VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Simple Camera-based Object Distance Estimation

Jednoduchý odhad vzdálenosti objektu založený na kamerových záznamech

Thach Ngoc Pham

Bachelor Thesis

Supervisor: Hossein Barghi Jond, Ph.D

Ostrava, 2022

VSB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Bachelor Thesis Assignment

Student:	**Ngoc Thach Pham**
Study Programme:	B2647 Information and Communication Technology
Study Branch:	2612R025 Computer Science and Technology
Title:	Simple Camera-based Object Distance Estimation Jednoduchý odhad vzdálenosti objektu založený na kamerových záznamech
The thesis language:	English

Description:

Distance measurement is a key role for an intelligent robot or an agent to understand its workspace. This feature has been embedded in many of Driver Assistance Systems, and mobile robot applications. There are many tools and methods used to determine the distance to objects or obstacles for various operating environments. Some tools send signals to object such as laser or ultrasonic range-finder. However, some tools only receive information about the object position such as measurements based on vision algorithms. The student is expected to prepare a review of the most well-known but yet simple a stationary single-camera based algorithms for distance estimation. The camera here is supposed to be a simple webcam or phone camera. The algorithms can include deep learning-based models. Later on, a PC or mobile-based device app should be developed based on these algorithms. The app receives the real-time video frames (preferably from the camera in a Laptop or phone) and shows an estimation of the preferred object.

The thesis should be organized as follows:
1. Introduction to vision-based object distance estimation and its application in different areas.
2. Describing of the well-known vision (or deep) based algorithms for distance estimation.
3. Implementation of the described algorithms in the form of an app.
4. Discussing the performance of the implemented algorithms.
5. Conclusion.

References:

[1] https://www.youtube.com/watch?v=3rQn_BKWVeI
[2] Z. Sadreddini, T. Çavdar, H. B. Jond, A Distance Measurement Method Using Single Camera for Indoor Environments, 39th International Conference on Telecommunications and Signal Processing (TSP), Vienna, 2016, pp. 462-465.
[3] Jiaxu Zhang, Shaolin Hu, Haoqiang Shi, Deep Learning based Object Distance Measurement Method for Binocular Stereo Vision Blind Area, International Journal of Advanced Computer Science and Applications, Vol. 9, No. 9, pp. 608-613, 2018.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Hossein Barghi Jond, Ph.D.**

Date of issue:	01.09.2021
Date of submission:	08.07.2022

doc. Ing. Petr Gajdoš, Ph.D.	prof. Ing. Jan Platoš, Ph.D.
Head of Department	*Dean*

Abstrakt

V posledních letech došlo k rychlému a efektivnímu nárůstu výzkumu v oblasti počítačového vidění, který bude pokračovat. Část tohoto úspěchu lze přičíst přijetí a přizpůsobení metod strojového učení, zatímco další část lze přičíst vynálezu nových reprezentací a modelů pro specifické problémy počítačového vidění a také vývoji nákladově efektivních řešení. Detekce objektů je jednou z oblastí, která v posledních letech dosáhla významného pokroku. Detekce objektů se používá v řadě aplikací, včetně robotiky, spotřební elektroniky (např. chytrých telefonů), bezpečnosti a dopravy (např. autonomní a asistované řízení). V této práci je detekční úloha první splněnou úlohou, protože umožňuje získat další informace o identifikovaném objektu i o okolní scéně. Jakmile je instance předmětu detekována, je možné získat další informace, například možnost identifikovat objekt a odhadnout jeho vzdálenost. Cílem této studie je podat podrobný a zevrubný výklad o tom, jak najít objekty a zjistit, jak jsou od sebe vzdáleny. Tato práce se zabývá především vytvořením algoritmů pro měření vzdálenosti objektů a extrakci prvků pomocí metody You Only Look Once (YOLO) v kombinaci s přístupem Triangle Similarity a Monodepth2 pro výpočet vzdálenosti pomocí jedné pevné kamery. Cílem této práce je prozkoumat detekční schopnost metody YOLOv4-tiny, která je v současné době jednou z nejrozšířenějších. Navíc je přesnější než ostatní metody detekce a provádí se rychleji. Metoda YOLO překonává všechna námi zkoumaná opatření a zároveň poskytuje vysokou snímkovou frekvenci pro použití v reálném čase. Namísto výběru nejatraktivnější části snímku předpovídá technika YOLO třídy a ohraničující boxy pro celý snímek v jediném běhu algoritmu. Doporučujeme použít kombinaci YOLOv4-tiny a trojúhelníkové podobnosti a velmi známého přístupu nazvaného Monodepth2 objektivu kamery pro odhad vzdálenosti mezi detekovaným předmětem a kamerou. To umožní přesnější měření vzdálenosti. Pomocí přístupu YOLO detekujeme objekt v obraze a extrahujeme z něj jeho polohu a šířku. Tento obraz je také znám jako virtuální obraz. Předměty využité v testech jsou fotografie věcí každodenní potřeby, jako jsou láhve, lidé, tašky a auta,... Porovnáním skutečné a imaginární šířky předmětu bude přístup založený na trojúhelníkové podobnosti schopen určit ohniskovou vzdálenost fotoaparátu a v důsledku toho určit nejlepší vzdálenost mezi ním a předmětem. Na konci procesu se použije přístup lineární regrese k předpovědi chyby ze zjištěné vzdálenosti.

