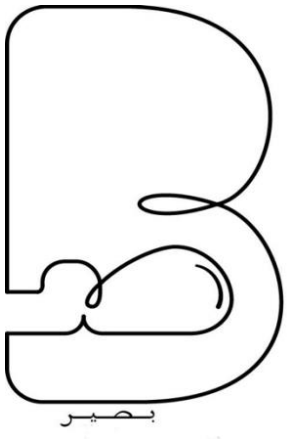
Imam Mohammad Ibn Saud Islamic University
College of Computer and Information Sciences
Computer Science Department
2 nd 1445-2024



Baasyir: A YOLO Based approach to Assist Visually Impaired People by leveraging Smart Glasses System

Team members:

Name	ID
Raseel Nasser Alkhamees	441021460
Fatima Fahad Altamimi	441020000
Rahaf Mohammed Alkhurif	441020819

Supervisor:
Dr. Dhouha Ben Noureddine

Submission date:
4/2/2024

Acknowledgments

First and foremost, by the grace and favor of Allah, we have successfully completed this Project.

Secondly, we extend our sincere gratitude to our esteemed supervisor, Dr. Dhouha, for her unwavering support, dedication, and kindness throughout the year. We will forever be grateful for her guidance and encouragement.

Thirdly, we express our gratitude to our department and its members for their collective efforts especially Dr.Sahar Abusaeed for her support and caring as well as to our university that brought us together.

Last but certainly not least, heartfelt thanks to our beautiful, caring, and supportive families. To our dear fathers and mothers, your unwavering support is the foundation of our journey. We wouldn't be here without you you are the silent strength and motivation behind our achievements, and for that, we are eternally thankful.

To our dear sisters and brothers, thank you for your patience and steadfast support throughout these two semesters. Sharing this accomplishment with you brings us immense joy.

Acknowledgments to Behind-the-Scenes Supporters: Eng. Salaeh Alkhamees, Abdullah Alkhurif.

Abstract

In this project, we aim to address the challenges faced by blind individuals due to resource limitations by introducing the 'Baasyir' smart glasses system, incorporating Deep Learning for obstacle detection. The system aims to assist blind people in recognizing objects, overcoming obstacles, and enhancing their independence. We proposed an approach that integrates two models the YOLO version 7 real-time object detection algorithm and the YOLOv7-tiny model, supported by an extensive literature review and we used relevant datasets. The developed approach's performance is evaluated using metrics such as Precision, Recall, F1-Score, IoU, and mAP. We conducted multiple experiments with YOLOv7 and YOLOv7-tiny implementation and parameter optimization on target datasets that showed promising results, achieving a higher mAP. The object detection models YOLOv7_Furniture, YOLOv7-tiny_Furniture, YOLOv7_Baasyir, and YOLOv7-tiny_Baasyir achieved detection accuracies of 75.3%, 71.1%, 49.4%, and 42.7%, respectively, on their respective datasets. Following the implementation of the object detection model, the Baasyir smart glass system was developed on the Raspberry Pi 3 Model B+. We concluded by suggesting potential future directions and areas for further research in this field.

Keywords: Baasyir Smart glasseses, Deep Learning, Object Detection, YOLOv7, YOLOv7-tiny, Baasyir dataset, Furniture-detection dataset, and Raspberry Pi.

Abstract in Arabic

في هذا المشروع، نهدف إلى معالجة التحديات التي يوجهها المكفوفون بسبب الموارد المحدودة من خلال تقديم نظام نظارات "بصير" الذكية، التي تدمج التعلم العميق وخوارزميات لكشف العوائق. يهدف النظام إلى مساعدة المكفوفين في التعرف على الأشياء واكتشاف العوائق وتعزيز استقلاليتهم. طبقنا نموذجين لخوارزمية كشف الأشياء في الوقت الفعلي نموذج YOLOv7 ونموذج YOLOv7-tiny، مدعومة بمراجعة شاملة لأبحاث سابقة واستخدام مجموعات بيانات ذات صلة. تم تقييم أداء المنهجية المطورة باستخدام مقاييس مثلا Precision، Recall، F1 Score، IoU وmAP. أجرينا تجارب متعددة على النموذجين وتجارب لتحسين المعاملات على مجموعات البيانات المستهدفة التي أظهرت نتائج واعدة، محققة mAP عالي. حققت نماذج كشف الأشياء YOLOv7_Furniture، YOLOv7-tiny_Furniture، YOLOv7_Baasyir، وYOLOv7- tiny_Baasyir نتائج كشف بنسب 75.3٪، 71.1٪، 49.4٪، و42.7٪ على التوالي، على مجموعات البيانات الخاصة بها. بعد تنفيذ نموذج كشف الأشياء، تم تطوير نظام نظارات بصير الذكية على Raspberry Pi 3 Model B+. اختتمنا بتقديم اقتراحات لأبحاث المستقبلية ومجالات البحث الإضافية في هذا المجال.

Contents

1 Introduction **11**

1.1 Introduction . . . 11
1.2 Aim and Objectives . . . 12
1.3 Methodology . . . 12
1.4 Timetable . . . 13
1.5 Team Qualifications . . . 14
1.6 Conclusion . . . 14

2 Literature Review **15**

2.1 Introduction . . . 15
2.2 Background . . . 15
2.2.1 Object Detection . . . 15
2.2.2 Machine Learning for Object Detection . . . 16
2.2.3 Deep Learning for Object Detection . . . 16
2.2.4 Object Detection in Smart Glasses . . . 18
2.3 Related Work . . . 18
2.3.1 R-CNN . . . 18
2.3.2 Fast R-CNN . . . 19
2.3.3 Faster R-CNN . . . 20
2.3.4 Single Shot MultiBox Detector . . . 22
2.3.5 YOLO . . . 22
2.4 Comparative table . . . 28
2.5 Conclusion . . . 30

3 Research Methodology **31**

3.1 Introduction . . . 31
3.2 Theoretical Formalization and Algorithm Design . . . 31
3.3 Experimental design . . . 37

3.3.1 Programming Language and Framework 37
3.3.2 Datasets 37
3.3.3 Hardware setup 40
3.4 Conclusion 45

4 Results and Discussion 47
4.1 Introduction 47
4.2 Data pre-processing experiments 47
4.3 Baasyir dataset experiments 48
4.3.1 YOLOv7 experiments 48
4.3.2 YOLOv7-tiny experiments 51
4.3.3 Results and discussion 53
4.4 Furniture-Detection dataset experiments 53
4.4.1 YOLOv7 experiments 54
4.4.2 YOLOv7-tiny experiments 56
4.4.3 Results and discussion 58
4.5 Real-time experiments 58
4.6 Comparison with previous researches 61
4.7 Conclusion 62

5 Conclusion and Future Work 63
5.1 Introduction 63
5.2 Contributions 63
5.3 Future Work 64
5.4 Conclusion 64

Reference 66

List of Figures

1.1 Project timeline 13

2.1 The relationship between AI, Machine Learning, and Deep Learning [1] . 16
2.2 Illustration of a Neural Network, the foundational architecture in Deep Learning [2]. 17
2.3 Representation of the R-CNN Structure, showcasing the multi-stage process for object detection in computer vision [3]. 19
2.4 Overview of the Fast R-CNN architecture, highlighting its efficient design for object detection in computer vision [4]. 20
2.5 The Faster R-CNN architecture, showcases its enhanced design for efficient and accurate object detection in computer vision tasks [5]. 21
2.6 The YOLO system architecture, a real-time object detection framework with a single pass through the network, provides swift and accurate results [6]. 22
2.7 The YOLO architecture, a state-of-the-art real-time object detection system, demonstrates efficient and simultaneous detection across the entire image [6]. 23
2.8 The YOLOv7-tiny architecture, a compact version of the YOLO model, is designed for real-time object detection with reduced computational complexity. 26
2.9 A comparison between SSD and YOLO versions. 27

3.1 Structure of object detection system 32
3.2 Structure of raspberry pi 3 Model B+ [7] 40
3.3 Connected hardware components enhancing the Baasyir smart glasses system. 42
3.4 The process of identifying the IP address of our Raspberry Pi, a crucial step for establishing remote connections and interactions. 43
3.5 Setup and enable the camera. 44
3.6 Raspberry pi camera ceiling. 44
3.7 Raspberry Pi 3 Model B+ terminal displaying motion status. 45

3.8 Raspberry Pi 3 Model B+ terminal displaying motion status after modifying the source. 45

4.1 Bassyir dataset before and after feature selection. 48
4.2 Confusion matrix for Baasyir dataset using YOLOv7 model 49
4.3 Precision, Recall and F1-score of Baasyir dataset on YOLOv7 model . . 50
4.4 mAP@0.5 curve for Baasyir dataset using YOLOv7 model. 50
4.5 Confusion matrix for Baasyir dataset using YOLOv7-tiny model. 51
4.6 Prcesion, Recall and F1-score for Baasyir dataset using YOLOv7-tiny model. 52
4.7 mAP@0.5 curve for Baasyir dataset using YOLOv7-tiny model. 52
4.8 Confusion matrix of the furniture dataset of YOLOv7 model. 54
4.9 Prcesion, Recall and F1-score for Furniture-Detection dataset using YOLOv7 model. 55
4.10 Precision-recall curve of the furniture dataset of YOLOv7 model. 55
4.11 Confusion matrix of the furniture dataset of YOLOv7-tiny model. 56
4.12 Prcesion, Recall and F1-score for Furniture-Detection dataset using YOLOv7-tiny model. 57
4.13 Precision-recall curve of the furniture dataset of YOLOv7-tiny model. . . 57
4.14 Real-time results for YOLOv7 model. 59
4.15 Real-time results for YOLOv7 BAASYIR model. 60

List of Tables

1.1 Team Qualifications 14

2.1 Summarization of YOLO models. 28
2.2 Comparative table between different models in the related work. 29

4.1 Results obtained from the YOLOv7 and YOLOv7-tiny models, conducted on Baasyir Dataset 53
4.2 Results obtained from the YOLOv7 and YOLOv7-tiny models, conducted on Furniture-Detection Dataset 58
4.3 Results obtained from experiments with the YOLOv7 and YOLOv7 Baasyir models, focusing on specific object classes. 60
4.4 Comparison of different object detection models on specific datasets, highlighting mean Average Precision (mAP) performance. 61

List of Abbreviations

ANNs	Artificial Intelligence
ANNs	Artificial Neural Networks
AP	Accuracy Percentage
BoF	Bag of Freebies
CNN	Convolutional Neural Networks
DL	Deep Learning
E-ELAN	Extended Efficient Layer Aggregation Network
FPN	Feature Pyramid Network
FPS	Frames Per Second
IoU	Intersection over Union
ML	Machine learning
MS COCO	Microsoft COCO
NMS	Non-Maximum Suppression
PANet	Path Aggregation Network
R-CNN	Regional-based Convolutional Neural Networks
RoI	Region of Interest
RPN	Region Proposal Network
SG	Smart Glasses
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machines
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Introduction

In recent years, there has been a notable increase in interest surrounding the advancement of innovative solutions to enhance the quality of life for individuals with visual impairments. Among these solutions, smart glasses (SG) have emerged as a promising platform to address the challenges faced by visually impaired individuals, providing real-time assistance in various activities. In the past decade, Artificial Intelligence (AI) has achieved significant success in a variety of fields, including the domain of computer vision especially in object detection. Utilizing Machine Learning (ML) and Deep Learning (DL) models has become a cornerstone in the development of SG applications, enabling them to extract valuable information and make informed decisions. smart glasses can leverage DL model for various object detection applications. Imagine a person wearing SG with an integrated camera and DL-based object detection software. As the user goes about their day, the SG continuously captures the scene in real-time through the camera. The DL model running on the SG processes the video feed and identifies objects in the wearer's field of vision. This can include:

- Face recognition: DL can be used for face recognition to identify people and provide information about them, such as their name, profession, or relationship to the wearer.
- Language translation: if the glasses recognize text, they can offer instant translation for foreign languages.
- Augmented reality information: SG can overlay relevant information based on the detected objects on the user's display.
- Navigation assistance: if the DL model identifies road signs or traffic signals, SG can provide turn-by-turn navigation instructions and real-time traffic updates.
- Object recognition: the glasses can help visually impaired users by identifying and describing objects in their surroundings, such as "a person," "an animal," or "a car."

- Safety alerts: DL can detect potential safety hazards, like warning signs, construction zones, or obstacles, and alert the wearer to avoid them.
- Inventory management: in a warehouse or retail setting, SG can use DL to track and identify products or items on shelves for inventory management.

In this project, we examine the performance of Convolutional Neural Networks (CNN) models in object detection to develop our Baasyir SG approach. We present our work as a comparative study where we experiment with different DL architectures. In this chapter, we give a preamble to the project aim, objectives, and methodology, along with a list of qualifications of each team member.

1.2 Aim and Objectives

The primary objective of the Baasyir glasses system project is to develop a wearable technology based on Smart glasses that enhances the independence and overall quality of life of individuals with visual impairments. According to the World Health Organization [8], there are over 2.2 billion people globally who suffer from vision impairments, many of which could have been prevented or treated.

The main aim of this research project is to build a smart glasses system based on the DL model to detect and classify objects in real-time accurately. To fulfill the aim of this project, the following objectives need to be met:

- Provide background and summary of research efforts in the field of smart glasses system that employs computer vision techniques and DL models for Blind people.
- Data collection and preparation: select an appropriate dataset that includes sensor data for wearable sensors.
- Data pre-processing: pre-process the raw measurements recorded in the selected datasets.
- Model architecture: implement deep learning classification models.
- Training, Evaluation, and Tuning: train and evaluate the performance of the implemented classification models.
- Deployment to smart glasses.
- Testing and Optimization.

1.3 Methodology

The Baasyir smart glasses project employs a methodology that centers on the implementation of the YOLO version 7 model for object detection. First, we initiated the process by conducting an extensive literature review to gather insights from previous research

and identify relevant techniques and approaches. Then, to optimize the performance of the object detection algorithm, we implemented various data preprocessing techniques, including the collection of suitable datasets, annotation of the data, augmentation to create diverse variations, and balancing to ensure equal representation of different object classes within the dataset. These preprocessing steps play a crucial role in enhancing the performance and accuracy of the object detection model. Then, we trained some DL models such as the YOLOv7, and YOLOv7-tiny models using two different datasets. Next, we implemented our contribution using YOLOv7 on the Raspberry Pi 3 Model B+ to develop the Baasyir system. We then evaluated the developed tool's performance using appropriate metrics and visualized the results to gain insights into its effectiveness. Furthermore, we interpreted the results and provided recommendations for future improvements to enhance the accuracy and efficiency of the object detection model in the SG. Finally, we deployed it to smart glasses.

1.4 Timetable

The project management plan schedule is shown in figure 1.1

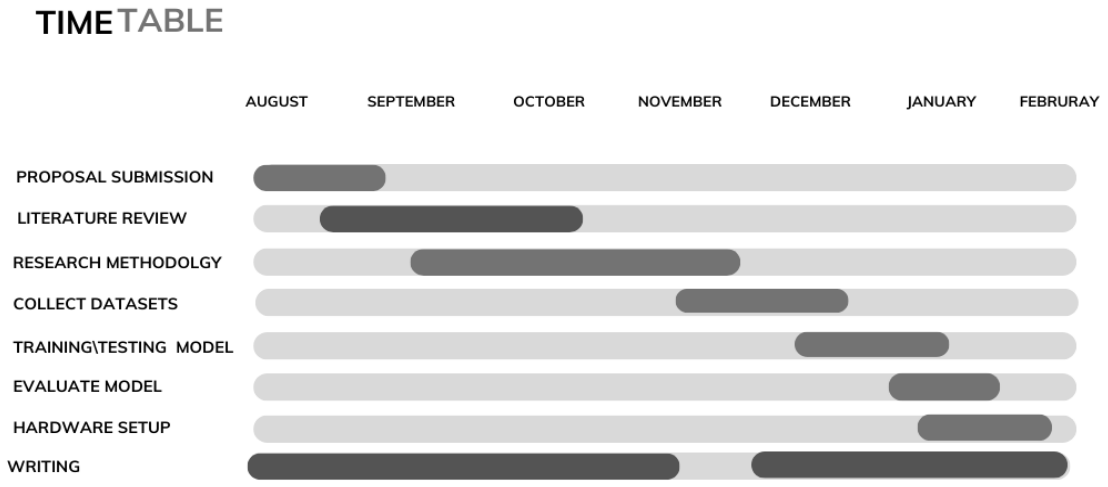


Figure 1.1: Project timeline

1.5 Team Qualifications

Table 1.1 shows the qualifications of our team.

Student Name	Qualifications
Rahaf Alkhurif	**Programming languages:** Python, Java, Swift **Database System:** MySQL **Operating System:** Windows, IoS, ubuntu
Raseel Alkhamees	**Programming languages:** Python, Java, C# **Database System:** MySQL **Operating System:** Windows, IoS, ubuntu
Fateema Altamimi	**Programming languages:** Python, Java **Database System:** MySQL **Operating System:** Windows, IoS, Ubuntu

Table 1.1: Team Qualifications

1.6 Conclusion

This chapter introduced the aim, objectives, and methodology of the project as a preamble to give a better understanding of the project scope of work. In the next chapter, the literature review, we will review related work in the area of computer vision including object detection in real-time.

Chapter 2

Literature Review

2.1 Introduction

After the discussion of the project aim, objectives, and the proposed methodology in the previous chapter, the next chapter delves deeper into the project details. This chapter comprises two primary sections: background and related work. In the background section, we provide a brief explanation of the knowledge and terminologies necessary to understand the scope of Smart Glasses using DL. In the related work section, we provide a summary of previous research related to this domain.

2.2 Background

2.2.1 Object Detection

Object detection is a more complex task than image classification, as it involves both object localization and object recognition within images.

- Image Classification: focus on categorizing images based on their features. In an image, there may be multiple classes or items to identify.
- Object Localization: locate the presence of objects in an image and indicate their location with a bounding box- ROI (Region of Interest). ROI is a new term we need to become familiar with for object recognition. We will learn how to obtain boxes that are as close as possible to the detected object.
- Object Detection: locate the presence of objects with a bounding box and detect the classes of the located objects in these boxes.

Object recognition neural network architectures created until now are divided into 2 main groups: Multi-Stage vs Single-Stage detectors.

