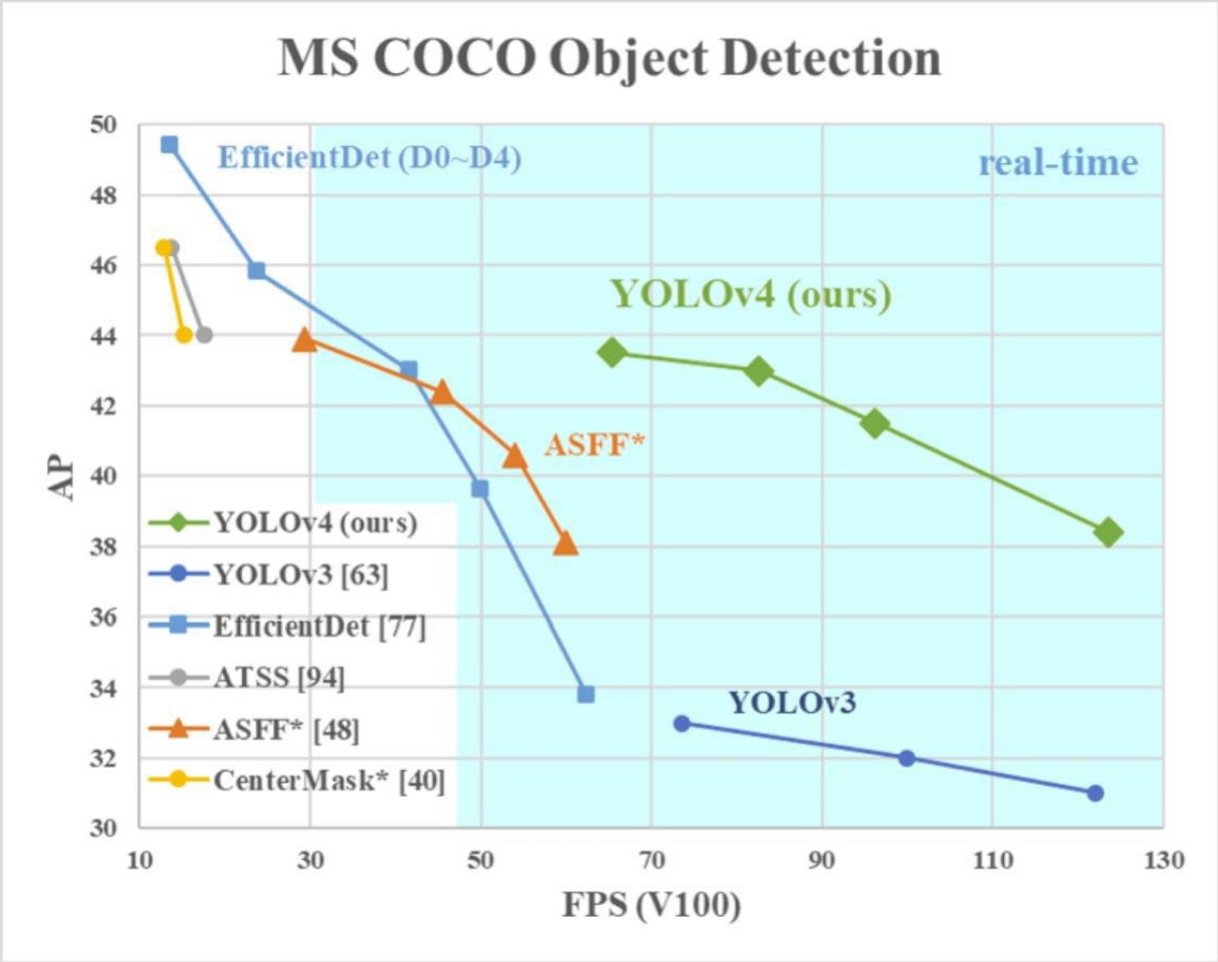
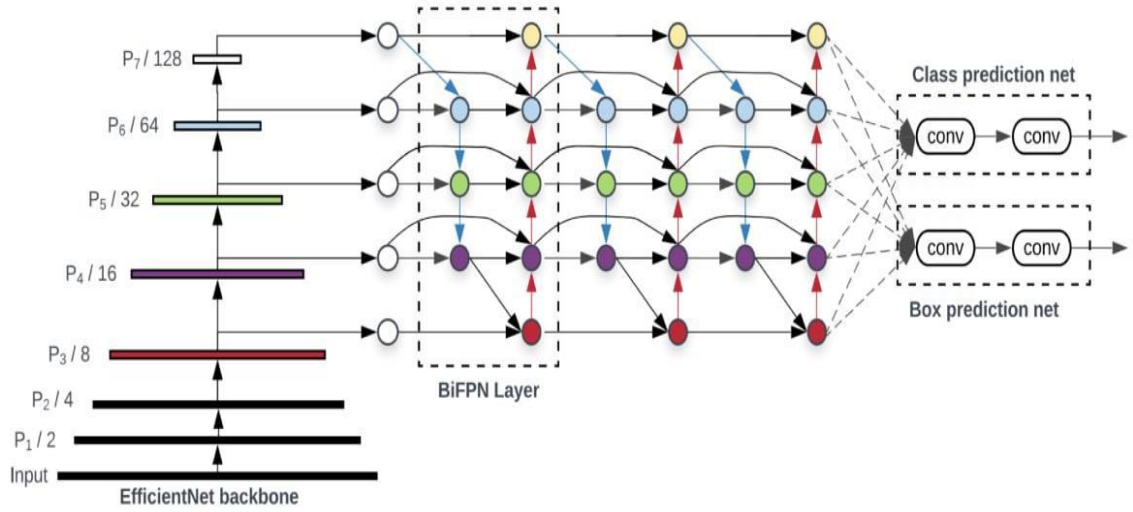


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.

YOLOv5:

YOLOv5 was introduced in 2020 by the same team that developed the original YOLO algorithm as an open-source project and is maintained by Ultralytics. YOLO v5 builds upon the success of previous versions and adds several new features and improvements.