Klíčová slova

Neuronová síť; hluboké učení; detekce objektů; odhadovaná vzdálenost; YOLO; trojúhelníková podobnost; Monodepth2; počítačové vidění; reálný čas;

Abstract

There has been a quick and effective increase in computer vision research in recent years, and this will continue. Part of this success may be attributed to the adoption and adaptation of Machine Learning methods, while other parts can be attributed to the invention of novel representations and models for specific computer vision challenges, as well as the development of cost-effective solutions. Object detection is one area that has made significant strides in recent years. Object detection has been used in a variety of applications, including robotics, consumer electronics (e.g., smart phones), security, and transportation (e.g., autonomous and assisted driving). In this thesis, the detection task is the first job completed since it enables the acquisition of further information about the identified object as well as about the surrounding scene. Once an instance of an item has been detected, it is possible to gain more information, such as the ability to identify an object and estimated its distance. It is the goal of this study to give a detailed and in-depth explanation of how to find objects and figure out how far apart they are. This thesis is primarily concerned with the creation of object distance measurement and feature extraction algorithms using the You Only Look Once (YOLO) method combined with the Triangle Similarity and Monodepth2 approach for calculating distance with a single fixed camera. The purpose of this thesis is to investigate the detection ability of the method, YOLOv4-tiny, which is one of the most common nowadays. Furthermore, it is more accurate than other detection methods and executes more quickly. The YOLO method outperforms all of the measures we looked at while still delivering a high frame rate for real-time use. Instead of picking the most appealing part of an image, the YOLO technique predicts classes and bounding boxes for the entire image in a single algorithm run. We recommend using a combination of the YOLOv4-tiny and the Triangle Similarity and a very well-known approach called Monodepth2 of the lens camera to estimate the distance between the detected item and the camera. This will allow for a more accurate measurement of the distance. Using the YOLO approach, we detect an object in an image and extract its location and width from the image. This is also known as a virtual image. The items utilized in the tests are photographs of everyday things such as bottles, people, bags, and cars,... By comparing the real and imaginary widths of an object, the triangle similarity approach will be able to determine the focal length of a camera and, as a result, determine the best distance between it and the object. At the end of the process, the linear regression approach is used to forecast the error from the observed distance.

Keywords

Neural network; Deep Learning; Object detection; Estimated distance; YOLO; Triangle Similarity; Monodepth2; computer vision; real-time;

Acknowledgement

I have received a great deal of assistance and direction from teachers and friends as I worked on my thesis. To begin, I would want to offer my heartfelt appreciation to my supervisor, Mr. Hossein Barghi Jond Ph.D, for assisting me in completing my thesis. Following that, I would want to offer my gratitude to all of my professors for their lectures, which aided me significantly in completing this thesis. Finally, I would like to express my gratitude to my family and friends, Ms. Hoang Thi Cam Tu, Ms. Truong Thi Thanh Phuong, and Mr. Nguyen Xuan Sinh, for their constant encouragement, support, and assistance in completing my thesis.

Contents

List of symbols and abbreviations

YOLO	–	You Look Only Once
IoU	–	Intersection over Union
R-CNN	–	Region Based Convolutional Neural Networks
NN	–	Neural Network
MLP	–	Multilayer perceptron
CNN	–	Convolutional Neural Network
SPPNet	–	Spatial Pyramid Pooling Network
SSD	–	Single Shot MultiBox Detector
DPM	–	Deformable Part-based Model
CVPR	–	Conference on Computer Vision and Pattern Recognition
RMSE	–	Root Mean Square Error
UI	–	User Interface

List of Figures

List of Tables

Chapter 1

Introduction

We focus our research in this thesis on the processes of detecting objects and calculating distance from a single stationary camera, both of which are widely studied nowadays.