2.2.2 Machine Learning for Object Detection

ML is a branch of AI that focuses on developing models that enable computers to learn and make predictions or decisions based on data. ML techniques play an essential role in the training of accurate object detection models within the SG context. This process contains the utilization of algorithms that analyze visual data to identify objects present in images or videos. It comprises several stages, including feature extraction, classification algorithms, and model optimization, all of which contribute to enhancing performance. SG uses ML techniques to train object detection models on labeled datasets. These models learn to distinguish patterns and features that differentiate between object classes, allowing them to detect similar objects in new images. Feature extraction is employed to extract pertinent attributes such as edges, textures, and shapes. Classification algorithms, such as Support Vector Machines (SVM) [9] or CNNs, are then used to differentiate between various object classes. Model optimization is subsequently employed to fine-tune parameters and improve accuracy.

The primary motivation behind utilizing ML for object detection is its ability to adapt to a wide range of visual data and generalize to recognize new images. This capability empowers SG to achieve real-time object detection and enhances the overall user experience to a considerable extent. Despite the significant achievements of traditional ML methods, the time-intensive process of manual feature extraction is a challenging task, heavily dependent on human expertise, and continues to pose challenges in object detection. DL is capable of learning patterns automatically from raw data without human intervention [10].

2.2.3 Deep Learning for Object Detection

Deep Learning, a subfield of ML as illustrated in figure 2.1, is concerned with training deep neural networks, which are artificial neural networks with multiple layers, to acquire intricate representations from data.

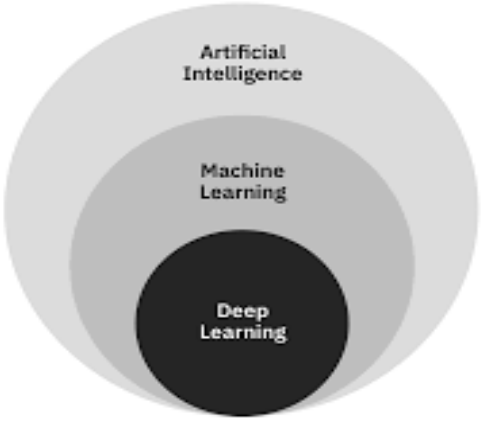


Figure 2.1: The relationship between AI, Machine Learning, and Deep Learning [1]

One of the primary advantages of DL lies in its capability to automatically learn and extract features from raw data, eliminating the need for manual feature engineering. This characteristic renders it particularly suitable for tasks such as image and speech recognition, natural language processing, and various other domains where the data may exhibit complex and unstructured characteristics.

DL models are composed of interconnected neurons organized into multiple layers as illustrated in figure 2.2. These layers typically include an input layer, multiple hidden layers, and an output layer. The training process in DL involves iteratively adjusting the weights of the connections between neurons using optimization algorithms like gradient descent. The objective is to minimize the disparity between the networks predictions and the actual labels provided in the training data.

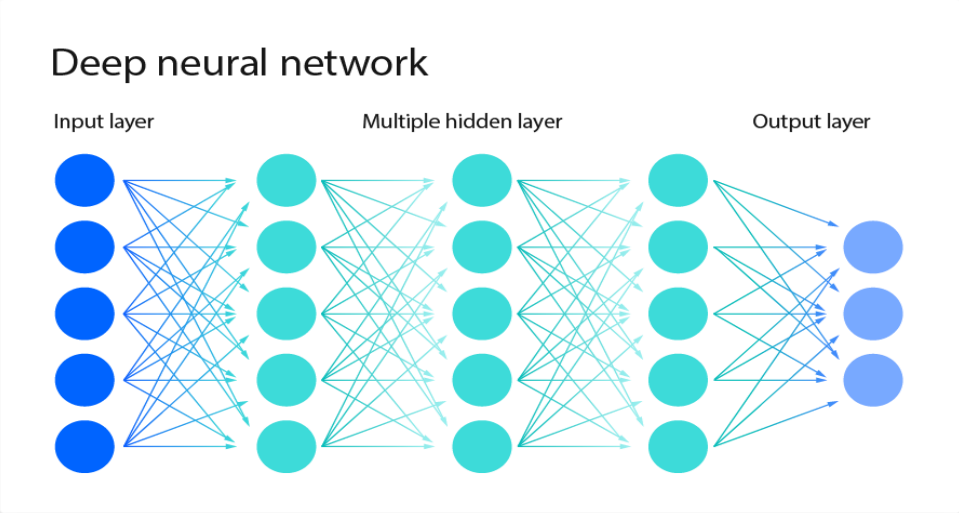


Figure 2.2: Illustration of a Neural Network, the foundational architecture in Deep Learning [2].

Deep neural networks, also known as deep learning models, have gained significant prominence due to their inherent ability to autonomously learn intricate features and patterns from extensive datasets. These networks encompass various types of neural networks, including Convolutional Neural Networks (CNNs) for image analysis, Recurrent Neural Networks (RNNs) for sequential data, Long Short-Term Memory (LSTM) networks for capturing long-term dependencies, and Generative Adversarial Networks (GANs) for generating realistic synthetic data.

A notable example of deep learning-based object detection is the Faster R-CNN [5] algorithm, which blends DL techniques with region proposal methods to achieve accurate and efficient object detection. Faster R-CNN has gained widespread use in the field of smart glasses by creating region proposals through a network and then classifying and refining them using a CNN in real-time.

2.2.4 Object Detection in Smart Glasses

The capability of smart glasses to detect objects within the wearer's visual field is a significant aspect of their functionality. This technology relies on computer vision algorithms and ML techniques to analyze the visual data obtained through the cameras embedded in the smart glasses for example in [11], [12]. Consequently, it enables the provision of real-time information pertaining to the objects present in the surrounding environment.

The general context of SG mechanisms is as follows: First, we collect the images and (or) videos from the surrounding environment through integrated cameras' sensors within the SG. Then, the captured visual data undergoes preprocessing procedures to optimize its quality by eliminating any potential noise or distortions. Next, to detect and locate objects within the scene, computer vision algorithms such as YOLO version 3 or Faster R-CNN are applied to the preprocessed data based on their features, a process known as object detection [13]. Finally, the identified objects are presented to the user through the smart glasses' display or audio feedback, thereby providing valuable information regarding the objects present in their immediate surroundings.

2.3 Related Work

In this section, we provide a brief summary of research in object detection on smart glasses using deep learning models. We have collected a total of 11 papers. At the end of this section, a table summarizes the contributions of each study, including the datasets used, proposed deep learning models, and performance metrics with results.

2.3.1 R-CNN

[3] proposed a Region-based Convolutional Neural Networks (R-CNN) approach, representing a significant advancement. R-CNN has combined CNN and regional proposals, and it is used for object detection and localization within an image. It consists of three modules as presented in figure 2.3: *(i)* a module generates category-independent region proposals, *(ii)* a model is CNN which extracts fixed-length features from each region, and *(iii)* a module is a set of class-specific linear support vector machines.

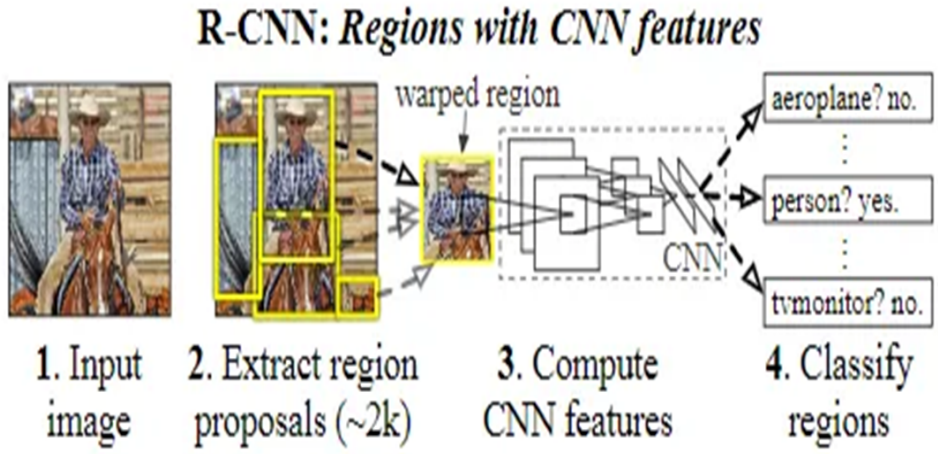


Figure 2.3: Representation of the R-CNN Structure, showcasing the multi-stage process for object detection in computer vision [3].

The model generates object regions using the selective search algorithm, which, as described in [14], combines the benefits of a comprehensive search and segmentation. It generates sub-segmentations and then groups the regions based on their pixel intensities. Iteratively, it combines similar regions to form objects.

The authors used in [3] a large dataset for classification, so the convolutional layer can learn image features, and by fine-tuning CNN it can learn new domains and identify detection from region proposal tasks. They used SVM for each class to detect which object belongs to a present class or not, by taking the 4096-d feature vector of each region proposal as an input. For the output a set of positive object proposals per class.

R-CNN bounding box is also an important aspect of understanding how R-CNN works. To compute features of the image in a region it must be converted into a compatible with the CNN. A bounding box is one of the simplest transformations by wrapping all pixels in a tight bounding box to the required size. The bounding box regression step is to predict the bounding box size and location to improve localization performance. So, the output from the SVM is all positive region proposals have an accurate corrected bound box around each object. R-CNN achieved a significant result, it gave a 30% relative improvement over the best previous results.

2.3.2 Fast R-CNN

According to Girshick in [4] the Fast R-CNN succeeds in solving some R-CNN problems such as building a faster object detection algorithm, reflecting the name. Fast R-CNN takes an image as input to the CNN instead of region proposals. Fast R-CNN as shown in figure 2.4 consists of CNN, the pooling layer replaced with an ROI pooling layer and fully connected layers that branch into a softmax layer and bounding box regressor (class-specific). The convolutional layer extracts features to produce a convolutional feature map.

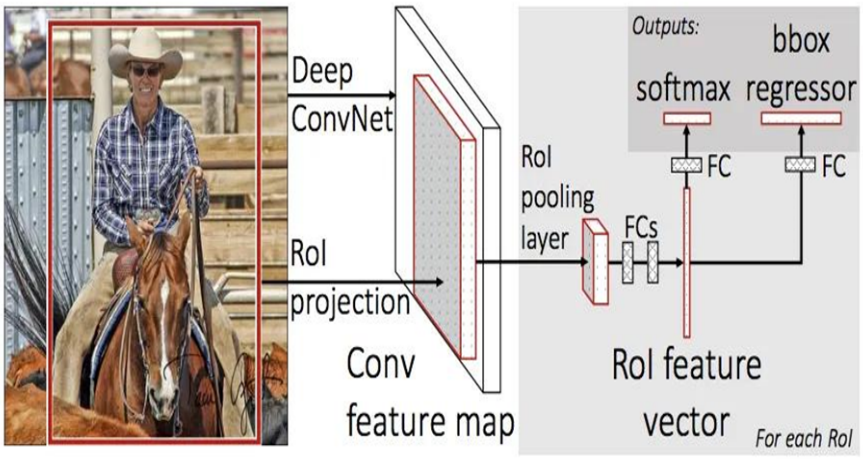


Figure 2.4: Overview of the Fast R-CNN architecture, highlighting its efficient design for object detection in computer vision [4].

The author stated in [4] that the ROI pooling layer is a special case of spatial pyramid pooling, but it has only one level. ROI takes each valid obtained region proposal from the feature map to extract a fixed feature vector by wrapping the region of proposals into rectangular windows to be compatible with the fully connected layer.

The fully connected layers and their branches take the output features from the RoI pooling layer as an input. The softmax branch predicts the probability value of each proposed region that belongs to which object class plus a catch-all background class. The bounding box regressor branch produces 4 bounding box regression offset values, the output makes the bounding boxes more precise. Fast R-CNN is a clean update of R-CNN and spatial pyramid pooling. For training and testing, fast R-CNN outperformed R-CNN on many datasets. Fast R-CNN reduced the detection time by more than 30 % because unlike R-CNN it doesnt have to feed 2000 regional proposals to CNN every time.

2.3.3 Faster R-CNN

Faster R-CNN is an extension of Fast R-CNN, it was proposed after the bottleneck problem during the detection the most time was taken by the selective search region proposal generation algorithm. As the name implies, Faster R-CNN is faster than Fast R-CNN because the authors in [5] introduced the Region Proposal Network (RPN) in place of the selective search algorithm. Faster R-CNN comprises two modules as presented in figure 2.5. These modules create a system with a unified network for object detection: deep fully convolutional network (RPN), and Fast R-CNN detector.

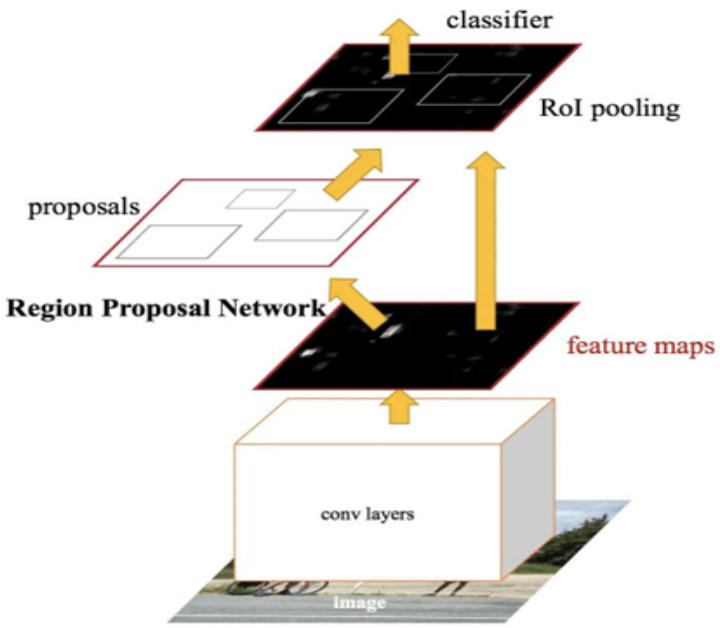


Figure 2.5: The Faster R-CNN architecture, showcases its enhanced design for efficient and accurate object detection in computer vision tasks [5].

The RPN consists of a classifier and a regressor. RPN processes an image of any size with a fully convolutional network and produces a set of region proposals. RPN slides a small rectangular window over the feature map produced from the last convolutional layer shared with Fast R-CNN. Each sliding window is mapped to a location in the feature map anchor boxes are used to generate a region proposal. Classifiers determine if an object in an anchor box is present or not, or if it contains a part of the background then generate the objectness score for each proposal. Regressor output is a coordination of the bounding box of an object. To remove redundancy (noise data) and select the most accurate proposals. Non-maximum suppression (NMS) was used based on the objectness score. NMS ensures selected proposals are both accurate and non-overlapping among a large of region proposals.

Fast R-CNN detector is responsible for object detection the first step is the region of interest (RoI) pooling is fed by the region proposal output from the RPN then transforms the region proposals into fixed-size feature map for each one by dividing each region proposals and applying max pooling within each cell. The same CNN used in RPN takes the feature map as input to extract features that object-specific information. Provide a better understanding of the proposed region content. The fully connected layers take the output of RoI pooling and extracted features as input to classify and bounding box regression. Faster R-CNN was experimenting on the datasets, decreasing the object detection time compared to fast R-CNN (0.2 seconds with Faster R-CNN compared to 2.3 with Fast R-CNN).

2.3.4 Single Shot MultiBox Detector

Wei Liu et al. proposed in [15] a Single Shot MultiBox Detector (SSD) approach, which is an object detection algorithm that uses a single DNN to detect objects at various scales. Applying multiple convolutional layers with different receptive fields to predict object categories and bounding box offsets. SSD uses a feature pyramid network (FPN) to extract features from the image at different scales. It then predicts bounding boxes and class labels of objects using a head network. SSD is fast and accurate, making it suitable for real-time object detection applications. But takes high memory usage and computational cost. The results on the dataset SDD showed significantly better performance for larger objects, with improvements of 4.8% in AP and 4.6% in AR. However, it showed smaller gains for smaller objects (1.3% in AP and 2.0% in AR).

2.3.5 YOLO

J. Redmon et al. developed in [6] a unique object detection system called You Only Look Once (YOLO), characterized by its high speed and more accuracy than SSD, the system models detection as a regression problem as presented in figure 2.6).

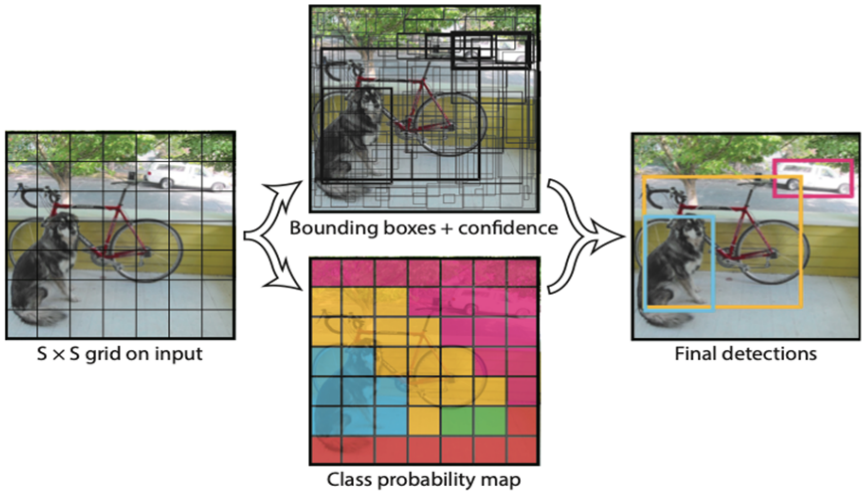


Figure 2.6: The YOLO system architecture, a real-time object detection framework with a single pass through the network, provides swift and accurate results [6].

Treated object detection as a regression problem to spatially differentiate objects and identify them in real-time. A single neural network predicts bounding boxes and class probabilities directly from full images. Processing up to 45 to 155 frames per second with different model versions. Figure 2.7 presents the detection network in [6]. It has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 Œ 1 convolutional layers reduces the feature space from preceding layers. They pretrained the convolutional layers on the ImageNet classification task at half the resolution (224 Œ 224 input image) and then doubled the resolution for detection.