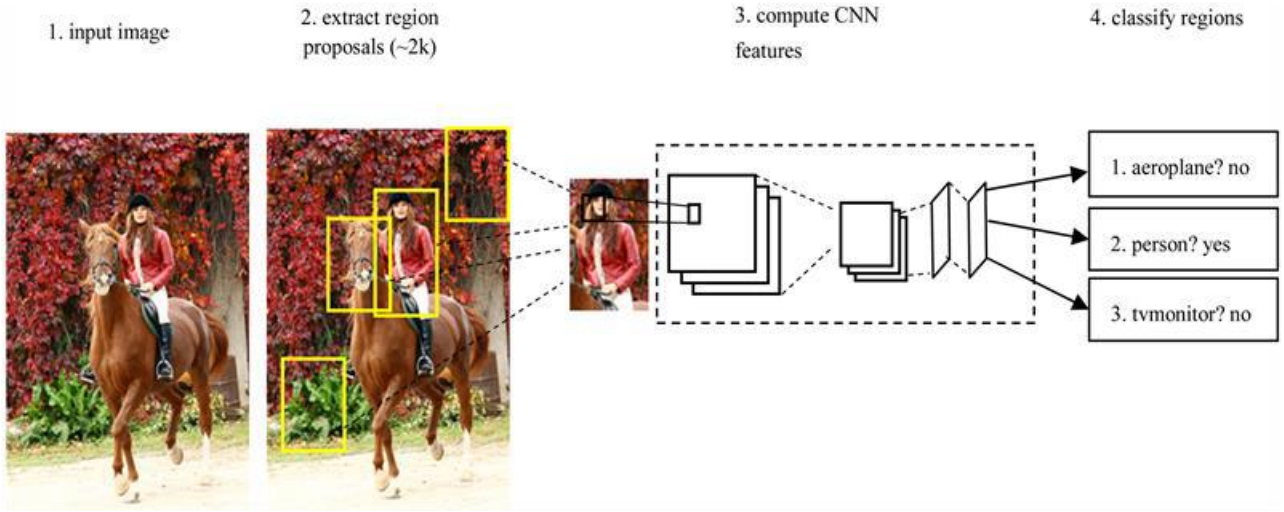
Unlike YOLO, YOLO v5 uses a more complex architecture called EfficientDet (architecture shown below), based on the EfficientNet network architecture. Using a more complex architecture in YOLO v5 allows it to achieve higher accuracy and better generalization to a wider range of object categories.



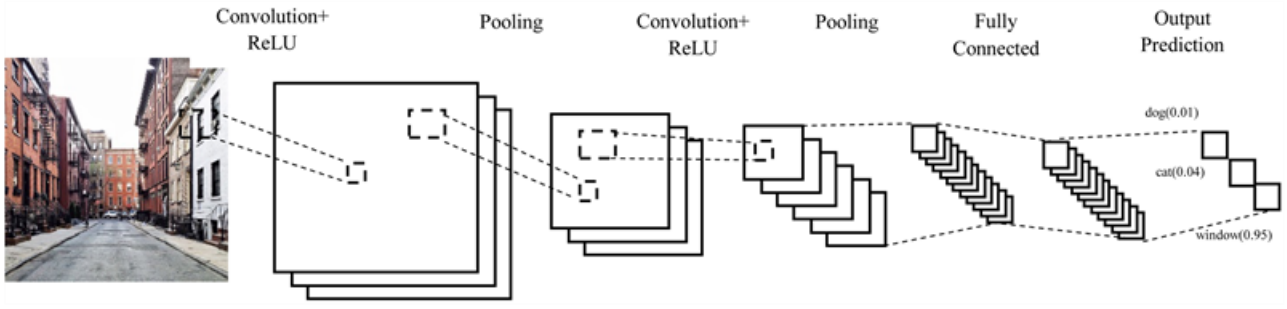
EfficientDet architecture – It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

R-CNN:

RCNN stands for "Region-based Convolutional Neural Network". It's a kind of deep learning model for identifying objects in pictures. It first generates region proposals. RCNN, region proposals are generated using a selective search algorithm. This algorithm analyzes the image and identifies potential object regions based on similarities in color, texture, and other visual features. These proposed regions are then passed through the convolutional neural network for further analysis and classification. It's a popular approach in computer vision.

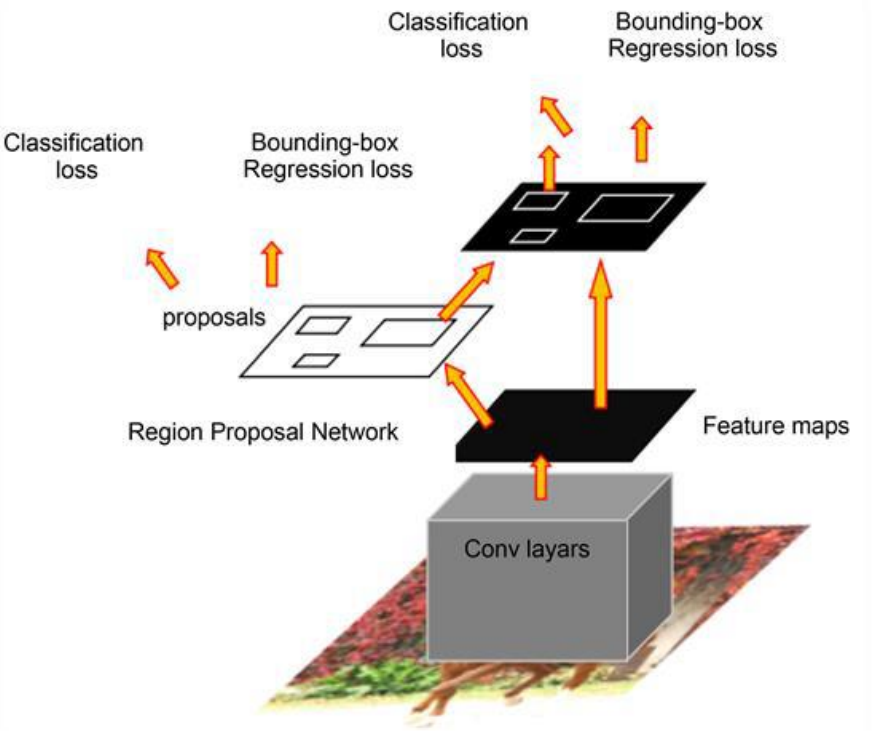


R-CNN, a short form of region-based convolutional neural network, is one of the most widely used object detection model that falls under two-stage object detectors. Girshick et al. proposed R-CNN that is a first region-based CNN detector.
R-CNN uses a selective search method that generates 2000 regions from the image, called region proposals. These regions are input to CNN that produces a 4096-dimensional feature vector as an output. SVM is applied on this generated vector to classify the objects. Moreover, the bounding box is also drawn surrounding to an object.
The major problem with R-CNN is its speed as it is very slow. Also, selective search, the algorithm used to generate the proposals, is very fixed that discards the possibility of learning. The major drawbacks are selective search algorithm proposes 2000 regions per image; for each region of image, it generates CNN feature vector and there is no shared computation between these stages. R-CNN obtained a value of 53.3% of map which is significant improvement over the previous work on PASCAL VOC 2012.