The detection of instances of visual objects of a specific class (such as people, animals, or cars) in digital photographs is an essential computer vision task. We can see that developing a good and optimal object detection and distance estimation technique is critical. A procedure for recognizing items can solve numerous problems in today's highly technological world. Instead of paying a supervisor, we can program a surveillance robot to recognize strangers in a safe area. Or imagine yourself in the future, living in a smart home with an object detection program installed, and you lose an item by accident and don't have time to look for it. Asking a simple question like "Hey, where is my item?" is all it takes. When things are discovered in a fraction of a second. Alternatively, for larger and more serious issues, such as police chasing a criminal's automobile, which sadly vanished on a major road, the police can swiftly pinpoint a car driving down a certain street using object-detecting equipment. In other words, object detection is a computer vision technique that uses automation to locate instances of objects in digital photos or videos. Object detection, in particular, creates bounding boxes around one or more effective targets in a picture or video clip. The object of interest in the picture or video data being analyzed is an effective target. This thesis's object detection algorithm is based on the You Only Look Once (YOLO)[1] algorithm. Nowadays, YOLO is the most widely used algorithm, with the vast majority of academics relying on it for detection tasks. This method has been described in detail in section 2.3.1. Many different types of objects, both stationary and moving, can now be detected with the help of the five versions from 1 to 5, each with numerous applications and enhancements.

Estimating the distance from a single camera is another task in this thesis. We created this project to estimate not only through images but also in real time, which we believe will be persuasive in our work. Nowadays, the distance estimate method is widely investigated and widely used in a wide variety of fields of endeavor. The investigation of the ideal distance estimator allows for more efficient and cost-effective experimental activity. We can estimate the distance between construction work and street construction and find the distance of moving objects from the camera, such as vehicles, trucks, and other moving objects. Distances are determined using the triangle similarity of the lens focal camera approach [2]. Accurate distance measuring is required in the area of autonomous robots and systems to enable an intelligent robot or agent to understand their environment. This feature is implemented in a large number of Driver Assistance Systems [3] and mobile robotic applications [4]. To assess the distance between objects or barriers, a variety of equipment and processes are used in a variety of operational conditions. Some tools, such as laser or ultrasonic rangefinders, transmit signals to the things they are used to locate. A number of other technologies, such as measures based on vision algorithms, only receive information of the location of the object. We will construct a review of the most well-known, but still simple, stationary single-camera-based distance estimation methods that are now available on the market. A simple webcam or phone camera is intended to be used in this situation. Various deep learning-based models can be implemented in the algorithms. The program collects real-time video frames (preferably from a laptop or phone camera) and shows an estimate of the distance of the desired object on the screen.

A particularly noteworthy aspect of my thesis is the integration of object detection and distance estimation for all objects in an image or in real time. This work shows two ways to combine the YOLO method with the Triangle Similarity and Monodepth2 distance estimation techniques. The problem of calculating the depth value (distance relative to the camera) of each pixel when only a single (monocular) RGB image is available is referred to as monocular depth estimation. This difficult task is a crucial requirement for establishing scene knowledge in order to support applications such as autonomous driving, augmented reality, and 3D scene reconstruction. Methods that are considered to be state-of-the-art typically fall into one of two categories: creating a complicated network that is powerful enough to directly regress the depth map or separating the input into bins or windows in order to lessen the computational complexity of the problem. Method Triangular Similarity is one that is utilized frequently and is considered to be fundamental in the distance measurement task. Using the fundamentals of physics, which are covered in greater detail in section 2.4.1, let us investigate what the result of combining the YOLO detector with Triangular Similarity will be. After YOLO has completed the detection operation, the output will display a list of the items that have been detected by the program. The distance between the camera and each item will now be determined by the estimation technique. Finally, the linear regression method is used to predict the error from the measured distances. The implementations are in Python.

This thesis is structured as follows:

- Chapter 1 is the introduction, which introduces the subject of the thesis, a basic overview of the application, the significance of the issue, and the author's technique of doing research.

- Chapter 2 includes an introduction to vision-based object detection and distance estimation and its application in different areas. After discussing neural networks and deep learning for computer vision, we will introduce the YOLO method, more specifically the YOLOv4-tiny algorithm, which we have used for object detection, as well as provide an overview of two object distance estimation methods: Triangular Similarity and Monodepth2.