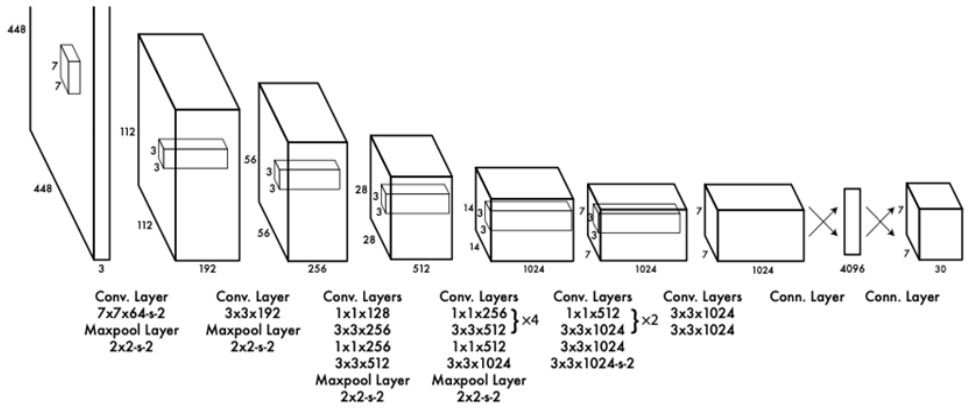


Figure 2.7: The YOLO architecture, a state-of-the-art real-time object detection system, demonstrates efficient and simultaneous detection across the entire image [6].

YOLO makes more localization errors but is less likely to predict false positives in the background. It streamlines the detection process by resizing the input image, running a single convolutional network on the image, and applying a model-confidence threshold for accurate results. The model consists of many layers: Input Image, CNN Backbone, Grid Division, Bounding Box Predictions, Class Predictions, and NMS. YOLO attained an average accuracy (mAP) of 63.4% which achieved a significant improvement over the previous method.

YOLOv2

Redmon and Farhadi optimized in [16] their last version of YOLO by proposing a new method to harness the large amount of classification data they already have and use it to expand the scope of current detection systems. Using a hierarchical view of object classification allows for a combination of distinct datasets together. Their method leverages labeled detection images to learn to localize objects while it uses classification images to increase its vocabulary and robustness. The YOLOv2 architecture consists of:

- The first input image used to identify objects,
- Darknet-19 Backbone: a unique 19-layer feature extraction architecture,
- Detection Head: numerous detection layers that forecast bounding boxes and class probabilities,
- Pre-defined anchor boxes: forecasting item sizes are known as anchor boxes,
- and NMS: A stage in the post-processing procedure that filters and improves detections.

YOLOv2 only predicts 98 boxes per image but with anchor boxes, predicts more than a thousand. Without anchor boxes, the intermediate model gets 69.5 mAP (Mean

Average Precision) with a recall of 81%. With anchor boxes, the model gets 69.2 mAP with a recall of 88%. Even though the mAP decreases, the increase in recall means that their model has more room to improve.

YOLOv3

After the revolution in object detection in YOLOv1. The authors were still open to changing their system to YOLOv2 and now YOLOv3 for more improvements. They introduced some key aspects of YOLOv3 in [13] such as detection at different scales, bounding Box predictions, class prediction with Logistic Regression, and performance optimizations Darknet-53. YOLOv3 changed the network classifier to logistic classifiers (like sigmoid) instead of softmax like previous versions of YOLO. In this version, the bounding box prediction has an objectness score for each bounding box using logistic regression, unlike YOLOv2. Predicts bounding boxes using dimension clusters as anchor boxes. Predicts boxes at 3 different scales. The system uses those scales to extract features using a similar concept to the feature pyramid network. For the feature extractor, a hybrid approach combines the YOLOv2 network and Darknet-53 is used instead of Darknet-19, which is 53 layers deep, and increases accuracy but might be slower. YOLOv3416 gives 55.3 mAP with inference time 29 ms, YOLOv3608 gives 57.9 mAP with inference time 51ms, whereas Faster RCNN gives 59.1 mAP with an inference time of 172 ms and YOLOv2608 gives 48 mAP with an inference time of 40 ms Note that -320, -416, -608 is the input image resolution.

YOLOv4

In 2020 scientists (such as [6], [16], [13], and [17]) were looking to improve YOLO even better. According to A. Bochkovskiy et al. in [17] YOLOv4 is a single-shot object detection algorithm that builds on the success of YOLOv3 [13]. It introduces several improvements, including a new backbone network, a new neck network, and a new loss function. YOLOv4 introduces the CSPDarknet53 backbone network and utilizes techniques like Path Aggregation Network (PANet) and Mish activation functions to enhance performance. It also introduces the YOLOv4 head network, which improves the accuracy and speed of the model. For real-time object recognition, YOLO v4 delivers cutting-edge results (43.5% AP). YOLOv4 improves on its predecessors by optimizing the network architecture and incorporating advanced features to enhance object detection accuracy and speed.

YOLOv5

Bochkovskiy et ali. used in [17] YOLOv5 in their research to propose a new clustering initialization method utilized in Couturier et al in [18]. The authors state that YOLO-v5 operates on an intelligent CNN to detect objects in real-time. It segments the image into various areas. It calculates the bounding boxes and probabilities for each region. The predicted probabilities are used to weight these bounding boxes. Requiring just a single forward pass through the neural network for predictions, the algorithm inspects

the image only once. It subsequently identifies and reports recognized objects along with their bounding boxes, following a process of non-max suppression to guarantee unique recognition of each object. YOLOv5 is a significant improvement over YOLOv4 in terms of speed and size. It builds on the success of YOLOv4 by making further improvements in network architecture, training techniques, and deployment efficiency.

According to Manikandan et al. in [19] YOLOv5 marks a substantial advancement in object detection capabilities. Utilizing the dataset, encompassing a preset of 80 objects, greatly expedites the detection process, enabling instantaneous identification of objects.

YOLOv6

According to Chuyi et al. in [20] YOLOv6 is a further improvement over YOLOv5 in terms of speed and accuracy [19]. It introduces several new features in [20], such as a new neck module called the Cascade Neck and a new Adaptive Anchor Matching algorithm, which improve the model's ability to detect objects of different sizes and shapes. YOLOv6 uses a new backbone network called CSPDarknet53, which is even more efficient and lightweight than the CSPDarknet53 network used in YOLOv5. This makes YOLOv6 easier to train and deploy on resource-constrained devices. According to Barazida in [21], YOLOv6 results on a dataset achieve 43.1 mAP.

YOLOv7

Chien-Yao et al. proposed in [22] YOLOv7, which builds on the success of previous versions by offering several improvements, including even better accuracy and speed. YOLOv7 introduces many new features, including a new neck module called the Extended Efficient Layer Aggregation Network (E-ELAN), Model Scaling for Concatenation-based Models, a new head module called the Trainable Bag of Freebies, and a new training strategy called the Planned re-parameterized convolution. YOLOv7 is easier to train and deploy on resource-constrained devices. YOLOv7 outperforms other object detection detection algorithms in terms of accuracy. [22] obtains an average precision of 37.2% at an IoU (Intersection over union) threshold of 0.5, which is comparable to other cutting-edge object detection methods.

YOLOv7-Tiny

YOLOv7 is an optimization based on Yolov5 with more advantages in detection accuracy and speed. It has a complex structure with numerous parameters, distinguishing it significantly from its counterpart, YOLOv7-tiny. It is part of the YOLO family, known for providing state-of-the-art results in object detection tasks, it is designed for optimized environments with limited computational resources like edge devices or applications requiring real-time processing with lower power consumption. As a smaller and faster version of the YOLOv7 model, designed to be more efficient while still maintaining a good balance between speed and accuracy.

YOLOv7-tiny is based on YOLOv7 with a simpler structure designed for edge GPUs

as shown in Figure 2.8. It consists of a backbone, neck, and head. In the core part of the model, known as the backbone, a more straightforward version called ELAN-T is utilized instead of a more complex version called E-ELAN (Extended Efficient Layer Aggregation Network) [23]. In this simpler version, certain convolution operations a type of mathematical operation used to process data are removed. Instead, only pooling, a method for reducing data dimensions, is used for downsampling, which is the process of reducing the resolution of the input data. This approach still maintains an optimized structure known as SPP (Spatial Pyramid Pooling), which helps in preparing more detailed feature maps that are then passed on to the next layer, known as the neck. In the neck of the model, the PANet structure is kept. PANet is a method used for combining various features at different scales, which is important for understanding the image at different levels of detail. Finally, in the head of the model, which is responsible for making the final decisions, a standard convolution is used to modify the number of channels in the data. This replaces a more complex method known as REPConv. Essentially, this change simplifies the process of adjusting how the model processes its information at the final stage.

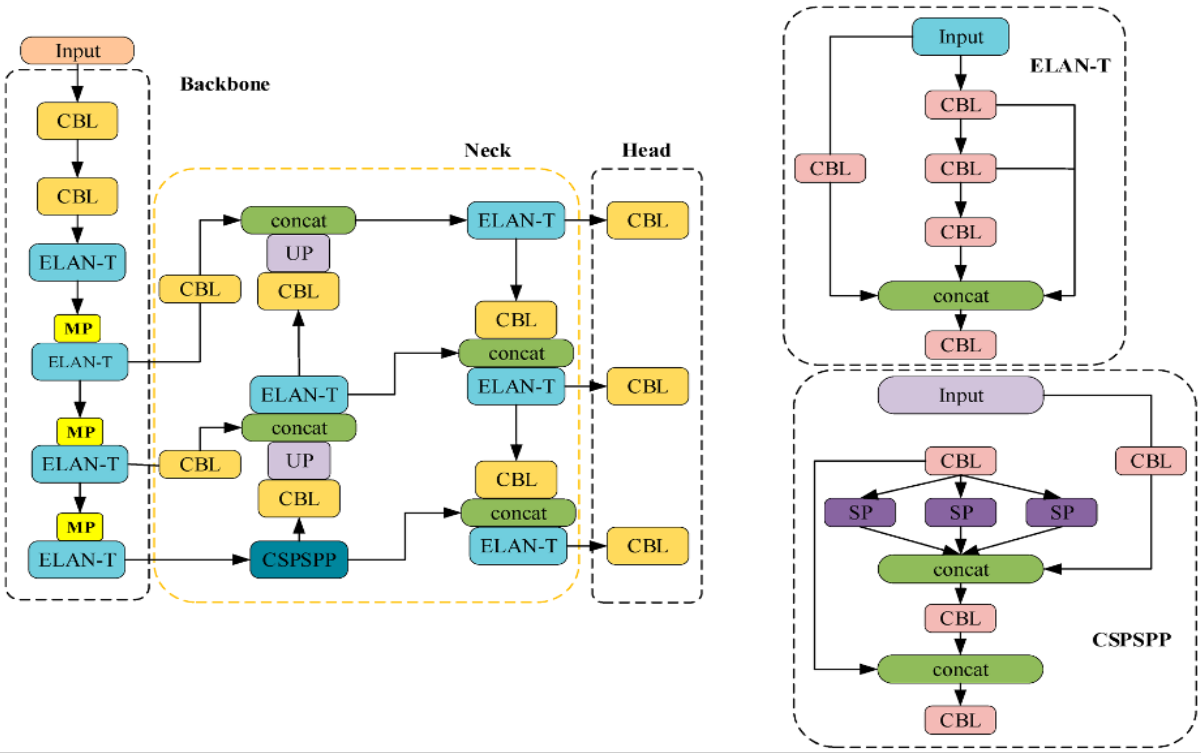


Figure 2.8: The YOLOv7-tiny architecture, a compact version of the YOLO model, is designed for real-time object detection with reduced computational complexity.

YOLOv7-tiny is optimized for fast inference times, making it suitable for real-time applications. While it may not match the accuracy of the full YOLOv7 model, it still delivers competent performance, especially in scenarios where processing power and memory are limited such as embedded systems.

A comparison between SSD and YOLO versions is illustrated in the figure 2.9. SSD model adds several feature layers to the end of a base network, which predicts the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300 Œ 300 input size significantly outperforms its 448 Œ 448 YOLO counterpart

in accuracy on the VOC2007 test while also improving the speed [6].

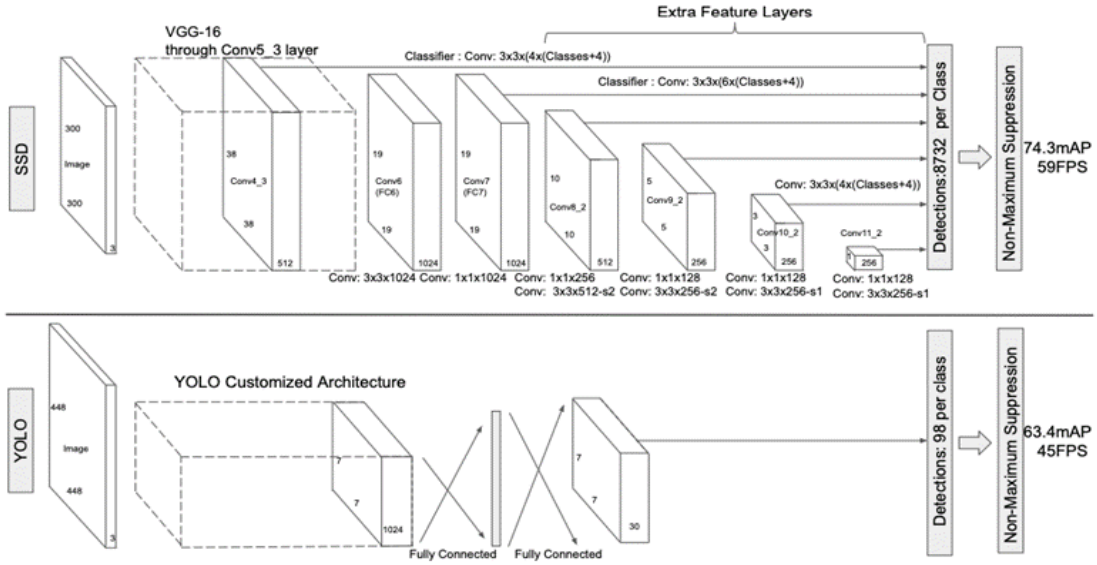


Figure 2.9: A comparison between SSD and YOLO versions.

Comparison between YOLO models

All the versions of YOLO use Microsoft COCO dataset that can be downloaded from the coco-dataset website [24]. The YOLOv7 model was trained using the MS COCO dataset, which is a comprehensive database for object discovery, classification, and annotation. This dataset comprises over 330,000 images from a wide range of categories, making it suitable for training and evaluating object detection and segmentation models. Each of these images is annotated with detailed object information, including 80 object categories, object locations (bounding boxes), pixel-wise segmentation masks for object instances, and 5 captions that describe the scene. The dataset's vast collection of objects and scenes provides a diverse range of training and evaluation opportunities. The reason for choosing the MS COCO dataset is that it has been extensively used in numerous research papers and studies.

Table 2.1 presents a comparison between YOLO versions. Note that *(i)* Tesla V100 is a powerful GPU that can be used for a variety of tasks, its unit is FPS. *(ii)* Accuracy ratio, all the versions used Microsoft COCO (MS COCO) dataset, its unit is AP (stands for Average Precision: which is a metric used to measure the performance of object detection algorithms.)

Table 2.1: Summarization of YOLO models.

Model	Year	Framework	Dataset	Tesla V100 GPU rate	Accuracy ratio	Backbone network
YOLO [6]	2016	Darknet	MS COCO	45.0% FPS	43.5% AP	AlexNet
YOLOv2 [16]	2016	Darknet	MS COCO	60.0% FPS	46.5% AP	Darknet-19
YOLOv3 [13]	2018	Darknet	MS COCO	66.7% FPS	50.0% AP	Darknet-53
YOLOv4 [17]	2020	PyTorch	MS COCO	72.5% FPS	52.0% AP	CSPDarknet53
YOLOv5 [18]	2021	PyTorch	MS COCO	60.0% FPS	46.5% AP	CSPDarknet53
YOLOv6 [20]	2022	PyTorch	MS COCO	66.7% FPS	50.0% AP	CSPDarknet53
YOLOv7 [22]	2022	PyTorch	MS COCO	72.5% FPS	52.0% AP	CSPDarknet53

Table 2.1 displays the summarization of the research-based papers on the YOLO models that have been reviewed, giving a total of 7 different papers. Beginning from 2016 to 2022, this shows that the growing interest in using object detection based on CNN models (YOLO) has only grown significantly. The same dataset (Coco MS) is used for all papers, indicating that there is a consistent choice of data source across the studies. Coco MS dataset is an available resource and serves as a common benchmark for evaluating object detection models.

These references have been applied in different settings, such as image recognition in autonomous vehicles to enhance road safety, surveillance systems for security and monitoring, retail automation to track inventory and customer behavior, and medical imaging for detecting and diagnosing various medical conditions. This broad applicability underscores the versatility and effectiveness of YOLO models in diverse domains.

2.4 Comparative table

We will summarize this section with a comparative table between YOLO models and other models cited in this review. As you can see in table 2.2 we compare R-CNN, Fast R-CNN, Faster R-CNN, SSD, and YOLOv7 models.

In Table 2.2 of our research paper, we reviewed 4 other models for object detection based on CNN architecture published between 2014 and 2022. So in total, we have reviewed 11 papers. Specifically, there was one paper published in 2014, two in 2015, two in 2016, one in 2018, two in 2020, one in 2021, and two in 2022. These papers covered a range of topics related to object detection, from advancements in model architectures to improved training techniques and applications across various domains.

Table 2.2: Comparative table between different models in the related work.

Model	Framework	Dataset	Tesla V100 GPU rate	Accuracy ratio	Backbone network	Neck module	Head module	Key characteristic
R-CNN [3]	Caffe	PASCAL VOC 2007	110 ms	58.5% AP	VGG16	None	None	Selective search region proposal
Fast R-CNN [4]	Caffe	PASCAL VOC 2007	18 ms	66.0% AP	VGG16	None	None	Region proposal network (RPN)
Faster R-CNN [5]	Caffe	PASCAL VOC 2007	7 ms	73.2% AP	VGG16	Feature Pyramid Network (FPN)	Fully connected layers	Improved RPN and FPN
SSD [15]	Caffe	PASCAL VOC 2007	31 ms	74.3% AP	VGG16	None	Multi-box layer	Single-shot detection
YOLOv7 [22]	PyTorch	COCO	72.5 ms	52.0% AP	CSPDarknet53	Extended Efficient Layer Aggregation Network (E-ELAN) Model Scaling for Concatenation based Models	Trainable Bag of Freebies	Single-shot detection with CSPNet, E-ELAN, and Trainable Bag of Freebies

The same dataset (Pascal VOC 2007) is used for R-CNN, Fast R-CNN, Faster R-CNN, and SSD, indicating the dataset's enduring relevance as a benchmark for evaluating object detection models over several years. Pascal VOC 2007 has played a significant role in assessing the performance and progress of these models and remains a valuable resource in the field of computer vision and object detection research. Backbone network architectures vary among the models. For instance, R-CNN, Fast R-CNN, Faster R-CNN, and SSD employ VGG16 as their backbone network. These backbone networks are crucial in feature extraction from input images. The neck module is responsible for further processing the features extracted by the backbone network. Different models employ various techniques in the neck module, such as Faster R-CNN uses Feature Pyramid Networks (FPN), while YOLOv7 employs an Extended Efficient Layer Aggregation Network (E-ELAN). These modules enhance the representation of object features. The head module, on the other hand, is responsible for predicting bounding boxes and class labels for detected objects. YOLOv4, for instance, employs a YOLO head module that predicts object coordinates and classes in a single pass. Each model's choice of head module influences its overall accuracy and efficiency in object detection.