Faster R-CNN:
The problem with Fast R-CNN: There is a bottleneck: selective search to find all candidate boxes, which is also very time-consuming. To obtain these candidate frames more efficiently, Faster R-CNN has added a neural network region proposal network RPN (region proposal network) that extracts edges. After the convolutional neural network is added RPN, the work of finding candidate frames can be completed, the region proposal network as part of the Faster R-CNN model, it is trained together with the entire model. Realizing the integration of candidate frame extraction into the deep network, RPN can learn how to

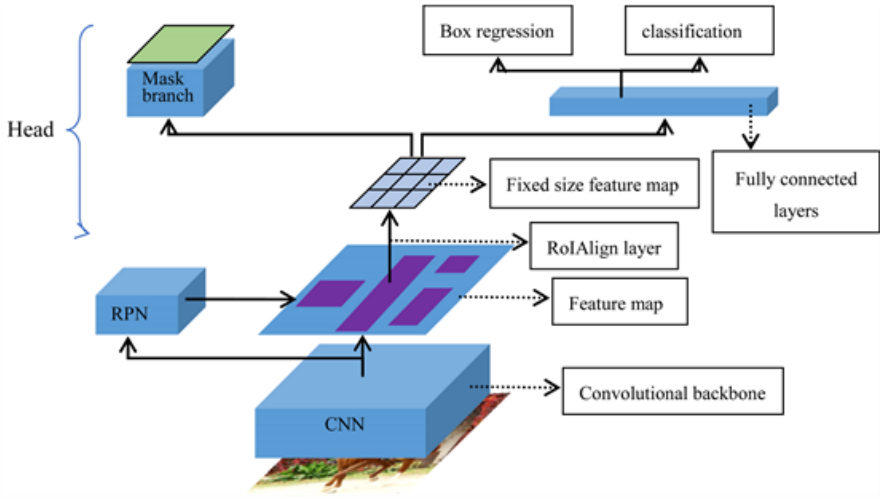
generate high-quality proposed regions, reduce the number of proposed regions learned from the data, and still maintain the accuracy of object detection.



Mask R-CNN

HeKaiming launched Mask R-CNN on ICCV in 2017. Mask R-CNN is an extension of the original Faster-RCNN, adding a branch to use existing detection to predict the target in parallel. At the same time, this network structure is relatively easy to implement and train, and the speed is relatively fast, and it can be easily applied to other fields, such as object detection, segmentation, and key point detection of people.

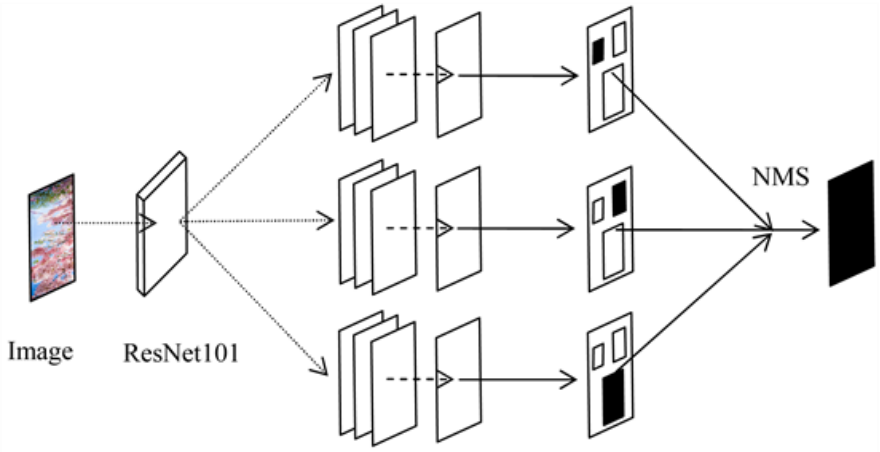
The image is first extracted through ResNet-FPN for feature extraction; then through the RPN network to predict Proposal; then RoI Align is used for feature extraction, then the classification and detection heads, and finally the Mask detection head, that is, each category predicts a Mask.



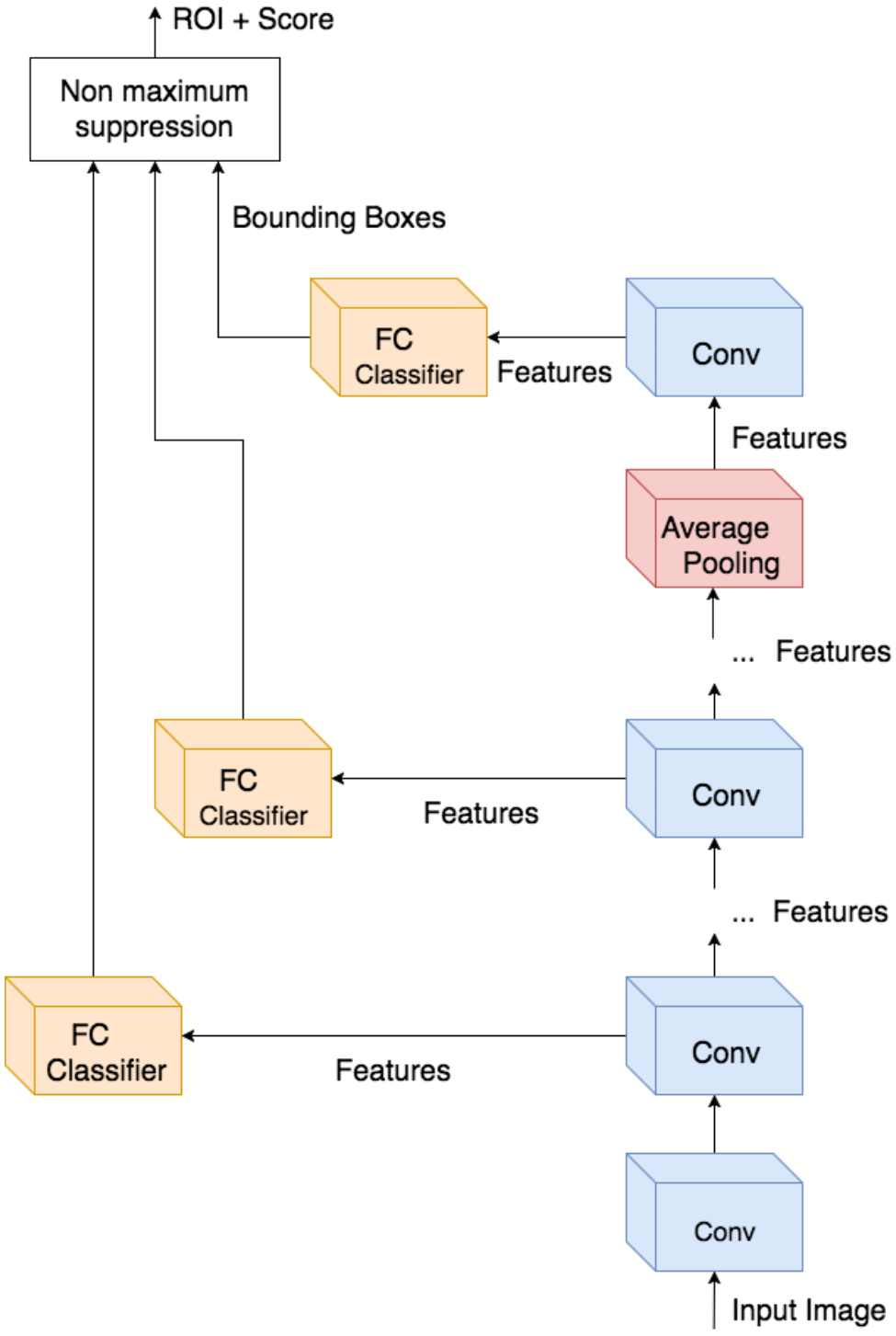
The architecture of Mask R-CNN framework