2.5 Conclusion

In conclusion, in this literature review section, we have covered the terminologies and knowledge needed to understand the project scope in the background section. We have also reviewed the state-of-the-art in the area of computer vision including object detection in smart glasses in real-time.

Chapter 3

Research Methodology

3.1 Introduction

In the previous chapter, we outlined the project's scope and provided an overview of the prior research on the use of machine learning and deep learning models in object detection for smart glasses. However, since the chosen project topic is Baasyir smart glasses, this chapter will be dedicated to this research domain.

This chapter is structured into two distinct sections: the theoretical formalization and algorithm design section, and the experimental design section. In the first section, we explain the project's workflow plan in detail, and the algorithm design used, while the second section covers the essential resources required for the project's implementation, such as programming languages, frameworks, and datasets.

3.2 Theoretical Formalization and Algorithm Design

YOLOv7, an improved variant of the YOLO architecture, offers enhanced speed and accuracy, making it a popular choice for object detection applications where real-time performance is crucial. We initially implemented YOLOv7 and YOLOv7-tiny, and based on the experimental results and the accuracy rates published, we proposed our contribution, which involves improving YOLOv7 within our specific context. As YOLO v7 pre-trained on MS COCO achieves a mAP of 52.0% at a frame rate of 72.5 FPS (Frames Per Second) [22].

The objective of this project is to implement deep learning algorithms for object detection within the field of smart glasseses. The workflow of our project is depicted in Figure 3.1.

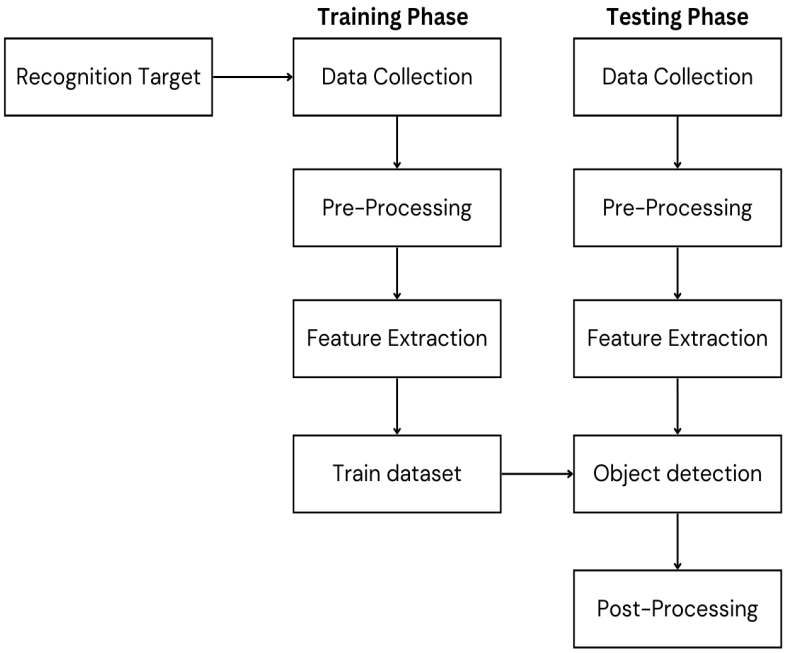


Figure 3.1: Structure of object detection system

Step 1: Data collection

Typically, the object detection system initiates by capturing and gathering images from individuals through embedded sensors in smartphones and wearable devices (e.g., smart glasses). The data collection step is imperative for this project; additionally, we utilize a publicly available dataset, with further details provided in Section 3.3.2.

Data annotation

The YOLOv7 format for object detection annotations typically involves labeling each object with its corresponding class index and providing the bounding box coordinates. These coordinates consist of the x and y coordinates of the box's center, its width, and its height. Notably, the bounding box coordinates are normalized, meaning they are proportional to the dimensions of the input image. For instance, a label line might look like "class x_center y_center width height," where the class represents the object category, and the bounding box coordinates are specified in normalized values. This format allows YOLOv7 to efficiently detect and classify multiple objects within an image in real-time, making it a popular choice for various computer vision applications, including autonomous vehicles, surveillance, and robotics. So, to combine datasets we needed to edit the corresponding class on the label file for each image to match the array on the configuration file.

Step 2: Data pre-processing

To improve the object detection system's performance, data preprocessing techniques are employed, including data augmentation, and data splitting.

Feature selection

When building a machine learning or deep learning model, the feature selection holds significant importance. It also referred to as dimensionality reduction, is a method employed to carefully choose a subset of pertinent or meaningful features from the original set while eliminating irrelevant, noisy, or redundant ones. This process can be automated using algorithms or performed manually by explicitly excluding specific columns. In our scenario, we manually conduct feature selection by explicitly dropping irrelevant columns from the dataset. Further information can be found in Section 4.1.

Data augmentation

Data augmentation is a fundamental technique in object detection that involves generating new training data from existing samples by applying various transformations. These transformations can include rotations, flips, scaling, and more, which will be explored further in the upcoming section 3.3.2. Data augmentation helps diversify the dataset, making the model more robust and capable of generalizing to different scenarios. It mitigates overfitting and enhances the models ability to detect objects under various conditions, ultimately improving its overall performance and accuracy in real-world applications.

Data splitting

Data splitting in object detection is the process of dividing a dataset into three main subsets: a training set, a validation set, and a test set. The training set is used to train the object detection model, the validation set helps fine-tune and optimize the models hyperparameters, and the test set is used to evaluate the models performance on unseen data. Proper data splitting is essential for assessing the models generalization ability and ensuring it performs well in real-world scenarios. This technique helps prevent overfitting and provides a reliable measure of the models accuracy and robustness, which will be further examined in the upcoming section 3.3.2.

Step 3: Train and Test

After performing the data pre-processing step, the data is ready to be fit into DL models for training. Implementing DL models requires an important step known as hyperparameter tuning. Training and testing a model using YOLOv7 in object detection follows the typical DL workflow with YOLOv7 architecture. During training, the model learns to detect objects by iteratively adjusting its parameters through a process called backpropagation. It utilizes a labeled dataset to minimize prediction errors and improve accuracy.

After training, the model's performance is evaluated using a separate testing dataset, ensuring its ability to generalize and accurately detect objects in real-world scenarios.

In our training process, we focused on three key parameters crucial for effective and efficient model training. Firstly, for the 'batch' parameter, we conducted extensive experimentation considering the trade-offs between learning rate, GPU memory usage, and training time. After careful consideration, we chose a batch size of 8. This decision took into account factors such as image size (set to 640 pixels for practical usability), learning rate and results (smaller batches potentially yielding better results), GPU memory usage (smaller batches requiring less memory per epoch), training time (smaller batches increasing the time per epoch), and overall resource optimization. Secondly, for the 'epochs' parameter, defining the number of iterations through the training dataset, we set it to 100 to ensure sufficient exposure for effective learning.

As mentioned before, we considered the epoch hyperparameter, which refers to the number of iterations the model trains its samples. For all models, we conducted experiments with 100 epochs. When tuning hyperparameters for YOLOv7, it's essential to focus on key factors that significantly impact the performance of the object detection model. We considered some key hyperparameters and factors:

- Learning Rate: it determines the step size during optimization. A too-high learning rate may lead to overshooting, while a too-low learning rate may result in slow convergence. For our experiments, we set the learning rate to 0.01, and we also considered using adaptive learning rate methods like Adam.
- Batch size: it determines the number of samples processed in each iteration. A smaller batch size may provide regularization effects, but it can also slow down training. On the other hand, a larger batch size can speed up training but may lead to overfitting. It's important to find a balance based on your hardware capabilities and the dataset size. For our experiments, we used a batch size of 8.
- IoU (Intersection over Union) Threshold: it is used to determine whether a predicted bounding box is considered a true positive. Adjusting this threshold can impact precision and recall. Experiment with different IoU thresholds to find a balance between precision and recall. In our experiments, we used an IoU threshold = 0.5.
- Confidence Threshold: it is used to filter out low-confidence predictions. Adjusting this threshold can impact the number of false positives and false negatives. Set a threshold that balances precision and recall based on your application requirements In our experiments, we utilized a Confidence Threshold of 0.25.
- Anchor Boxes: they help the model predict bounding box dimensions. The number and sizes of anchor boxes depend on the characteristics of the objects in your dataset. Experiment with different anchor box configurations to better match the object sizes in your dataset. In our experiments, we explored anchor boxes with sizes varying in the range [0.02 - 0.05].
- Number of Layers and Filters, in YOLOv7's architecture involves stacking layers with a certain number of filters. The depth and width of the network affect its

capacity to learn complex features. Adjust the number of layers and filters based on the complexity of our dataset. Be cautious not to make the network too deep, as it might lead to overfitting.

- Data augmentation parameter, in YOLOv7 supports data augmentation to increase the diversity of the training dataset. Experiment with parameters such as rotation, scaling, and flipping to improve model generalization. Configure augmentation parameters in the YOLOv7 configuration file based on the characteristics of our dataset. In our experiments, we explored different Data Augmentation parameters, specifically with rotate, rotation, and cutout techniques, which will be explored further in the upcoming section 3.3.2.

To mitigate overfitting in the YOLOv7 model, we employed various techniques, including data augmentation. YOLOv7 supports data augmentation through its configuration file, and we ensured that we appropriately configured augmentation parameters. Batch normalization layers were used to normalize the input to a layer, stabilizing and expediting the training process. YOLOv7 typically includes batch normalization layers. We selected Adam as an optimization algorithm with adaptive learning rates. This choice helps the model converge faster and prevents overfitting. We fine-tuned the learning rate based on the training progress. Additionally, we employed cross-validation to assess our model's performance on multiple folds of the data, ensuring generalization across different subsets of the dataset. Experimenting with these techniques, monitoring the training and validation performance, and fine-tuning accordingly are crucial steps to strike a balance between fitting the training data and generalizing to new data

Step 4: Performance evaluation

Performance evaluation for object detection using YOLOv7 involves assessing the model's ability to accurately identify and locate objects in images. Common evaluation metrics include precision, recall, F1 score, and mean average precision (mAP).

Confusion Matrix

It is a table that shows the correct and incorrect predictions of our model compared to the true labels. It helps us see when the model gets confused and mixes up different classes. We used the confusion matrix to identify which specific classes of furniture were most accurately detected by the model, as well as to pinpoint which classes were often mistaken for others.

Intersection over Union (IoU)

IoU is a metric used to quantify the percent overlap between the target bounding box (the ground truth) and the model's predicted bounding box. It is a ratio that ranges from 0 (no overlap) to 1 (perfect overlap).

Precision

It measures the proportion of correct detections. It tells us how many of the items the model labeled as furniture were truly furniture. Precision is critical when the cost of a false positive is high. We aimed for high precision to ensure that when our model detects an object as furniture, it's very likely to be correct. Precision (known as Positive Predictive Value), measures how many of the positive predictions made by a model are correct. Precision is defined formally as follows:

$$TruePositives/(TruePositives + FalsePositives) \tag{3.1}$$

Recall

It tells us what proportion of actual positives was correctly identified by the model. It's a measure of our model's ability to find all the relevant cases within a dataset. High recall is important in scenarios where every object is important to detect. We used recall to ensure that our model doesn't miss any instances of the furniture items it's supposed to find. Recall (known as Sensitivity, True Positive Rate), measures how many of the actual positive instances in the dataset were correctly predicted by the model. Recall is defined formally as follows:

$$Recall = TruePositives/(TruePositives + FalseNegatives) \tag{3.2}$$

F1-score

It combines both metrics into a single value. It provides a balance between these two metrics. F1 score is defined formally as follows:

$$F1Score = 2 * (Precision * Recall)/(Precision + Recall) \tag{3.3}$$

Mean Average Precision at IoU=0.5

mAP is a comprehensive metric for evaluating object detection models, especially in scenarios with multiple object classes and various levels of difficulty. It involves several steps, including computing precision-recall curves for each class, calculating the area under the curve for each class, and then taking the mean of these areas across all classes. mAP@0.5 is a specific instance of the mAP metric where the IoU threshold is set to 0.5. This means that detection is considered correct only if the IoU between the predicted bounding box and the ground truth is 0.5 or higher. While IoU is a continuous measure used for individual predictions, mAP@0.5 is an aggregate performance metric across the entire dataset that uses IoU=0.5 as a binary classifier to determine whether each individual detection is correct or not.

By analyzing these metrics, we gained a comprehensive understanding of our model's strengths and where it needs improvement. The confusion matrix and IoU gave us insights into the model's accuracy and precision in object localization, while recall and precision provided a measure of its classification performance. Together, these metrics paint a

detailed picture of how well our model is performing and where we might focus on future improvements.

3.3 Experimental design

3.3.1 Programming Language and Framework

We used Python [25] as a programming language in our project, which is an interpreted, object-oriented, and high-level programming language originally developed by Guido van Rossum in the early 1990s [26]. Python is widely adopted in machine learning and deep learning due to its availability of supportive libraries and packages. These tools simplify the implementation, training, and testing of ML and DL models, allowing for concise code that makes Python an accessible and favored choice in this domain.

For object detection, we chose PyTorch [27], a popular framework known for its rich ecosystem of pre-trained models and tools, making it an excellent choice for developing and training object detection models. PyTorch's flexibility and research-friendly environment make it a preferred platform for both researchers and developers in the field of object detection. Its seamless integration with other libraries and strong community support further solidify its position as a top choice for object detection tasks.

To leverage computing resources efficiently, we utilized Colab (Google Colaboratory), a cloud-based platform that provides free access to computing resources, including GPUs, making it popular for machine learning and data analysis tasks. Developed by Google, Colab allows users to write and execute Python code in a collaborative environment. It integrates seamlessly with Google Drive, enabling easy sharing and version control of notebooks. With pre-installed libraries and the ability to install custom ones, Colab streamlines the development process, making it a convenient and efficient tool for collaborative coding projects, research, and educational purposes.

3.3.2 Datasets

In this project, we employed two datasets: Baasyir and furniture. The Baasyir dataset is a compilation from 12 datasets, which will be elaborated in section 3.3.2. The second dataset is available for download from the website [28].

3.3.2.1 Baasyir dataset

We collected the Baasyir dataset by focusing on items that would be helpful for impaired individuals, such as bathroom items, and items used in daily life as toothbrushes hair driers, and furniture which the common obstacles in indoor environments.

Data collection

Our Baasyir dataset is an extensive collection created by merging 12 different datasets, resulting in approximately 21,337 images across 42 classes. The dataset is partitioned into train, test, and validation sets, with the following percentage allocations: 70% for training (about 18,657 images), 15% for testing (1,340 images), and 15% for validation (also 1,340 images). Datasets included in the compilation are:

1. Furniture detection Computer Vision Project Dataset [28]
2. Basket_detection Computer Vision Project Dataset [29]
3. Bathroom items Computer Vision Project Dataset [30]
4. Bathroom Computer Vision Project Dataset [31]
5. Broom detection Computer Vision Project Dataset [32]
6. Deteksi Sikat Gigi Computer Vision Project Dataset [33]
7. Hair Dryer Computer Vision Project Dataset [34]
8. Hdsljshvöyhcf Computer Vision Project Dataset [35]
9. Shuilongtou Computer Vision Project Dataset [36]
10. Stairs Image Dataset | Parts of House | Indoor Computer Vision Project Dataset [37]
11. Toilet Computer Vision Project Dataset [38]
12. Talov-hairdryer Computer Vision Project Dataset[39]

Data pre-processing

After downloading the dataset images to the Roboflow website [40], we organized the classes to ensure no repetition or duplication of classes. Subsequently, a series of preprocessing techniques were applied to elevate the overall quality, consistency, and relevance of the dataset, with the primary goal of enhancing the model's adaptability to diverse conditions and mitigating the risk of overfitting. These techniques included auto-orienting all images for a consistent orientation based on their metadata, resizing to standardize dimensions at 640x640 pixels, modifying classes by adjusting or altering object labels/categories as needed and filtering out instances with missing or null values to ensure cleaner dataset.

Data augmentation

To enhance the training datasets, we employed several data augmentation techniques. Firstly, we applied the "Rotate" technique, rotating images by 90 degrees to introduce variation and contribute to better model generalization. Secondly, the "Rotation" technique was utilized, randomly rotating images within the range of -15 to +15 degrees,

aiming to introduce variability to the training set and enhance the model's robustness to different orientations. Additionally, we incorporated the "Cutout" technique, implementing four boxes, each covering 10% of the image's size, to remove rectangular regions. This approach helps prevent overfitting by compelling the model to focus on other parts of the image, ultimately promoting better generalization during training.

Data splitting

Baasyir dataset is partitioned into three sets: the training set, comprising 70% of the data, which equates to approximately 18,657 images; the validation set, consisting of 15%, equivalent to 1,340 images; and the test set, also encompassing 15%, totaling 1,340 images. This division ensures a balanced distribution of data for training, validation, and testing, facilitating the evaluation and optimization of the model's performance across different subsets of the dataset.

3.3.2.2 Furniture-Detection dataset

We selected the Furniture-Detection dataset based on its relevance to indoor environments, which aligns with our primary focus. This dataset is especially well-suited for benchmarking against our Baasyir dataset, facilitating meaningful comparisons and evaluations.

Data Composition

The dataset encompasses 11 classes, representing common indoor furniture items. It includes a total of 9911 images, offering a diverse range of examples for each furniture category.

Data pre-processing

The "Auto-Orient" technique was applied as part of the preprocessing phase to ensure the uniform orientation of all images based on their metadata. This technique plays a crucial role in standardizing the orientation of images within the Furniture-Detection dataset, contributing to a consistent and well-organized dataset.

Data splitting

The dataset is divided into three sets for training, validation, and testing purposes. The training set constitutes 75% of the dataset, encompassing 7404 images. The validation set makes up 25% of the dataset, consisting of 2506 images. However, the test set is relatively small, comprising only one image, representing 0% of the dataset. This limited size of the test set may be due to specific constraints or considerations, and it's essential to acknowledge its small representation when interpreting the evaluation results of the model. The distinct proportions allocated to each set aim to strike a balance

between providing sufficient data for training and validation while maintaining a realistic representation for testing the model's generalization on new, unseen data.

3.3.3 Hardware setup

Raspberry pi 3 model B+

The Baasyir smart glasses system relies on the Raspberry Pi 3 Model B+ as its central computing unit. As you can see in figure3.2 this compact computer, comparable in size to a credit card, is equipped with a powerful 64-bit quad-core processor running at 1.4GHz. It offers dual-band wireless LAN (2.4GHz and 5GHz) for seamless connectivity, as well as Bluetooth 4.2 for wireless communication. Additionally, it features Gigabit Ethernet over USB 2.0 for high-speed wired connections. What sets it apart is its ability to be powered over Ethernet (PoE) with a separate PoE HAT, providing a flexible power solution. Despite its small size, the Raspberry Pi 3 Model B+ maintains compatibility with its predecessor models and delivers impressive performance. As a result, it serves as the perfect core for smart glasses, delivering the necessary computational power and connectivity to ensure the smooth functioning of the Baasyir system.

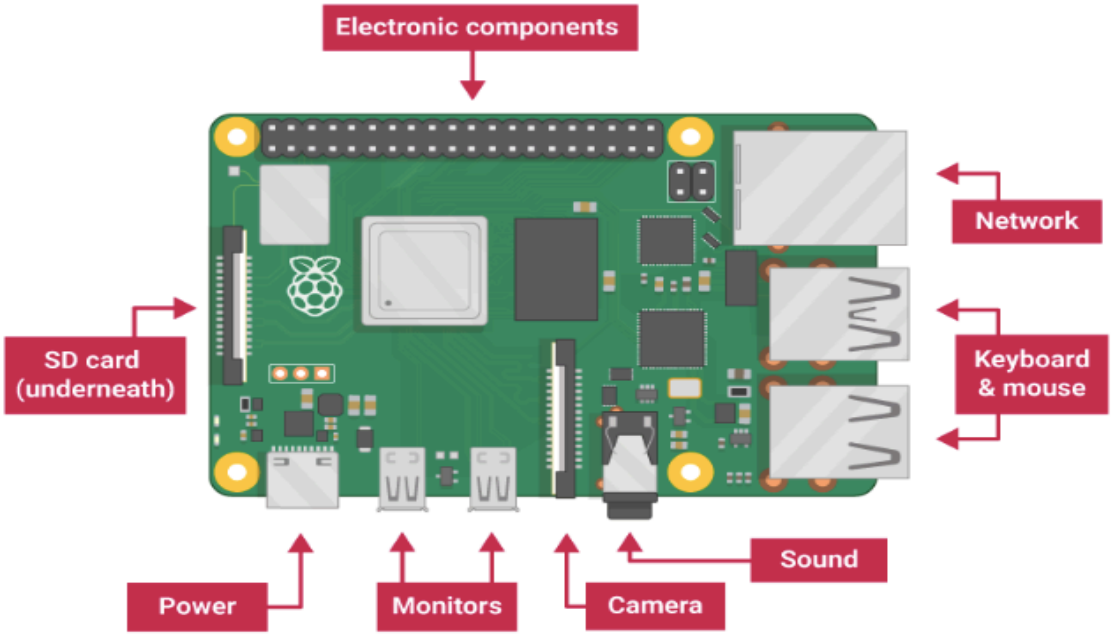


Figure 3.2: Structure of raspberry pi 3 Model B+ [7]

Raspberry pi camera

The inclusion of the Raspberry Pi Camera greatly enhances the imaging capabilities of the Baasyir smart glasses system. This camera module, boasting a 5-megapixel resolution, excels in capturing high-quality 1080p videos and still images. It directly connects to

the Raspberry Pi via a ribbon cable linked to the Camera Serial Interface (CSI) port, establishing a crucial link for visual data. Once powered up with the latest version of Raspbian (the official operating system for the Raspberry Pi), the camera is ready for use. Despite its compact size (approximately 25mm x 20mm x 9mm and weighing just over 3g), the camera doesn't compromise performance. Its fixed focus lens ensures clarity in capturing still images at 2592 Œ 1944 pixels. The camera supports various video modes, including 1080p30, 720p60, and 640Œ480p60/90, making it versatile for diverse applications. This compact yet powerful camera module is particularly well-suited for mobile and weight-sensitive applications, significantly enhancing the overall functionality of the Baasyir smart glasses system.

Display and other I/O devices

The Samsung monitor is an essential component of our hardware setup for the Baasyir smart glasses system. It serves as a programming interface for the Raspberry Pi, allowing developers to interact with and configure the system during the programming and development phases. To facilitate programming tasks, we have also included a wireless keyboard and mouse, which can be conveniently connected to the Raspberry Pi through USB slots. This setup enables developers and programmers to efficiently work on the system, making necessary adjustments, coding, and configuring parameters. By combining the Samsung monitor with the wireless input devices, we have created a streamlined development environment for the Baasyir smart glasses system.

Power supply

The Goui G-BELT10-K power bank is an essential component of our Baasyir smart glasses hardware setup, ensuring uninterrupted functionality. Boasting a robust 10000mAh capacity, this power bank serves as a reliable energy source for the smart glasses, guaranteeing continuous operation. It offers versatile input options with Micro USB and Type-C, while its multi-faceted output includes two 5V-3A ports and an additional Type-C port. With a total of three charging ports and a 30cm Micro USB cable, this power bank caters to various charging needs, providing flexibility and convenience for the Baasyir smart glasses system.

Connecting the hardware

Figure 3.3 illustrates the configuration of the Baasyir smart glasses system, emphasizing the crucial role played by the connectivity options of the Raspberry Pi 3 Model B+. The inclusion of dual-band wireless LAN and Bluetooth capabilities enhances the system's versatility and connectivity. The dual-band wireless LAN, operating at both 2.4GHz and 5GHz, ensures stable and efficient wireless communication. This is particularly valuable in scenarios where high data transfer rates are essential. The Bluetooth 4.2/BLE (Bluetooth Low Energy) feature expands the connectivity horizon, facilitating seamless communication with compatible devices. These connectivity options contribute to the overall functionality of the smart glasses system, allowing for wireless data transfer and

interaction with other devices.

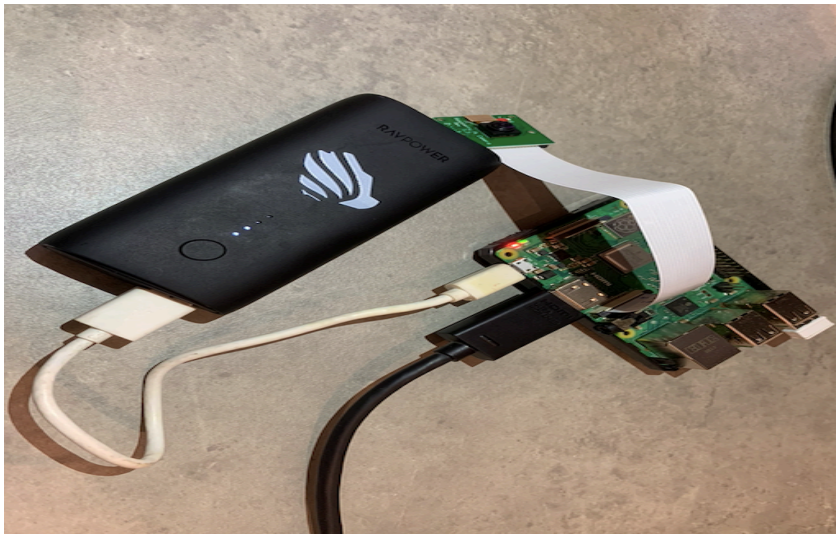


Figure 3.3: Connected hardware components enhancing the Baasyir smart glasses system.

Hardware setup steps

To initiate the hardware setup for the Baasyir smart glasses system, we established three essential connections:

1. We connected the Raspberry Pi 3 Model B+ with the Goui G-BELT10-K power bank using an appropriate power cable.
2. We established a connection between the Raspberry Pi and the Samsung monitor by using an HDMI cable.
3. We utilized the USB slots on the Raspberry Pi to connect both the wireless keyboard and mouse.

We opened the Raspberry Pi interface and set up the YOLOv7 Model through the terminal in the following steps: we first ensured that all required dependencies, including the appropriate version of Python, YOLOv7, and necessary libraries, were installed on the Raspberry Pi. Then, we downloaded the pre-trained weights for the Baasyir Smart glasses from our Google Drive. Next, we configured YOLOv7 for inference by adjusting the YOLOv7 configuration files, specifying the paths to the downloaded weights, and configuring parameters such as the confidence threshold. In the end, We ran YOLOv7 in real-time to assess its performance on live data from the smart glasses camera, utilizing the following command in the terminal.

Real-time object detection implementation with YOLOv7 on Raspberry Pi

To run real-time on our hardware, we needed to follow the previous real-time detection steps. However, we faced issues downloading the requirement.txt to the Raspberry Pi and encountered difficulties installing PyTorch directly on the Raspberry Pi. So, we decided

to adopt another approach, running our code on our local machine and using Raspberry Pi as a video source for real-time object detection. This is a common scenario when dealing with resource-intensive tasks on edge devices. These are the detailed steps:

1. Cloning YOLOv7 repository: to implement real-time object detection, we cloned the YOLOv7 repository, a widely used open-source object detection framework developed by Wong Kin Yiu [41].

2. Installing dependencies: we already have the Dependencies that we downloaded when we were training the model.

3. Raspberry Pi IP address retrieval: to use the Raspberry Pi as a video source for real-time object detection, we needed to determine its IP address. Several methods can be employed for this purpose:

   - Check router interface: we logged into our router's web interface and navigated to the 'Connected Devices' or 'Device List' section. We located our Raspberry Pi in the list and noted its IP address.
   - Use hostname -I Command: we connected the Raspberry Pi to a monitor and keyboard or accessed it via SSH. Opening a terminal on the Raspberry Pi, we ran the 'hostname -I' command, and the output displayed the IP address of the Raspberry Pi.
   - Check network configuration: if we had physical access to the Raspberry Pi, we checked its network configuration by running the 'ifconfig' command. We looked for the line starting with 'inet' or 'inet addr,' indicating the assigned IP address.

   We used the the second way after downloading SSH to access the Raspberry Pi from our local terminal and inserting the password we could use the Raspberry Pi without connecting it to a monitor our IP address is 192.168.100.41 as shown in figure 3.4.

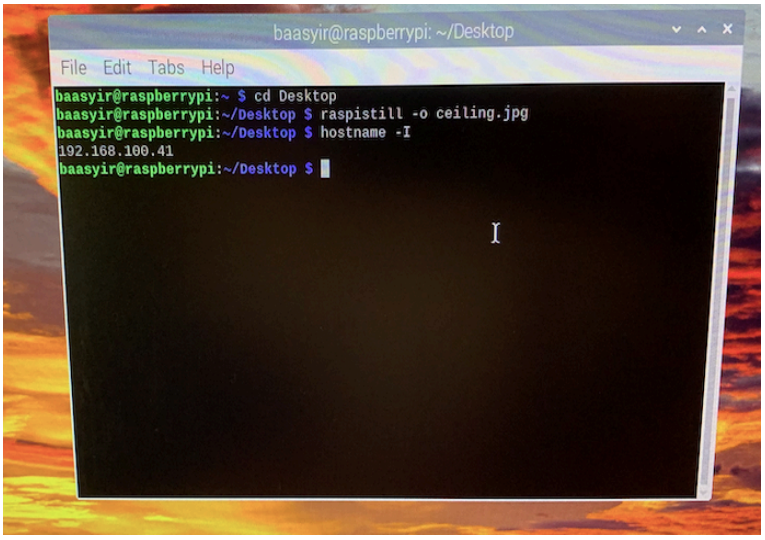


Figure 3.4: The process of identifying the IP address of our Raspberry Pi, a crucial step for establishing remote connections and interactions.

4. Run the modified Python code on your local machine: we leveraged the Raspberry Pi as a video source, and the heavy processing, including object detection, was done on our local machine. The video stream was transmitted over the network, allowing us to visualize the results in real-time.

   After connecting the camera to the camera port, as illustrated in figure 3.3, we accessed the Raspberry Pi interface by executing the command "sudo raspi-config". Subsequently, we navigated to the interface option and enabled the camera, as depicted in figure 3.5. Following this configuration, we restarted the Raspberry Pi and verified the camera functionality using the command "raspistill -o ceiling.jpg". This command captured an image of the current ceiling view from the camera, as shown in figure 3.6.

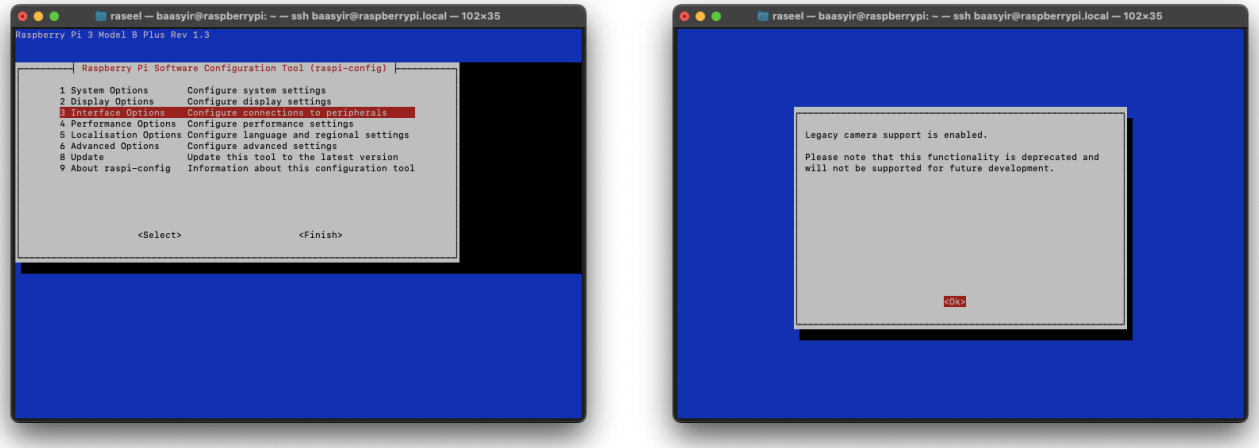


Figure 3.5: Setup and enable the camera.

Figure 3.6: Raspberry pi camera ceiling.

Following that, we checked the status of motion, as indicated in figure 3.7. We encountered some errors related to the log folder of the motion library, which we

addressed by changing the directory of the file to the home directory. Subsequently, we executed our Python code to run the model, modifying the source to our Raspberry Pi camera at http://192.168.100.41:8081, as demonstrated in figure 3.8.

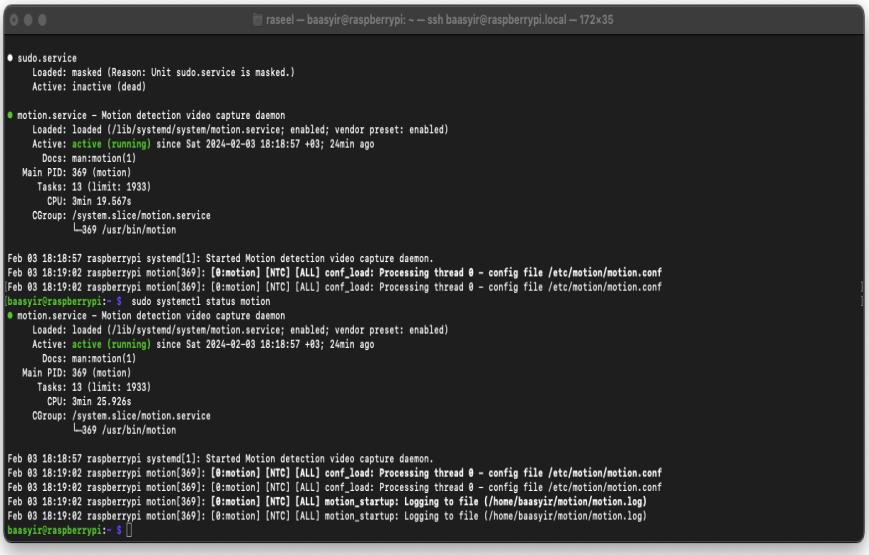


Figure 3.7: Raspberry Pi 3 Model B+ terminal displaying motion status.

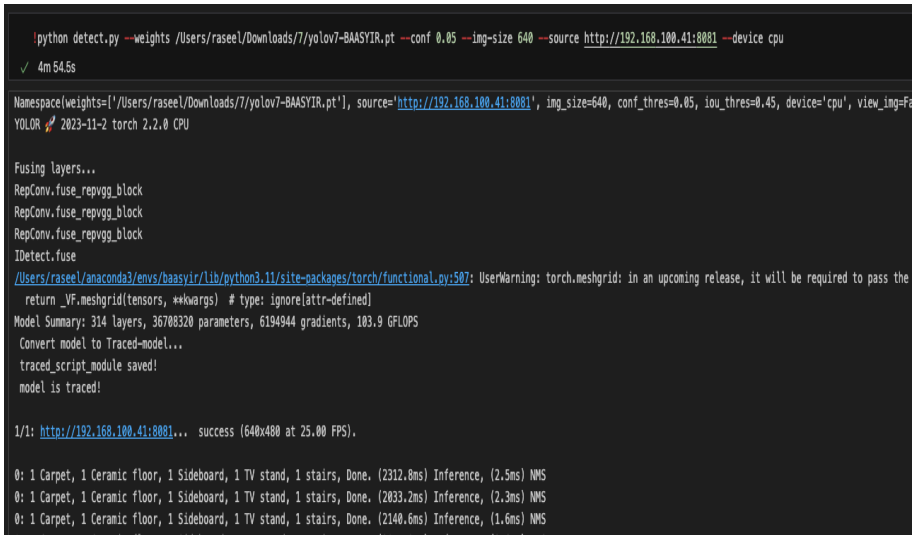


Figure 3.8: Raspberry Pi 3 Model B+ terminal displaying motion status after modifying the source.

3.4 Conclusion

In this chapter, we have presented the research methodology used. We have discussed the required steps to implement YOLO models in general for object detection along with required resources, such as programming language, framework, candidate dataset, model

evaluation metrics, and the hardware environment. The next chapter will cover the results of the project along with their discussions.

Chapter 4

Results and Discussion

4.1 Introduction

In the previous chapters, we provided a comprehensive background on the subject of object detection in the background chapter and discussed related work that has been done in this field in the related work chapter. After generally discussing the research efforts in the object detection field, we limited our focus to the use of the YOLOv7 in the object recognition field. Lastly, the steps required to implement the project are explained in detail in the research methodology chapter.