Trident Net

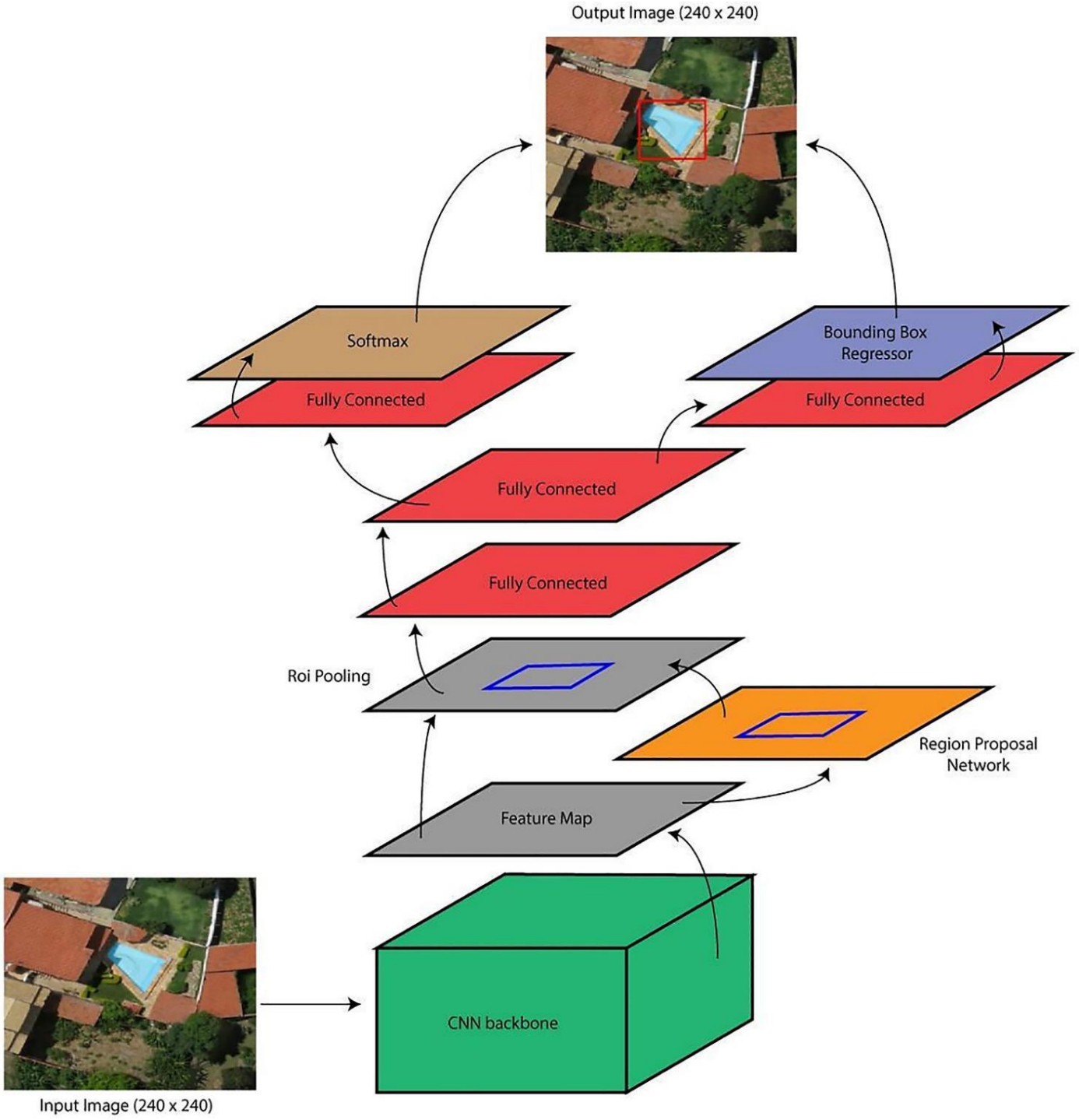
The TridentNet module mainly includes 3 of the same branches, the only difference is the expansion rate of the expanded convolution. From top to bottom, the expansion rates are 1, 2, and 3 respectively, which can detect small, medium, and large targets respectively, which can better realize multi-scale object detection. The three branches share weights.



System Design: Architecture Diagram



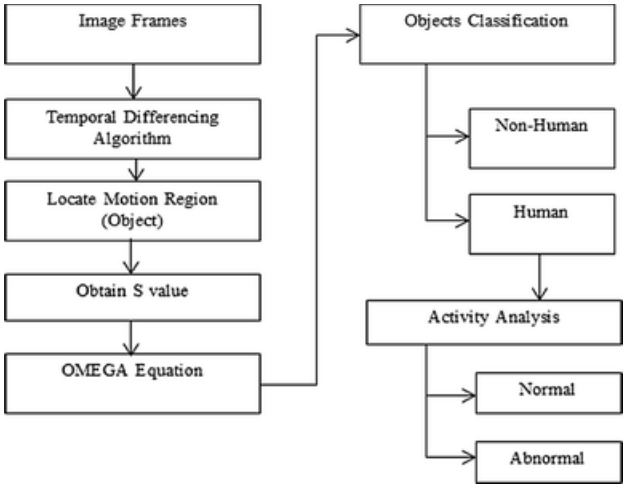
Faster R-CNN Architecture:



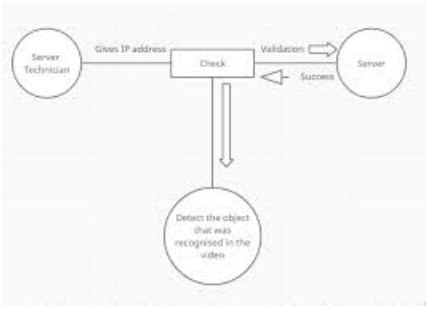
DATA FLOW DIAGRAM:

Data Flow Diagrams

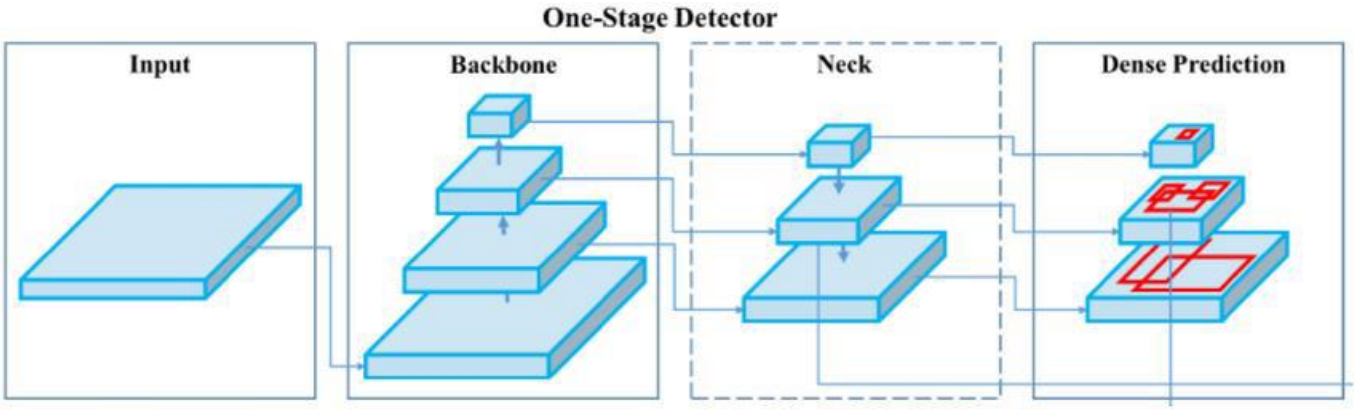
A data-flow diagram (DFD) is a tool to show how data flows through a system or process (usually an information system). The DFD additionally gives details about each entity's inputs and outputs as well as the process itself.



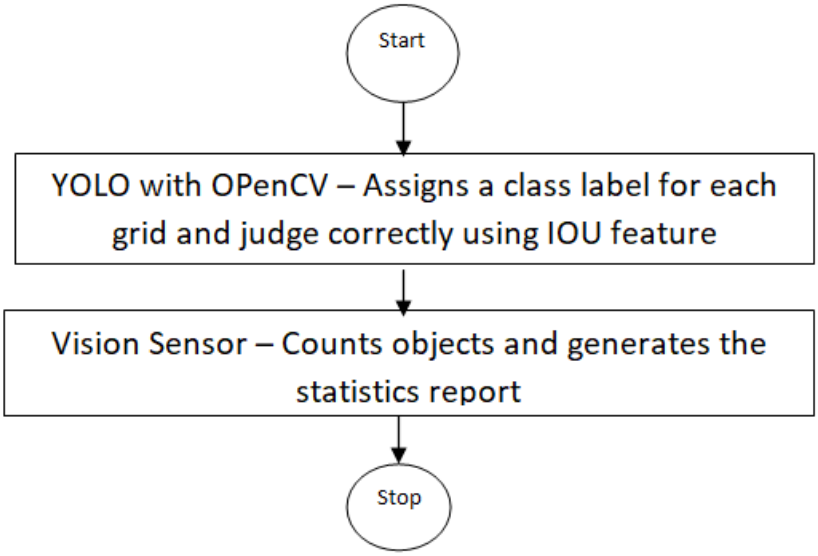
DFD (Level 0) (Level 0) An abstraction level of data flow in this program is depicted



DFD (Level 1) The schema is divided into many BUBBLES/PROCESSES in level-1 DFD.