In this chapter, the results of the implemented deep learning model are presented and analyzed clearly and concisely, using appropriate tables, figures, and graphs to help illustrate the findings. Each result is attached to its discussion to provide a direct and clear understanding of the results, where the discussion provides an interpretation of the results, highlighting the key findings and their significance.

4.2 Data pre-processing experiments

To simplify the analysis and management of the dataset, the individual dataset files have been consolidated into a file with the extension .yolo. This file contains all the data about object detection activities conducted by various users, resulting in a dataset comprising 21,337 rows. Centralizing the data into a single file yields several advantages. This consolidation enhances the efficiency of data analysis, allowing for seamless filtering and sorting based on specific criteria, such as Object Coordinates, Object Classes, Confidence Scores, Timestamps, User IDs, Image Metadata, etc.

We conducted class remapping within our Baasyir dataset to standardize objects belonging to the same category, such as merging 'carpet' and 'mat.' Additionally, we unified disparate class names, simplifying them into a single representation. As you can see in figure 4.1 we remapped the sikat gigi class which means the toothbrush in Indonesia into a toothbrush and combined the images of two classes.

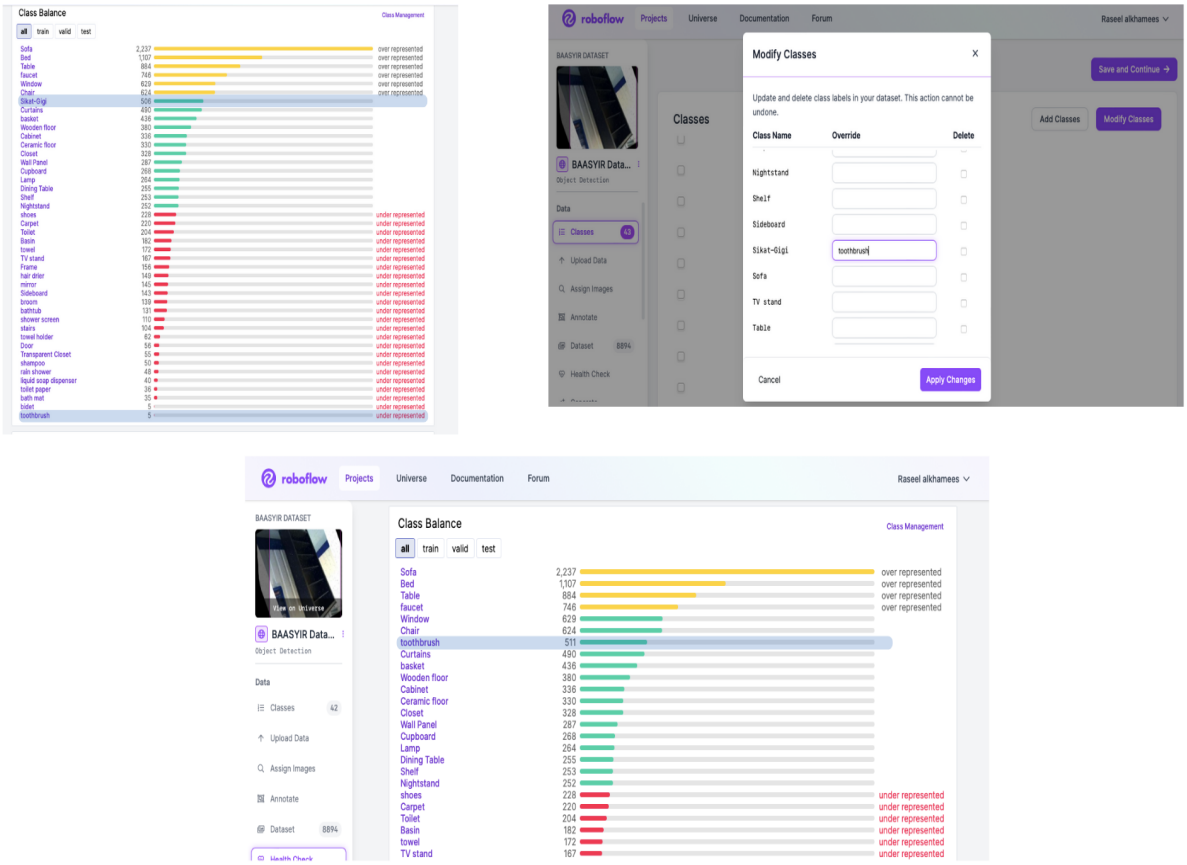


Figure 4.1: Bassyir dataset before and after feature selection.

4.3 Baasyir dataset experiments

The experiments were conducted using YOLOv7 and YOLOv7-tiny models on Baasyir dataset. The outcomes of these experiments are illustrated through various graphs, which assess the models based on evaluated metrics such as mAP, F1-score, recall, precision, and confusion matrix. Further details regarding the evaluation metrics can be found in section 3.2.

4.3.1 YOLOv7 experiments

After training the YOLOv7 model on our Baasyir dataset, which required 24.294 hours to complete 100 epochs, the results were illustrated using the forthcoming graphs in the following figures 4.2, 4.3 and 4.4. The confusion matrix presented in figure 4.2 visually illustrates the performance of the YOLOv7 model on our Baasyir dataset. Dark squares along the diagonal, particularly for items like 'Bed' and 'Chair', indicate high accuracy. However, for some items such as 'basket' and 'shower screen', there are lighter squares or off-diagonal markings, suggesting less accurate predictions.

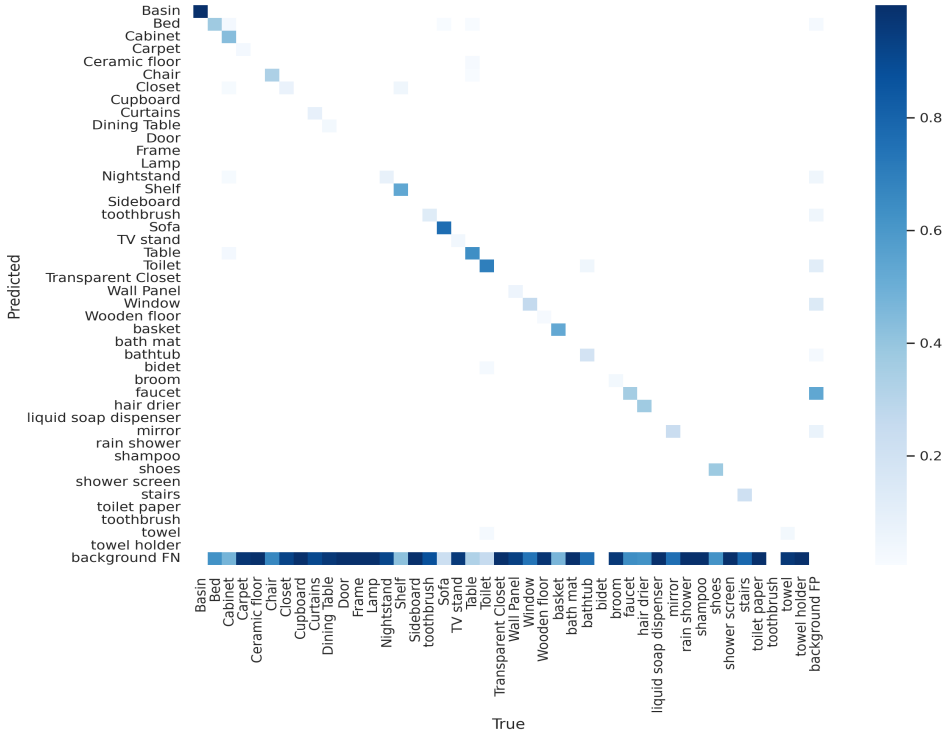
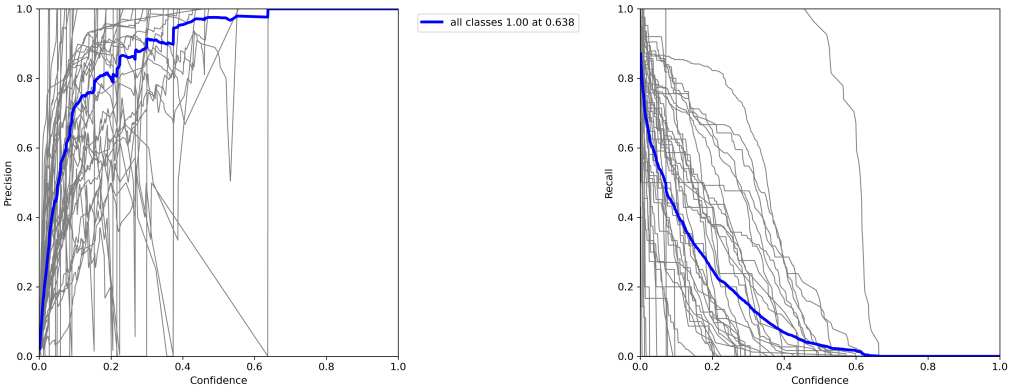


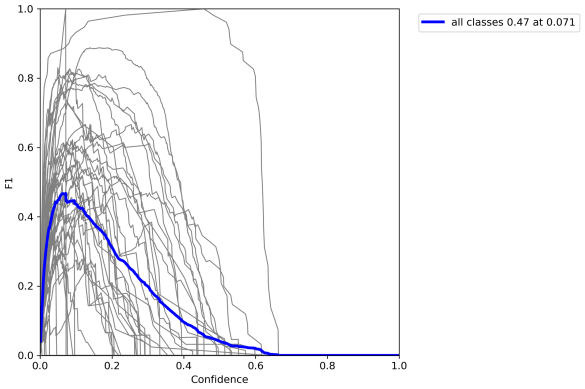
Figure 4.2: Confusion matrix for Baasyir dataset using YOLOv7 model

The experimental results in figures 4.3a, 4.3b and 4.3c are presented in multiple graphs where the x-axis represents the confidence and the y-axis represents the evaluated metrics such as Precision, Recall, and F1-score. In figure 4.3a, the precision curve showcases a perfect precision (1.00) for 'all classes' at a confidence threshold of approximately 0.638. This indicates that when the model is more than 63.8% confident, it demonstrates an exceptionally high level of accuracy in its predictions. The recall curve, as depicted in figure 4.3b, indicates that the recall for 'all classes' begins at a very high value (0.87) at the lowest confidence threshold. This implies that the model can detect most items even when it is least selective. Then, figure 4.3c depicts the F1 score curve, revealing that the F1 score for 'all classes' reaches its peak at 0.47. This peak occurs at a low confidence threshold of 0.071, indicating a reasonable balance of precision and recall across all classes at this threshold.



(a) Precision curve

(b) Recall curve



(c) F1-score curve

Figure 4.3: Precision, Recall and F1-score of Baasyir dataset on YOLOv7 model

Figure 4.4 illustrates the precision-recall curve, indicating a mean Average Precision (mAP) of 0.494 for 'all classes' at a threshold of 0.5. This metric suggests moderate overall precision and recall for the model across all classes.

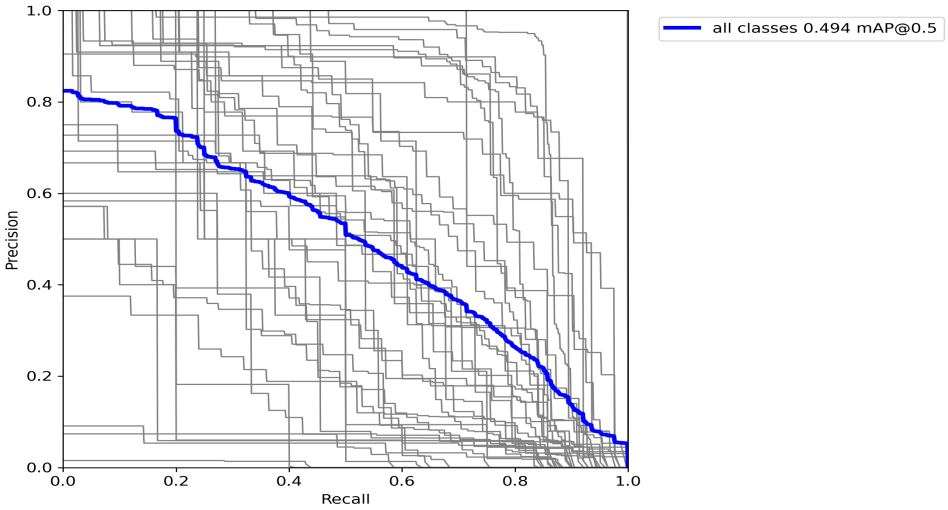


Figure 4.4: mAP@0.5 curve for Baasyir dataset using YOLOv7 model.

4.3.2 YOLOv7-tiny experiments

After training the YOLOv7-tiny model on our Baasyir dataset, which required 9.296 hours to complete 100 epochs, the results were illustrated using the forthcoming graphs in the following figures 4.5, 4.6 and 4.7. As illustrated in figure 4.5, the confusion matrix visually represents the prediction outcomes for various household items. Dark squares along the diagonal, particularly for items such as 'Bed', 'Cabinet', 'Chair', and 'TV Stand', indicate accurate predictions. The intensity of the squares reflects the prediction accuracy, with darker squares signifying higher accuracy for the corresponding items.

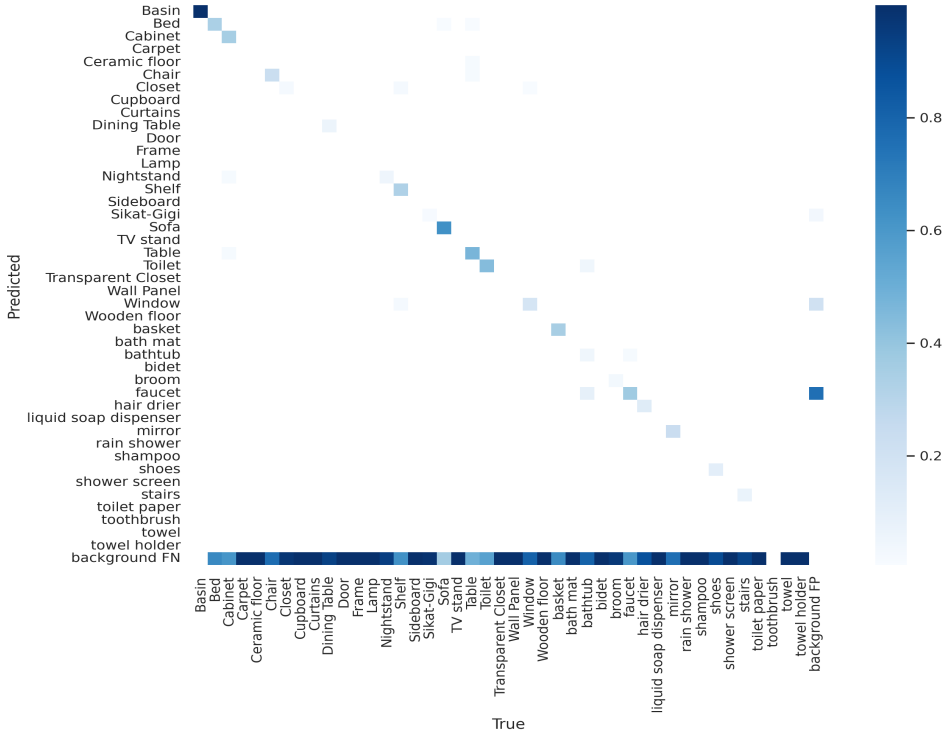
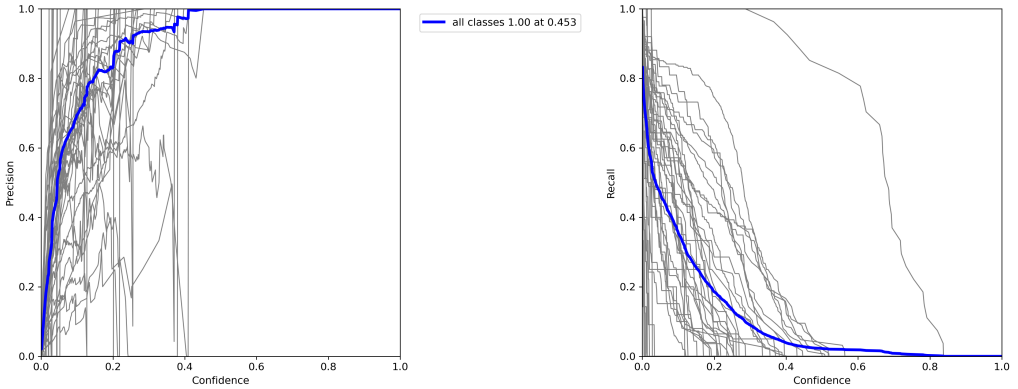


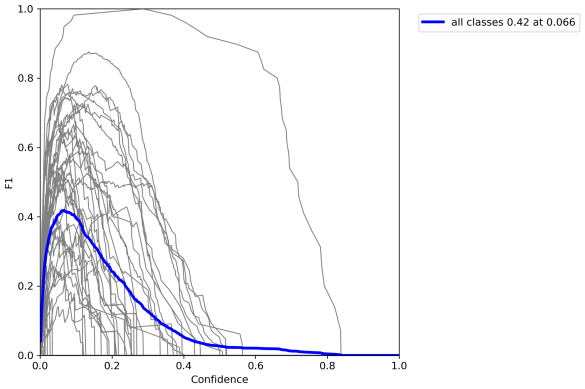
Figure 4.5: Confusion matrix for Baasyir dataset using YOLOv7-tiny model.

The experimental results in figures 4.6a, 4.6b and 4.6c are presented in multiple graphs where the x-axis represents the confidence and the y-axis represents the evaluated metrics such as Precision, Recall, and F1-score. The precision curve, as depicted in figure 4.6a, reveals that the model achieves a precision of 1.00 for 'all classes' at a confidence threshold of 0.453. This indicates that the model's predictions are highly accurate when it exhibits moderate confidence. In 4.6b, the recall curve highlights the model's performance in achieving a high recall of 0.83 for 'all classes' at the lowest confidence threshold (0.000). This suggests the model's capability to detect the majority of items when it prioritizes inclusivity. Individual lines on the curve likely represent class-specific recalls at different confidence levels. In figure 4.6c, the F1 score curve demonstrates that the F1-score for 'all classes' reaches its peak at 0.42, occurring at a relatively low confidence threshold of 0.066. This suggests that the model's optimal balance between precision and recall is achieved when it is not excessively confident.



(a) Precision curve

(b) Recall curve



(c) F1-score curve

Figure 4.6: Prcesion, Recall and F1-score for Baasyir dataset using YOLOv7-tiny model.

Figure 4.7 displays the precision-recall curve, offering insights into the trade-off between precision and recall across different classes. The mean Average Precision (mAP) for 'all classes' is 0.427 at a detection threshold of 0.5. The blue line represents the average precision aggregated across all classes.

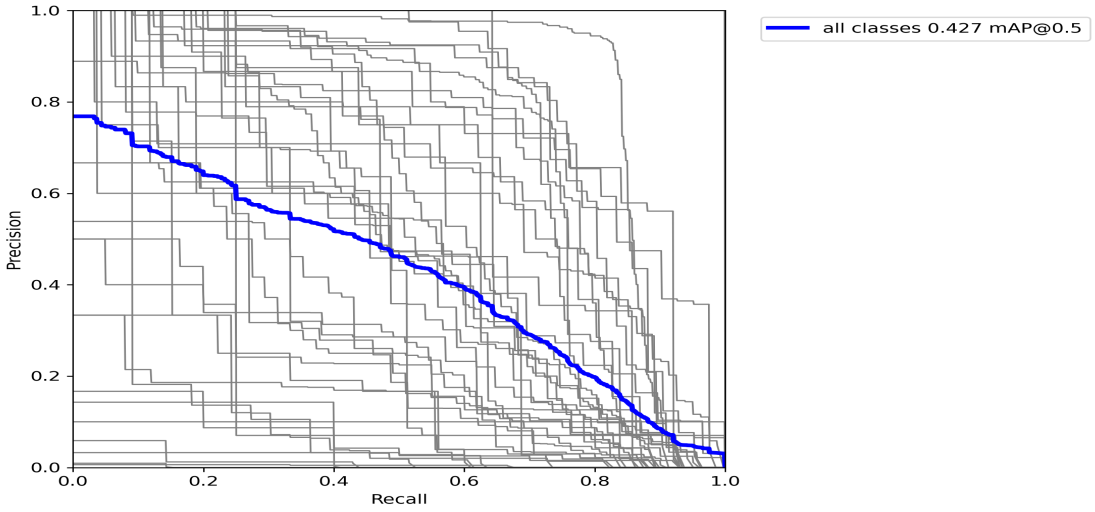


Figure 4.7: mAP@0.5 curve for Baasyir dataset using YOLOv7-tiny model.

4.3.3 Results and discussion

We conducted training on our Baasyir dataset, recognized as the Baasyir dataset is an image dataset. The study involved a comparative performance analysis between two models, YOLOv7 and YOLOv7-tiny, using the Baasyir dataset, as presented in table 4.1.

Table 4.1: Results obtained from the YOLOv7 and YOLOv7-tiny models, conducted on Baasyir Dataset

Metrics	**YOLOv7 model**	**YOLOv7-tiny model**
Precision	58.5%	61.5%
Recall	49.8%	43.4%
mAP	49.4%	42.7%

We observed a precision of 61.5% for the YOLOv7-tiny model, showcasing comparable accuracy to the YOLOv7 model despite the reduced complexity of the tiny variant. The recall, however, was slightly lower at 43.4%, indicating a trade-off between model simplicity and recall performance. The mean Average Precision (mAP) for YOLOv7-tiny was 42.7%, slightly below the mAP of YOLOv7. This discrepancy in mAP might be attributed to the efficiency optimizations implemented in the YOLOv7-tiny model, demonstrating a nuanced balance between model size and performance.

YOLOv7 model trained on our Baasyir dataset demonstrates strong performance across various classes, particularly when operating with a low confidence threshold. The model exhibits high recall, indicating its capability to detect most instances of the trained classes. The F1-score suggests a good balance of precision and recall, though there may be opportunities for further optimization. The precision curve emphasizes the model's exceptional accuracy at higher confidence levels, while the precision-recall curve's moderate mAP score hints at potential improvements in precision with further model refinement.

YOLOv7-tiny trained on our Baasyir dataset showcases a robust ability to detect various household items, though with variations in prediction accuracy across different classes. While the model demonstrates general reliability, there is room for improvement, particularly in enhancing precision and recall for specific classes. The notable high recall at low confidence levels indicates the model's effectiveness in capturing a significant portion of the items it's trained to detect. However, the F1 score's peak at a lower confidence level suggests a potential inclination towards false positives, signaling a potential for refinement by adjusting the confidence threshold.

4.4 Furniture-Detection dataset experiments

The experiments were conducted using YOLOv7 and YOLOv7-tiny models on Furniture-Detection dataset. The outcomes of these experiments are illustrated through various graphs, which assess the models based on evaluated metrics such as mAP, F1-score, recall, precision, and confusion matrix. Further details regarding the evaluation metrics can be found in section 3.2.

4.4.1 YOLOv7 experiments

After training the YOLOv7 model on our Furniture-Detection dataset, which required 11.520 hours to complete 100 epochs, the results were illustrated using the forthcoming graphs in the following figures 4.8, 4.9 and 4.10. The confusion matrix presented in figure 4.8 visually illustrates the performance of the YOLOv7 model on Furniture-Detection dataset. The YOLOv7 model demonstrates excellent prediction accuracy for items such as 'ac', 'bunk_beds', and 'closet' achieving respective confidence values of 0.91, 0.90, and 0.92. These results underscore the model's capability to accurately identify these items, demonstrating its robust performance in these specific categories.

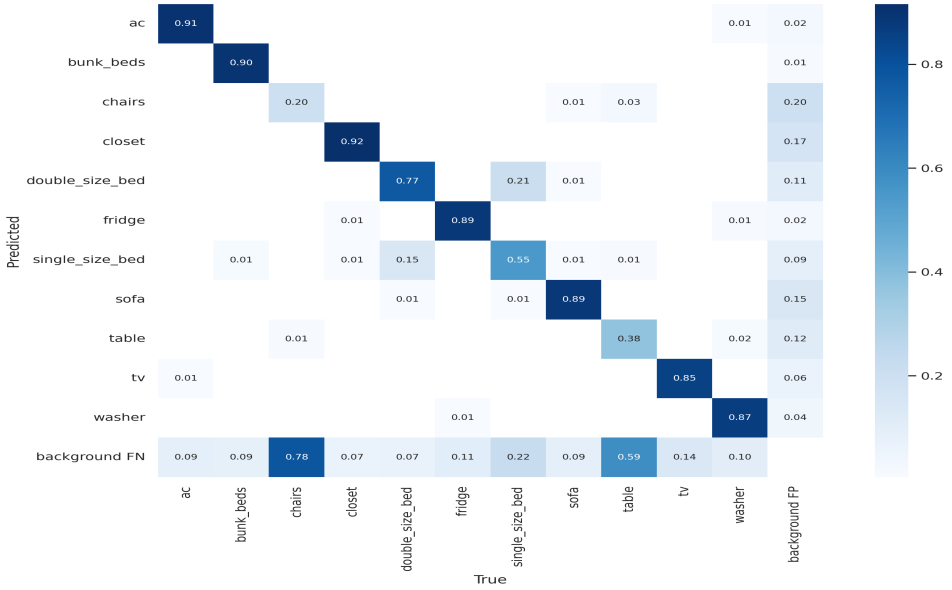
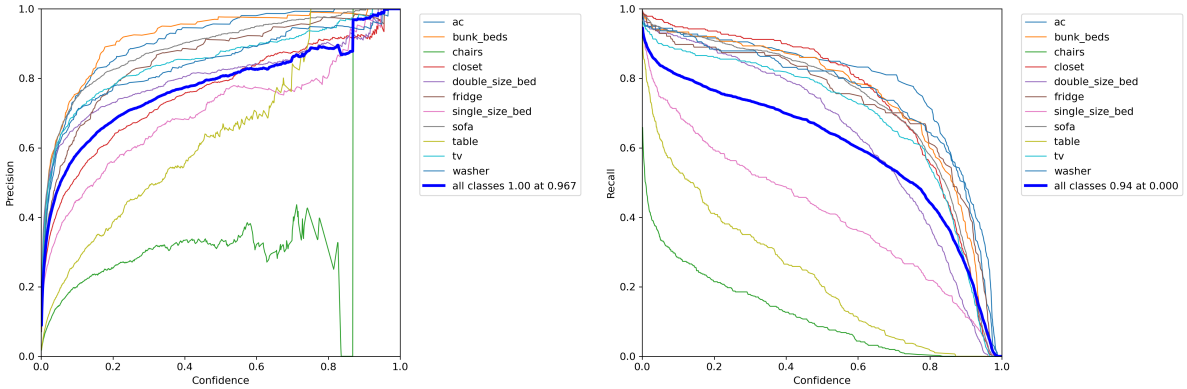


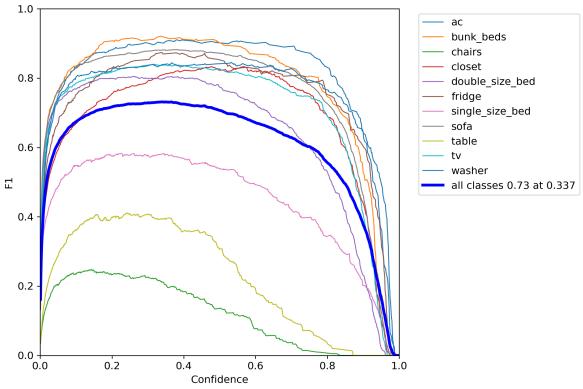
Figure 4.8: Confusion matrix of the furniture dataset of YOLOv7 model.

As shown in figure 4.9a, the Precision Curve for 'all classes' reaches perfection (1.00) at a high confidence threshold of 0.967. This perfect precision score suggests that the model's. Figure 4.9b illustrates the Recall Curve, where for 'all classes', the recall commences at a high rate (0.94) even at the most lenient confidence threshold. This indicates that the model maintains a substantial detection rate across all classes when applying minimal selectivity in its predictions. The F1-Score Curve, depicted in figure 4.9c, reveals that the optimal F1 score for 'all classes' is around 0.73 at a confidence threshold of about 0.337. This performance marks an improvement over previous iterations, indicating enhanced overall effectiveness of the YOLOv7 model.



(a) Precision curve

(b) Recall curve



(c) F1-score curve

Figure 4.9: Prcesion, Recall and F1-score for Furniture-Detection dataset using YOLOv7 model.

The Precision-Recall curve, depicted in figure 4.10, provides an overall mean average precision (mAP) of 0.753 for 'all classes' at a threshold of 0.5. This metric, combined with high scores in specific classes such as 'ac,' 'bunk_beds', 'closet', 'fridge', and 'sofa' highlights the model's proficiency in accurately predicting a diverse range of items.

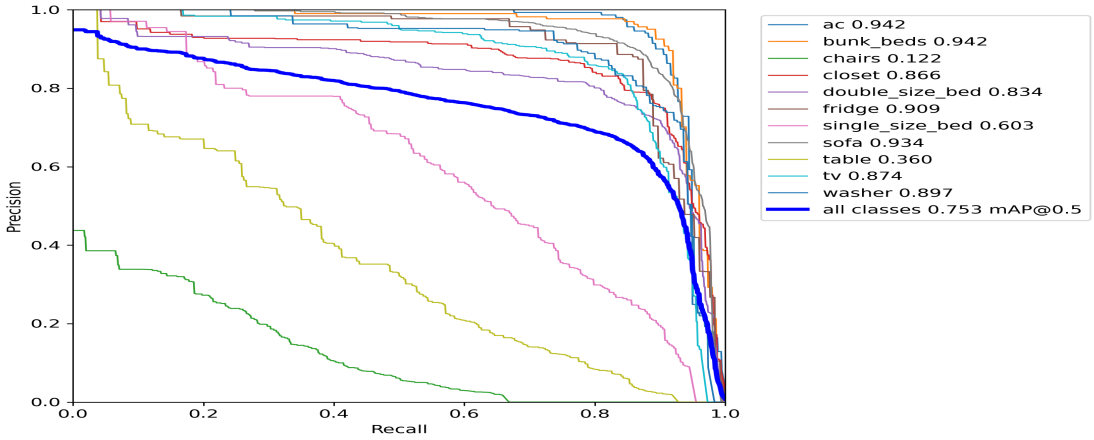


Figure 4.10: Precision-recall curve of the furniture dataset of YOLOv7 model.

The YOLOv7 model's particularly precise in recognizing items like 'ac', 'bunk_beds', 'closet', 'fridge', and 'sofa', demonstrating high precision and recall rates. The model's exceptional precision at higher confidence levels and its balanced overall F1-score across all classes suggest that it is a well-adjusted and robust solution for detecting a broad spectrum of furniture items.

4.4.2 YOLOv7-tiny experiments

After training the YOLOv7-tiny model on the Furniture-Detection dataset, which required 5.570 hours to complete 100 epochs, the results were illustrated using the forthcoming graphs in the following figures 4.11, 4.12 and 4.13. As depicted in figure 4.11, the model demonstrates exceptional accuracy in identifying 'ac' and 'closet'. This is evident from the dark squares along the diagonal of the confusion matrix, corresponding to values of 0.83 and 0.88, respectively. These findings suggest that the model excels at accurately distinguishing between these two items within the dataset.

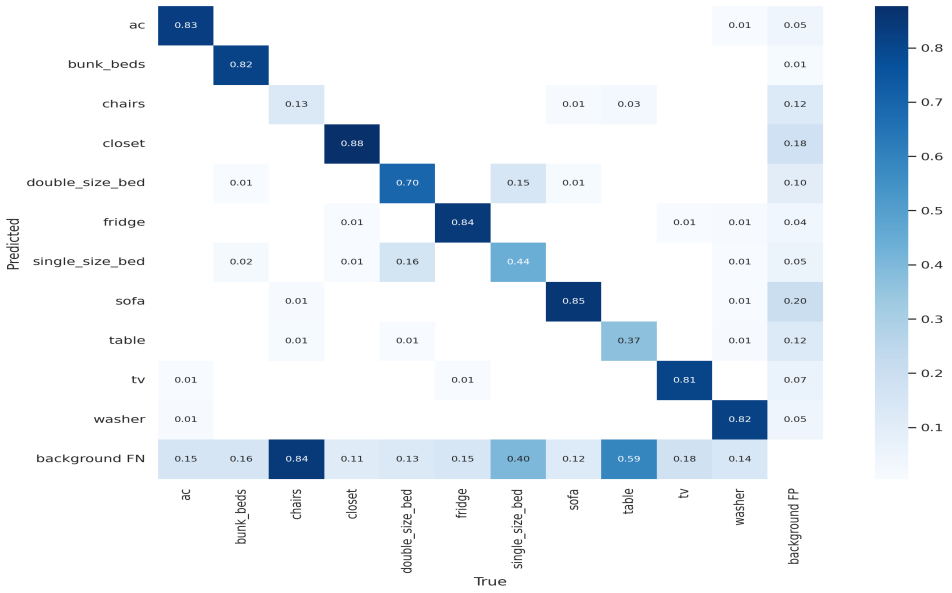
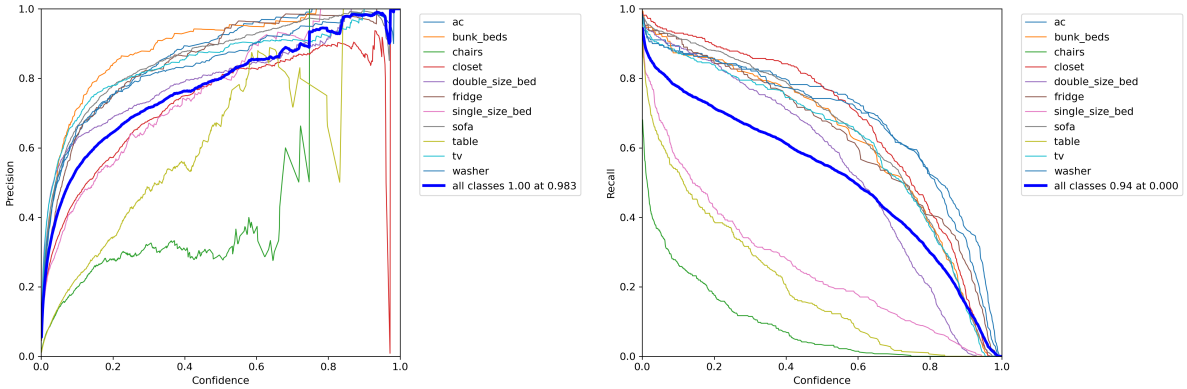


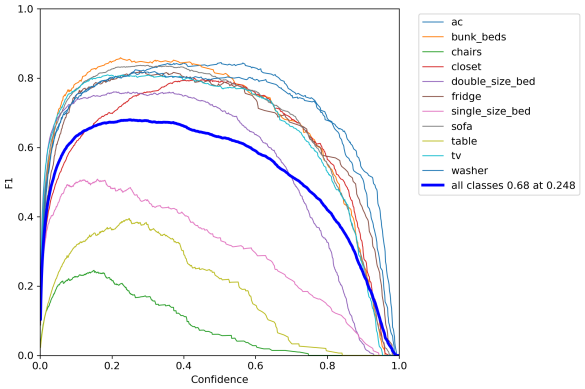
Figure 4.11: Confusion matrix of the furniture dataset of YOLOv7-tiny model.

In figure 4.12a, the Precision curve demonstrates consistently high precision for 'ac,' 'closet,' and 'fridge,' even at lower confidence levels. For 'all classes,' precision achieves perfection at the highest confidence levels, though this could potentially indicate overfitting to highly certain predictions. As observed in figure 4.12b, the Recall Curve indicates that 'ac' and 'closet' exhibit high recall at low confidence levels. Moreover, the recall for 'all classes' is remarkably high (0.94) at the lowest confidence threshold, suggesting that the model can detect nearly all furniture items when it is least restrictive in its predictions. In figure 4.12c, the model attains its peak performance for 'all classes' at a confidence threshold of around 0.248. This indicates a balanced precision and recall for all furniture items collectively, marking an optimal point for the model's overall accuracy.



(a) Precision curve

(b) Recall curve



(c) F1-score curve

Figure 4.12: Prcesion, Recall and F1-score for Furniture-Detection dataset using YOLOv7-tiny model.

The Precision-Recall curve, shown in figure 4.13, reveals the best mean average precision (mAP) scores in 'bunk_beds' (0.898), 'sofa' (0.890), and 'ac' (0.883). The overall mAP for 'all classes' is 0.711 at a detection threshold of 0.5, showcasing good yet improvable aggregate performance across the board.