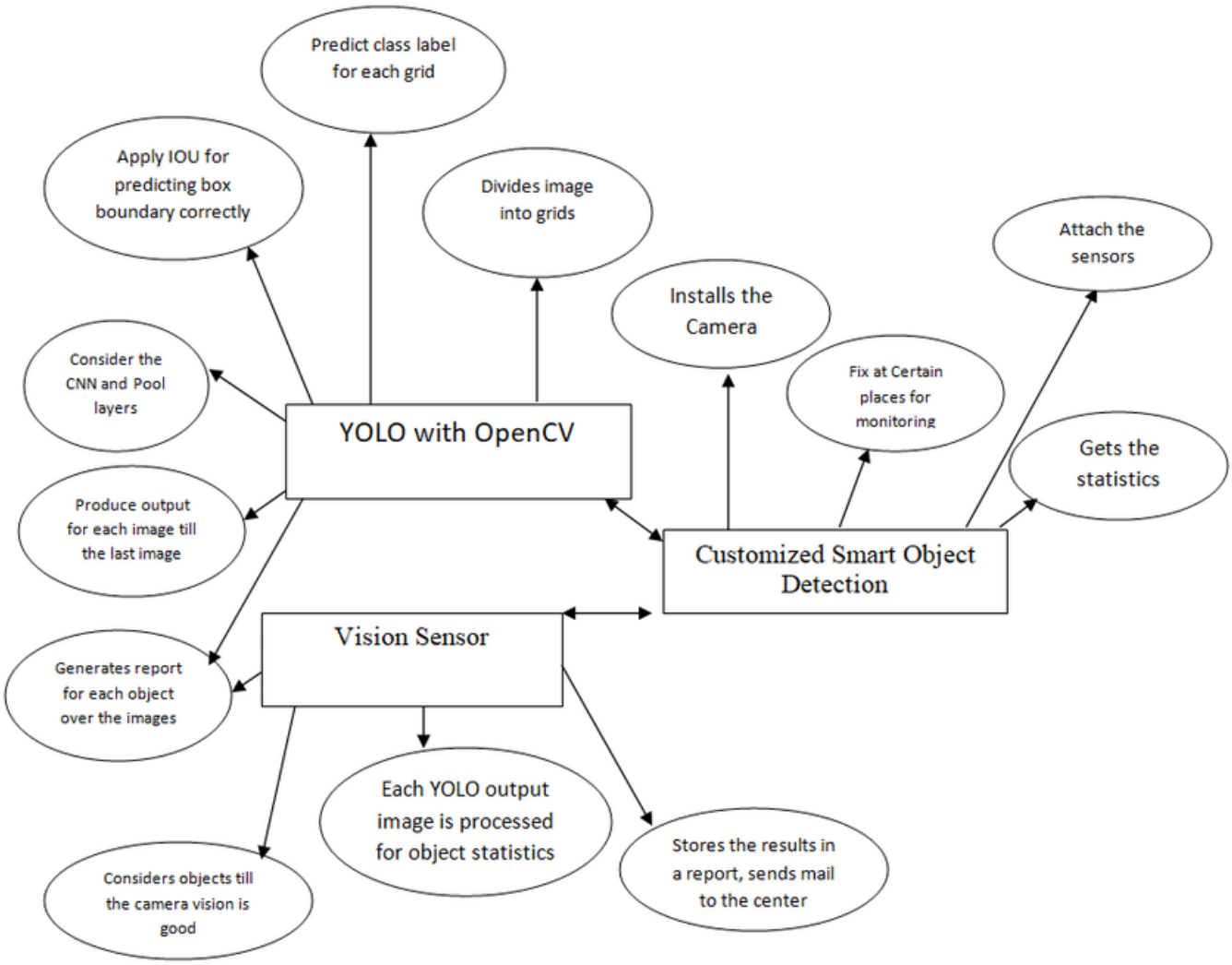


ER-Diagram



Modules in the flowchart of the proposed system

ER Diagram for customized smart object detection



Number of Images identified: 5					
No. of times Object1 is occurred					5
No. of times Object2 is occurred					2
No. of times Object3 is occurred					1
Statistics					
Image No. & its Time stamp	**Image1 10:33AM**	**Image2 10:45AM**	**Image3 12:47PM**	**Image4 5:10PM**	**Image5 5:15PM**
Object1 : 5 times occurred	Yes	Yes	Yes	Yes	Yes
Object2: 2 times occurred	Yes	NA	NA	Yes	NA
Object3: 1 time occurred	NA	Yes	NA	NA	NA

Statistics report of 3 objects using YOLO

This report is stored in the authorized cloud as per license of the organization as well as it is propagated as a mail to authorized member for further analysis. There are 3 variations in the YOLO in which YOLOv1 has faster speed, it have drawbacks of lower precision and localization errors especially for small objects which are resolved in next versions such as v2 and v3.

CHAPTER-4

RESULTS AND DISCUSSION

4.1 Code Implementation:

Model Selection:

- YOLOv5 is recommended for real-time performance due to its balance of speed and accuracy.

Main Modules in the Project:

Input Module

Captures real-time video from a webcam or loads an image/video file.

```
import cv2
# Open webcam (0 is default camera)
cap = cv2.VideoCapture(0)
if not cap.isOpened():
print("Error: Could not open webcam.")
```

Preprocessing Module

Prepares the input image/frame for the CNN model.

```
# Frame is already read by OpenCV
ret, frame = cap.read()
if not ret:
break
# Resize or convert if needed
# frame = cv2.resize(frame, (640, 640))
```

CNN Inference Module (Object Detection with YOLOv5)

Use the YOLOv5 model to detect objects.

```
import torch
# Load YOLOv5 pre-trained model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)
# Run object detection on a frame
results = model(frame)
```

Introduction:

A project work plan allows you to outline the requirements of a project, project planning steps, goals, and team members involved in the project.Within each goal, you're going to outline the necessary Key Action Steps in project planning, the requirements, and who's involved in each action step.

Key Action Step:

- Expected Outcome -Add this as a task. The Expected outcome will be the part of Project
- Assignees – Assigning the work to the team members.
- Completion Date -Add a due date and tries to finish the work within the time

3.2 Work Breakdown Structure:

In order to develop this system, we gave enormous importance to scheduling because we believed if we want to provide the best of quality in a given period of time then we must give due importance to scheduling which also helped us to achieve a better

results.we observe the entire work structure, meaning how the scheduling was maintained throughout the developmental phase. We shall also see the financial foundation of this project and furthermore the feasibility study should be also discussed.

Post-Processing Module

Processes the results from YOLOv5 and visualizes them.

Renders results (bounding boxes and labels)

results.render()

Display result using OpenCV

cv2.imshow('Real-Time Object Detection', results.imgs[0])

5.Display & Output Module

Displays the detection result in a window and allows exit with a keypress.

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

Complete Code – Real-Time Detection using YOLOv5 + OpenCV

import torch

import cv2

Load the YOLOv5 model (small variant)

model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

Open webcam

cap = cv2.VideoCapture(0)

```
while True:

ret, frame = cap.read()

if not ret:

break

# Perform detection

results = model(frame)

# Render results (bounding boxes + labels)

results.render()

# Show the output

cv2.imshow('Real-Time Object Detection', results.imgs[0])

# Exit on pressing 'q'

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release resources

cap.release()

cv2.destroyAllWindows()
```

4.2 Testing & Validation

Create test cases for different scenarios: Test Case	Input Frame Example	Expected Result
TC1	Single person in frame	Label: "person", with bounding box
TC2	Dog, bottle, and chair	Multiple labels: "dog", "bottle", "chair"
TC3	Low-light frame	May miss some detections (test accuracy)
TC4	Frame with unknown object	Either no label or low confidence

CHAPTER-5
CONCLUSION AND FUTURE WORK

October	Study of project related works like face recognition and detection techniques	Completed
November	Study of the Image processing in python and Open Computer Vision	Completed
December	Study of hardware and selection of components	Completed
January	Study of hardware implementation and installation OS	Completed
February	Study related to creating the environments and working platform	Completed
March	Study of packages/tools and installation of packages	Completed
April	Implementation of algorithm in Software.	Completed
May	Implementation of code in hardware	Completed

Table 5.1 Work Plan

Financial Plan

Financial Plan identifies the Project Finance needed to meet specific objectives. The Financial Plan defines all of the various types of expenses that a project will incur (equipment, materials and administration costs) along with an estimation of the value of each expense. The Financial Plan also summarizes the total expense to be incurred across the project and this total expense becomes the project budget. As part of the Financial Planning exercise, a schedule is provided which states the amount of money needed during each stage of the project.

Components	price
Jetson Nano Board	8500
High Definition Camera	2500
Sd Card	500
Hardware Accessories	500
Total	12000

Table 5.2 Financial Plan

Test.py file

```
from ultralytics import YOLO
import cv2

model=YOLO('../YOLO-Weights/yolov8n.pt')
results=model("../Images/2.png", show=True)

cv2.waitKey(0)
```

webcam.py

```
from ultralytics import YOLO
import cv2
import math

# Initialize webcam
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("❌ Error: Cannot access the webcam.")
    exit()

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# noinspection SpellCheckingInspection
out = cv2.VideoWriter('output.avi', cv2.VideoWriter_fourcc(*'MJPG'),
                      10,
                      (frame_width, frame_height))
```

```
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", "potted-plant", "bed",
              "dining-table", "toilet", "tv-monitor", "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()
    if not success or img is None:
        print("⚠️ Warning: Frame not received from webcam. Skipping...")
        continue

    results = model(img, stream=True)

    for r in results:
        boxes = r.boxes
        for box in boxes:
            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)

            conf = math.ceil((box.conf[0] * 100)) / 100
            cls = int(box.cls[0])
            class_name = classNames[cls]
            label = f'{class_name} {conf}'

            t_size = cv2.getTextSize(label, 0, fontScale=1, thickness=2)[0]
            c2 = x1 + t_size[0], y1 - t_size[1] - 3

            cv2.rectangle(img, (x1, y1), c2, [255, 0, 255], -1, cv2.LINE_AA)
            cv2.putText(img, label, (x1, y1 - 2), 0, 1, [255, 255, 255], thickness=1, lineType=cv2.LINE_AA)

    out.write(img)
    cv2.imshow("Image", img)

    if cv2.waitKey(1) & 0xFF == ord('1'):
        break
```

```
cap.release()
out.release()
cv2.destroyAllWindows()
```

FlaskWebcam,py file

```
from flask import Flask, render_template, Response,jsonify,request,session

#FlaskForm--> it is required to receive input from the user
# Whether uploading a video file  to our object detection model

from flask_wtf import FlaskForm


from wtforms import FileField, SubmitField,StringField,DecimalRangeField,IntegerRangeField
from werkzeug.utils import secure_filename
from wtforms.validators import InputRequired,NumberRange
import os


# Required to run the YOLOv8 model
import cv2

# YOLO_Video is the python file which contains the code for our object detection model
#Video Detection is the Function which performs Object Detection on Input Video
from YOLO_Video import video_detection
app = Flask(__name__)

app.config['SECRET_KEY'] = 'muhammadmoin'
app.config['UPLOAD_FOLDER'] = 'static/files'


#Use FlaskForm to get input video file  from user
class UploadFileForm(FlaskForm):
    #We store the uploaded video file path in the FileField in the variable file
    #We have added validators to make sure the user inputs the video in the valid format  and user does
upload the
    #video when prompted to do so
    file = FileField("File",validators=[InputRequired()])
    submit = SubmitField("Run")
```

```
def generate_frames(path_x = ''):
    yolo_output = video_detection(path_x)
    for detection_ in yolo_output:
        ref,buffer=cv2.imencode('.jpg',detection_)

        frame=buffer.tobytes()
        yield (b'--frame\r\n'
                b'Content-Type: image/jpeg\r\n\r\n' + frame +b'\r\n')

def generate_frames_web(path_x):
    yolo_output = video_detection(path_x)
    for detection_ in yolo_output:
        ref,buffer=cv2.imencode('.jpg',detection_)

        frame=buffer.tobytes()
        yield (b'--frame\r\n'
                b'Content-Type: image/jpeg\r\n\r\n' + frame +b'\r\n')

@app.route('/', methods=['GET','POST'])
@app.route('/home', methods=['GET','POST'])
def home():
    session.clear()
    return render_template('indexproject.html')
# Rendering the Webcam Rage
#Now lets make a Webcam page for the application
#Use 'app.route()' method, to render the Webcam page at "/webcam"
@app.route("/webcam", methods=['GET','POST'])

def webcam():
    session.clear()
    return render_template('ui.html')
@app.route('/FrontPage', methods=['GET','POST'])
def front():
    # Upload File Form: Create an instance for the Upload File Form
    form = UploadFileForm()
    if form.validate_on_submit():
        # Our uploaded video file path is saved here
        file = form.file.data
        file.save(os.path.join(os.path.abspath(os.path.dirname(__file__)),
app.config['UPLOAD_FOLDER'],
                               secure_filename(file.filename)))  # Then save the file
        # Use session storage to save video file path
        session['video_path'] = os.path.join(os.path.abspath(os.path.dirname(__file__)),
app.config['UPLOAD_FOLDER'],
                                             secure_filename(file.filename))
    return render_template('videoprojectnew.html', form=form)
```

```
@app.route('/video')
def video():
    #return Response(generate_frames(path_x='static/files/bikes.mp4'), mimetype='multipart/x-mixed-replace; boundary=frame')
    return Response(generate_frames(path_x = session.get('video_path', None)),mimetype='multipart/x-mixed-replace; boundary=frame')

# To display the Output Video on Webcam page
@app.route('/webapp')
def webapp():
    #return Response(generate_frames(path_x = session.get('video_path', None),conf_=round(float(session.get('conf_', None))/100,2)),mimetype='multipart/x-mixed-replace; boundary=frame')
    return Response(generate_frames_web(path_x=0), mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == "__main__":
    app.run(debug=True)
```

Math.py file

```
from ultralytics import YOLO
import cv2
import math

def video_detection(path_x):
    video_capture = path_x
    #Create a Webcam Object
    cap=cv2.VideoCapture(video_capture)
    frame_width=int(cap.get(3))
    frame_height=int(cap.get(4))
    #out=cv2.VideoWriter('output.avi', cv2.VideoWriter_fourcc('M', 'J', 'P','G'), 10, (frame_width,
frame_height))

    model=YOLO("YOLO-Weights/ppe.pt")
    classNames = ['Protective Helmet', 'Shield', 'Jacket', 'Dust Mask', 'Eye Wear', 'Glove', 'Protective
Boots']
    while True:
        success, img = cap.read()
        results=model(img,stream=True)
        for r in results:
            boxes=r.boxes
            for box in boxes:
                x1,y1,x2,y2=box.xyxy[0]
                x1,y1,x2,y2=int(x1), int(y1), int(x2), int(y2)
                print(x1,y1,x2,y2)
                conf=math.ceil((box.conf[0]*100))/100
                cls=int(box.cls[0])
                class_name=classNames[cls]
                label=f'{class_name}{conf}'
                t_size = cv2.getTextSize(label, 0, fontScale=1, thickness=2)[0]
                print(t_size)
                c2 = x1 + t_size[0], y1 - t_size[1] - 3
                if class_name == 'Dust Mask':
                    color=(0, 204, 255)
                elif class_name == "Glove":
                    color = (222, 82, 175)
                elif class_name == "Protective Helmet":
                    color = (0, 149, 255)
                else:
                    color = (85,45,255)
                if conf>0.5:
                    cv2.rectangle(img, (x1,y1), (x2,y2), color,3)
                    cv2.rectangle(img, (x1,y1), c2, color, -1, cv2.LINE_AA)  # filled
                    cv2.putText(img, label, (x1,y1-2),0, 1,[255,255,255], thickness=1,lineType=cv2.LINE_AA)
```

```
        yield img
        #out.write(img)
        #cv2.imshow("image", img)
        #if cv2.waitKey(1) & 0xFF==ord('1'):
            #break
    #out.release()
cv2.destroyAllWindows()
```