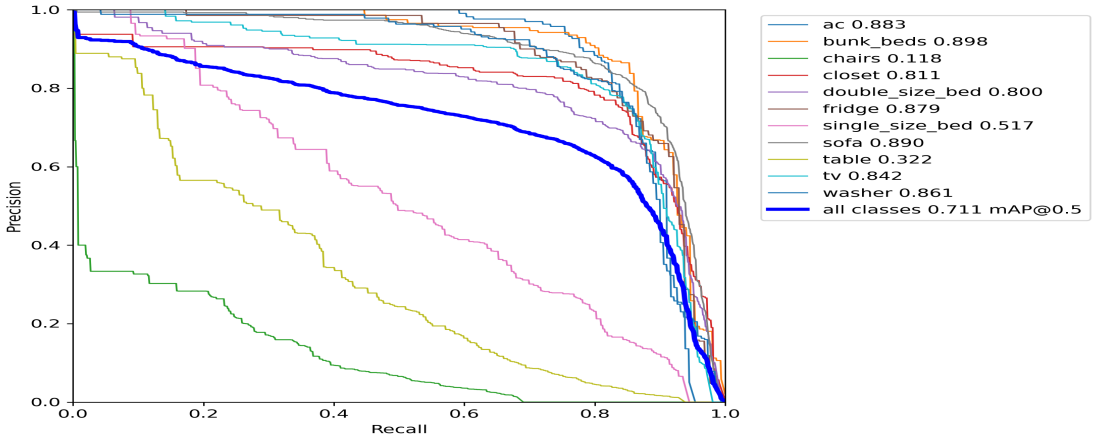


Figure 4.13: Precision-recall curve of the furniture dataset of YOLOv7-tiny model.

These images collectively show that while the model excels at identifying items like 'ac', 'bunk_beds', and 'closet', there is a notable potential for enhancement in its ability to consistently identify other items, particularly 'chairs' and 'table'. The overall performance across all classes is well-balanced at a moderate confidence level, indicating a robust model with specific areas for improvement.

4.4.3 Results and discussion

We conducted training and performed a comparative performance analysis between two models, YOLOv7 and YOLOv7-tiny, using the Furniture-Detection dataset, as shown in Table 4.2.

Table 4.2: Results obtained from the YOLOv7 and YOLOv7-tiny models, conducted on Furniture-Detection Dataset

Metrics	**YOLOv7 model**	**YOLOv7-tiny model**
Precision	75.1%	68.6%
Recall	72.4%	69.0%
mAP	75.3%	71.1%

We observed a precision of 68.6% for the YOLOv7-tiny model, which is reasonably close to the 75.1% precision of the more complex YOLOv7 model. This demonstrates that the YOLOv7-tiny variant maintains a high level of accuracy, despite its reduced complexity. The recall for the YOLOv7-tiny model was 69.0%, slightly lower compared to the YOLOv7 model's recall of 72.4%, indicating a trade-off between the simplicity of the tiny variant and its ability to detect all relevant instances. The mean Average Precision (mAP) of YOLOv7-tiny stood at 71.1%, which, while commendable, is marginally lower than the YOLOv7's mAP of 75.3%. These differences in mAP suggest that the efficiency optimizations in the YOLOv7-tiny model achieve a balanced compromise between the streamlined model architecture and its overall detection performance

4.5 Real-time experiments

In this section, we presented the real-time performance of the YOLOv7 model using the Baasyir dataset. The experiments were conducted within the PyCharm Community Edition as the development environment. The required dependencies for YOLOv7 were installed, and we employed the pre-trained YOLOv7_Baasyir.pt weights for the evaluation.

Model Initialization

The YOLOv7 model was initialized with the pre-trained weights we made YOLOv7_Baasyir.pt. This model was fine-tuned to detect objects present in the Baasyir dataset, facilitating real-time object detection.

Real-Time Inference

The camera feed was streamed into the YOLOv7 model to perform real-time object detection. The following steps outline the real-time inference process:

1. Environment configuration: we set up the necessary dependencies and YOLOv7 framework in the PyCharm CE environment.

2. Weight loading: we loaded the pre-trained YOLOv7_Baasyir.pt weights into the model.

3. Camera initialization: the mobile camera was set up and connected to the computer to capture real-time frames.

4. Object detection: each frame was passed through the YOLOv7 model for object detection.

5. Results display: we displayed real-time results, including bounding boxes and class labels, on the output video stream.

Some real-time results on the YOLOv7 and YOLOv7 Baasyir models are presented in figure 4.14 and figure 4.15, where confidence score is a probability value ranging from 0 to 1. A score of 0 indicates low confidence, meaning the model is uncertain about the detection, while a score of 1 indicates high confidence, signifying the model is very certain about the detection.

Figure 4.14: Real-time results for YOLOv7 model.

Figure 4.15: Real-time results for YOLOv7 BAASYIR model.

Results and discussion

The YOLOv7 model, designed for general object detection, exhibited higher confidence scores for specific detected objects, such as Toilet and Bed, in comparison to the YOLOv7 Baasyir model. The YOLOv7 Baasyir model, tailored for indoor objects, demonstrated a broader range of confidence scores for a diverse set of objects, encompassing both high and moderate values (refer to table 4.3 and figure 4.15 and 4.14).

Table 4.3: Results obtained from experiments with the YOLOv7 and YOLOv7 Baasyir models, focusing on specific object classes.

Detected Objects and Confidences	**Toilet**	**Hair Dryer**	**Bed**	**Sideboard**	**Window**	**Shoes**
YOLOv7	0.95	-	0.79	-	-	-
YOLOv7 Baasyir	0.35	0.40	0.35	0.46	0.40	0.41

In terms of model-specific performance, YOLOv7 Baasyir showcased detection results for additional objects like Hair Dryer, Sideboard, Window, and Shoes, indicating a more comprehensive recognition spectrum compared to YOLOv7's narrower focus. Variations in confidence scores between the models can be attributed to differences in training data, class distribution, and specific optimizations made in YOLOv7 Baasyir to cater to indoor scenarios and assist visually impaired individuals.

Practical considerations highlight the importance of choosing the right model for real-time applications based on specific use cases and requirements. YOLOv7 may be suitable for more general indoor object detection tasks, while YOLOv7 Baasyir, with its broader object recognition spectrum and varied confidence levels, proves beneficial for scenarios involving indoor environments, aiming to assist visually impaired individuals

with detailed object identification.

4.6 Comparison with previous researches

In this section, we focused on comparing the performance of various YOLOv7 models and understanding their practical implications. Table 4.4 summarizes the mean average precision (mAP) values of the YOLOv7, YOLOv7-tiny, YOLOv7_Furniture, YOLOv7-tiny_Furniture, YOLOv7_Baasyir, and YOLOv7-tiny_Baasyir models.

Table 4.4: Comparison of different object detection models on specific datasets, highlighting mean Average Precision (mAP) performance.

Model	**Dataset**	**mAP**
YOLOv7	MS COCO	52.0%
YOLOv7-tiny	VisDrone-2019	34.5%
YOLOv7_Furniture	Furniture	75.3%
YOLOv7-tiny_Furniture	Furniture	71.1%
YOLOv7_Baasyir	Baasyir	49.4%
YOLOv7-tiny_Baasyir	Baasyir	42.7%

After evaluation of different YOLOv7 models on their mean average precision (mAP) reveals varying performances across distinct datasets and objectives. The YOLOv7 model, designed for general-purpose object detection, achieves a commendable mean average precision of 52.0%, indicating its efficacy in accurately identifying objects across diverse scenarios and datasets. YOLOv7_Furniture, specializing in the detection of furniture-related objects, outperforms the general-purpose YOLOv7 model with a significantly higher mAP of 75.3%, emphasizing the model's proficiency in recognizing and localizing furniture items within its dedicated dataset. YOLOv7 Baasyir, tailored for indoor object recognition and crafted to assist visually impaired individuals, achieves a mAP of 49.4%. While slightly lower than the general-purpose YOLOv7, this model demonstrates competitive performance within the context of indoor scenarios. YOLOv7-tiny, a compact version of YOLOv7, achieves a mAP of 34.5%. Despite its reduced computational footprint, this model offers a reasonable level of performance, making it suitable for scenarios where resource constraints are a primary consideration.

The results underscore the trade-offs between general-purpose and specialized models, with YOLOv7 Furniture excelling in furniture-related object detection and YOLOv7 Baasyir maintaining competitive performance in indoor scenarios. YOLOv7_Baasyir, specifically designed for indoor environments and aiding visually impaired individuals, demonstrates its relevance in recognizing specific objects within room contexts, contributing to its effectiveness where detailed object identification is crucial. The choice of a YOLOv7 model depends on specific application requirements, with YOLOv7_Furniture excelling in furniture detection tasks and YOLOv7_Baasyir providing a balanced solution for general object recognition and specialized indoor applications. Additionally, YOLOv7-tiny, despite its reduced mean average precision, offers a resource-efficient al-

ternative suitable for applications with limited computational capabilities. In conclusion, the discussion illuminates the diverse capabilities of different YOLOv7 models, emphasizing the need to tailor models based on specific objectives and datasets, providing valuable insights for selecting the most suitable model for real-world applications.

4.7 Conclusion

In conclusion, this chapter extensively examined the results of experiments utilizing YOLOv7 and YOLOv7-tiny models on two datasets: Baasyir, encompassing general object detection scenarios, and Furniture-Detection, focusing on indoor furniture recognition. Both models exhibited strong performance in identifying objects in the Baasyir dataset, with YOLOv7 demonstrating higher precision and recall at elevated confidence levels. The Furniture-Detection dataset showcased remarkable precision and recall for both models, with YOLOv7 achieving a slightly higher mean Average Precision (mAP). A real-time experiment on a mobile camera feed revealed variations in confidence scores and object detection capabilities, emphasizing YOLOv7's higher confidence scores for specific objects and YOLOv7 BAASYIR's broader recognition spectrum in indoor scenarios. Overall, the experiments provide valuable insights, guiding refinements, and optimizations for specific applications, laying the groundwork for future research and advancements in object detection.

Chapter 5

Conclusion and Future Work

5.1 Introduction

In this chapter, we discussed the main contributions of our Baasyir smart glasses system and outlined future work. Beginning with a comprehensive introduction in chapter 1, we outlined the project's aim, objectives, and methodology to establish the scope of our work. Providing essential background knowledge and terminologies related to smart glasses using DL, we laid the foundation for our research. Moving on to chapter 2, we explored the background and related work, summarizing previous research in our domain. Chapter 3 further expanded with two distinct sections covering theoretical formalization, algorithm design, and experimental design. The latter detailed the project's workflow plan and essential resources. In Chapter 4, the results of our implemented deep learning model were presented and analyzed rigorously, supported by tables, figures, and graphs for clarity. Each result was paired with its discussion to provide a direct interpretation, highlighting key findings and their significance.

5.2 Contributions

In this comprehensive project, our team undertook a multifaceted approach, starting with diligent data collection and thorough data preprocessing. This foundational step involved organizing and preparing the dataset, ensuring it was conducive to effective model training. Our learning journey extended to deep learning models, specifically YOLOv7, a powerful tool for object detection tasks. Leveraging resources such as Google Colab, we honed our skills in deploying models efficiently in a cloud environment with access to GPU resources. The training phase was not limited to a single dataset; rather, we extended our efforts to diverse datasets, including our own Baasyir dataset. This strategy not only showcased our adaptability but also ensured that the trained model could generalize well across various real-world scenarios. Transitioning from the virtual realm to the physical, our project included the construction of the hardware system and the successful integration of the trained YOLOv7 model onto it. This amalgamation of skillsfrom data collection and preprocessing to model training and hardware integrationunderscores a holistic understanding of the end-to-end process in developing a computer vision ap-

plication. The ability to seamlessly traverse the realms of software, encompassing deep learning models, and hardware integration showcases a multidisciplinary approach. This project lays a solid foundation for future endeavors at the intersection of deep learning and computer vision, embodying a practical synthesis of theory and hands-on application.

5.3 Future Work

In our future work, we aim to enhance the dataset by incorporating additional variants, specifically focusing on small everyday objects such as pens, cups, and other items that play a vital role in the daily lives of individuals with impaired people. This expansion is intended to refine and diversify the model's ability to detect and interact with a broader range of objects, contributing to its practical utility in real-world scenarios. Furthermore, we have aspirations to optimize and develop the hardware, aiming to make it more compact and impactful in everyday life. The goal is to create a hardware system that seamlessly integrates into daily routines, providing valuable assistance to individuals with specific needs. This involves not only downsizing the hardware but also refining its functionalities to enhance user experience and overall effectiveness. In essence, our future work is geared towards continuous improvement, both in terms of the model's capabilities through dataset enrichment and the hardware's design to better align with the needs of individuals with impaired dependencies. By addressing these aspects, we strive to create a more robust and user-friendly system that can make a meaningful impact in the lives of those who benefit from assistive technologies.

5.4 Conclusion

Our research into the development of the Baasyir smart glasses system has been a dynamic journey encompassing deep learning, computer vision, and hardware integration. Starting with meticulous data collection and model training using YOLOv7, we seamlessly transitioned from virtual to physical with successful hardware integration.

The multidisciplinary approach showcased our team's adaptability and a holistic understanding of the end-to-end process. Notably, the YOLOv7 model demonstrated impressive performance with a precision of 58.5%, recall of 49.8%, and an mAP of 49.4%. The YOLOv7-tiny model, despite reduced complexity, maintained comparable accuracy with a precision of 61.5% and a recall of 43.4%, albeit with a slightly lower mAP of 42.7%.

Looking ahead, our results underscore the potential for widespread applications and lay a robust foundation for future exploration at the intersection of deep learning and computer vision. Continuous refinement of both software and hardware components will be critical for staying at the forefront of innovation.

Appendixes

The project code and implementation can be accessed on GitHub at the provided BAASIYR github repository

The code is written in Python and is uploaded as a Colab notebook file. Additionally, the dataset is made available on the Roboflow provided BAASYIR dataset

Reference

[1] IBM, "Machine learning." https://www.ibm.com/design/ai/basics/ml/, 2023.

[2] IBM. https://www.ibm.com/topics/neural-networks, 2023.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[4] R. Girshick, "Fast r-cnn," *arXiv*, vol. 1504, 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv*, vol. 1506, 2016.

[7] R. P. Foundation. https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/4, accessed January 28, 2024.

[8] W. H. Organization, "World health organization." https://www.who.int/, 2023. Who.int.

[9] P. D. Wardaya, "Support vector machine as a binary classifier for automated object detection in remotely sensed data," *IOP Conference Series: Earth and Environmental Science*, vol. 18, p. 012014, feb 2014.

[10] N. Gupta, S. K. Gupta, R. K. Pathak, V. Jain, P. Rashidi, and J. S. Suri, "Human activity recognition in artificial intelligence framework: a narrative review," vol. 55, no. 6, p. 47554808, 2018.

[11] W.-J. Chang, L.-B. Chen, C.-H. Hsu, J.-H. Chen, T.-C. Yang, and C.-P. Lin, "Medglasses: A wearable smart-glasses-based drug pill recognition system using deep learning for visually impaired chronic patients," *IEEE Access*, vol. 8, pp. 17013–10724, 01 2020.

[12] A. Ali, S. Rao, S. Ranganath, A. T S, and R. R. Guddeti, "A google glass-based real-time scene analysis for the visually impaired," *IEEE Access*, vol. PP, pp. 1–1, 12 2021.

[13] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, vol. 1804, 2018.

[14] Uijlings and al., "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, 09 2013.

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV 2016*, pp. 21–37, Springer International Publishing, 2016.

[16] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv*, vol. 1612, 2016.

[17] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv*, vol. 2004, 2020.

[18] R. Couturier, M. H. N. Noura, O. Salman, and A. Sider, "A deep learning object detection method for an efficient clusters initialization," *arXiv*, 2021.

[19] L. Manikandan, R. Malavika, B. Anjali, and S. Chandana, "Section a-research paper object detection using yolo v5 eur," *European Chemical Bulletin Access*, vol. 53, p. 62266233, 01 2023.

[20] Chuyi and et al., "Yolo v6: A single stage object detection framework for industrial applications," *arXiv*, vol. 53, p. 62266233, 2022.

[21] N. Barazida, "Yolov6: next-generation object detection review and comparison," *towardsdatascience*, vol. 53, 2022.

[22] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv*, 2022.

[23] Z. et al., "Improved object detection method utilizing yolov7-tiny for unmanned aerial vehicle photographic imagery," *https://doi.org/10.3390/a16110520*, vol. 16(11), p. 520536, 2023.

[24] Cocodataset. https://cocodataset.org/#home, accessed Oct. 30, 2023.

[25] Python, "What is python?." https://www.python.org/doc/essays/blurb/, accessed Oct. 30, 2023. Python 3.10.8.

[26] Python, "History and license." https://docs.python.org/3/license.html#history-of-the-software, accessed Oct. 30, 2023. History Python.

[27] Pytorch. https://pytorch.org/, accessed Oct. 30, 2023. Pytorch 2.1.

[28] Roboflow. https://universe.roboflow.com/mokhamed-nagy-u69zl/furniture-detection-qiufc, accessed on 23/3/2023.

[29] Roboflow. https://universe.roboflow.com/custombasket/basket_detection, accessed on 2022.

[30] Roboflow. https://universe.roboflow.com/prctiques-uabudg/bathroom-items, accessed on November,2023.

[31] Roboflow. https://universe.roboflow.com/bathroom-accessories/bathroom-rl23o, accessed on January,2023.

[32] Roboflow. https://universe.roboflow.com/laili/broom-detection-g6trp, accessed on 2022.

[33] Roboflow. https://universe.roboflow.com/gudlak/deteksi-sikat-gigi, accessed on 2022.

[34] Roboflow. https://universe.roboflow.com/ssafy-ab11i/hair-dryer, accessed on 2022.

[35] Roboflow. https://universe.roboflow.com/bene/hdsljshvoyhcf, accessed on April,2023.

[36] Roboflow. https://universe.roboflow.com/njau/shuilongtou, accessed on 2022.

[37] Roboflow. https://universe.roboflow.com/datacluster-labs-agryi/stairs-image-dataset-parts-of-house-indoor, accessed on july,2023.

[38] Roboflow. https://universe.roboflow.com/pruebas-0f3uc/toilet-loogr, accessed on june,2023.

[39] Roboflow. https://universe.roboflow.com/object-detection-l7hk4/talov-hairdryer, accessed on augest,2023.

[40] Roboflow. https://public.roboflow.com/, accessed on 2023.

[41] W. K. Yiu. GitHubhttps://github.com/WongKinYiu/yolov7, accessed on 6/7/2022.