Introduction:

The goal of the performance analysis is to quantify the reliability, precision, and responsiveness of the object detection system under varying conditions such as different lighting, object sizes, occlusions, and motion. This process helps in identifying bottlenecks in the model pipeline, validating the model's ability to generalize, and ensuring that the detection is accurate and fast enough for real-time deployment.

Analysis:

In this, the camera is made much powerful in judging the objects that occurred during the frames of its video. This approach is not only taking one object as input but also supports multiple objects at a time. The output to be produced consists of number of times an object is occurred, its time stamp, pointing marker at the place in the video and etc. These statistics to be carried out using significant modules such as YOLO (You only look once) with openCV framework and vision sensor. The first module in its process assigns label correctly to the images. The second module vision sensor counts the objects in each image and other significant details of that occurrence. The output is specified in terms of both the ways such as instantly alerting about object is detected and daily report that is generated would be stored in the cloud for accessing by the authorized user. In addition to storing the reports in the cloud, it also alerts though the mail or by SMS to the registered numbers.

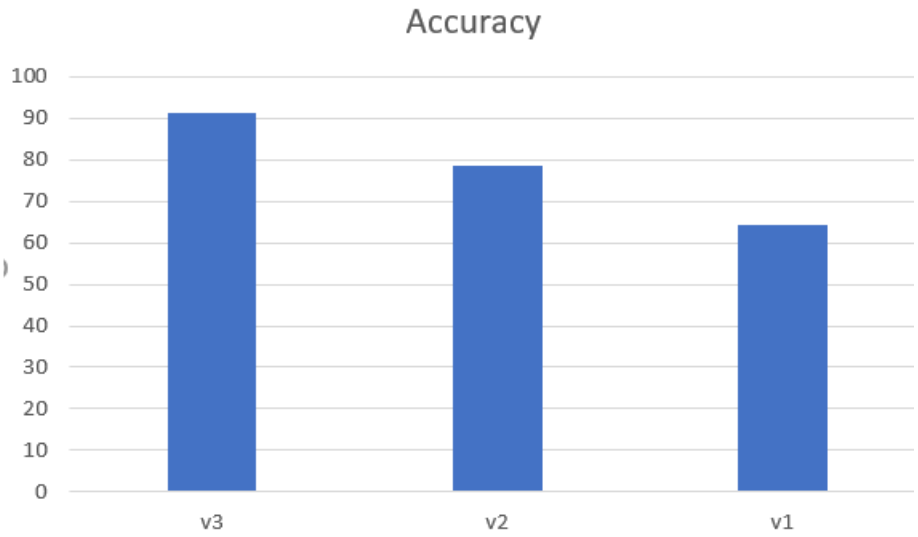


Figure: Flow Chart

CONCLUSION

The objective of this project was to develop a real-time object detection system using Convolutional Neural Networks (CNNs), particularly leveraging the YOLOv5 architecture. The system successfully captures video input from a webcam, processes each frame using a deep learning model, and outputs real-time predictions with bounding boxes and labels.

Through extensive testing, it was observed that the system performs efficiently in terms of both speed and accuracy, detecting multiple objects in real-time scenarios. The project demonstrates the power and potential of deep learning in solving computer vision problems, particularly in dynamic environments like surveillance, autonomous vehicles, and smart systems.

This implementation used a pre-trained YOLOv5 model which helped achieve state-of-the-art performance without the need for training from scratch, saving both time and computational resources. The project validates the capability of CNN-based models for object detection tasks and provides a strong foundation for future enhancements and real-world deployment

Future Work

While the current system is fully functional and demonstrates promising results, there is significant scope for improvement and extension:

1. Custom Dataset Training

- Future versions of the project can include training the model on a custom dataset relevant to specific domains (e.g., medical, industrial, agriculture) to improve detection performance and class specificity.

2. Deployment on Edge Devices

The system can be optimized and deployed on embedded devices like Raspberry Pi, Jetson Nano, or Android smartphones for real-world applications such as: Smart CCTV systems

Object counting

Wearable assistance devices

3. Integration with Cloud Services

- Enable cloud-based detection and analytics by integrating services like AWS Rekognition, Azure Cognitive Services, or custom Flask/Django APIs to store and analyze detection data.

4. Enhanced UI/UX

- A graphical user interface (GUI) can be designed using Tkinter, PyQt, or web technologies (React + Flask) for better usability and control.

5. Alert System

- Integrate a notification or alarm system that gets triggered when specific objects are detected (e.g., weapons, fire, or unauthorized personnel).

6. Model Optimization

Further improvements in inference speed and power efficiency can be achieved by: Using ONNX models
Applying model quantization and pruning
Exploring TensorRT optimization for NVIDIA GPUs

CHAPTER-6
REFERENCES

REFERENCES

1. **Redmon, J., & Farhadi, A.** (2018). YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*. https://arxiv.org/abs/1804.02767
2. **Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M.** (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. https://arxiv.org/abs/2004.10934
3. **Ultralytics YOLOv5 Documentation**. GitHub Repository. https://github.com/ultralytics/yolov5
4. OpenCV: Open Source Computer Vision Library. https://opencv.org
5. PyTorch: An Open Source Machine Learning Framework. https://pytorch.org
6. **COCO Dataset – Common Objects in Context**. https://cocodataset.org
7. **Pascal VOC Dataset**. http://host.robots.ox.ac.uk/pascal/VOC/
8. **Brownlee, J.** (2019). Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition. *Machine Learning Mastery*.
9. **Goodfellow, I., Bengio, Y., & Courville, A.** (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

YouTube Tutorials & Demos on YOLOv5 & OpenCV Integration Example: https://www.youtube.com/c/AladdinPersson