- Chapter 3 contains an implementation of the described algorithms in the form of an application. We show how to do both object detection and distance estimation with a variety of tests, each with a different distance.
- Chapter 4 discusses the performance of the implemented algorithms. We present the results of our studies in terms of accuracy, and from there, a comparison of the implemented methodologies is carried out here.
- Chapter 5 is about the conclusion, which sums up the whole thesis.

Chapter 2

Vision-based algorithms for distance estimation

Computer vision is a fast-growing subject that has achieved tremendous results, thanks in large part to upgrades, refining, and establishing of new models in Deep Learning training. Object detection is one of the computer vision research directions that has made significant progress; its applications are extremely important to society, and it is used in a wide range of practical applications such as robotics, security (identification, tracking), smart devices (smart phones, tablets), transportation (autonomous vehicles, automated delivery), human-machine interaction, and so on.

Object detection is a well-known problem in the field of computer vision. Most computer and robot vision systems rely heavily on object detection. Many consumer electronics (for example, face detection for auto-focus in smartphones) and assistant driving technologies have advanced significantly in recent years, but we are still a long way from approaching human-level performance, particularly in terms of open-world learning. It's worth noting that object detection isn't widely employed in many areas where it may be quite useful. Object detection systems are becoming increasingly important as mobile robots and autonomous machines in generally become more extensively deployed (e.g., quadcopters, drones, and eventually service robots). This challenge has now been successfully addressed using Deep Learning techniques. R-CNN, Fast R-CNN, and Faster R-CNN are some of the well-known algorithms that might be discussed here. YOLOv1, YOLOv2, YOLOv3, YOLOv4, YOLOv5. R-CNN.

It is very important for an intelligent robot or agent to be able to figure out how far away things are. Driver Assistance Systems (DAS) [3] and mobile robot applications [4] have used this ability to help people. There are a lot of tools and methods that can be used to figure out how far apart objects or barriers are in different situations. A laser range finder or an ultrasonic range finder sends signals to an object. In this section, we will introduce the Triangle Similarity and the Monodepth2 to figure out how far away an object is from the camera.

2.1 Neural Networks

A dog can distinguish between family members and strangers, and a child can distinguish between animals. Things that seem very simple but are extremely difficult to do with a computer. So, where is the difference? The answer lies in the brain, with its large number of interconnected neurons. Then should the computer simulate that model to solve the above problems?

Neural is an adjective of neuron (neuron). Network refers to a graph structure, so an (artificial) neural network (NN) is a computational system inspired by the activity of neurons in the nervous system. A mathematical model that simulates and represents some functions of neurons in the human brain is called a neural network. Neural networks are algorithms that try to find connections in a set of data by copying the way the human brain works. They do this by using a process that looks like how the human brain works.

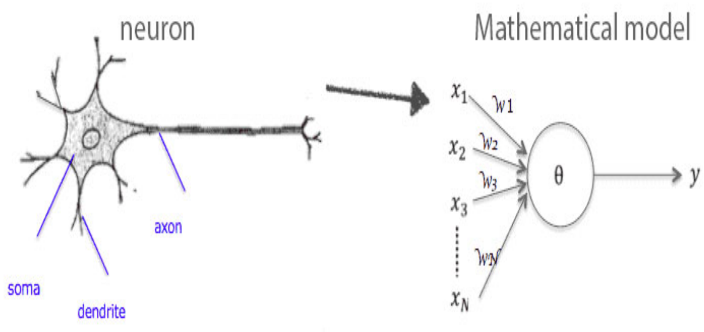


Figure 2.1: Biological model of neuron (on the left) and mathematical model of neuron (on the right) [5].

A neural network is a model capable of learning complex patterns of data through layers of neurons that transform data according to mathematical formulas. The layers of neurons between the input layer and the output layer are called "hidden layers." A neural network can discover and learn relationships between data properties and use these relationships to make predictions. Figure 2.2 represents an example of a simple artificial neural network.

The neural network in Figure 2.3(b) is called the Multi-layer Perceptron (MLP). An MLP network must have at least three layers: input, hidden, and output. These layers are fully interconnected. Each node in each layer connects to all the nodes of the next layer. The term "Deep Learning" is used to refer to a Machine Learning model built with many hidden layers and is also known as a Deep Neural Network.

An artificial neuron (also known as a percepton) is a mathematical transformation function that takes one or more inputs that have been multiplied by scalar weights and adds those weighted inputs together into a single value. This value is then fed into a nonlinear function (called an activation function), and the result of this function is the output of the neuron. In each neuron, there is a

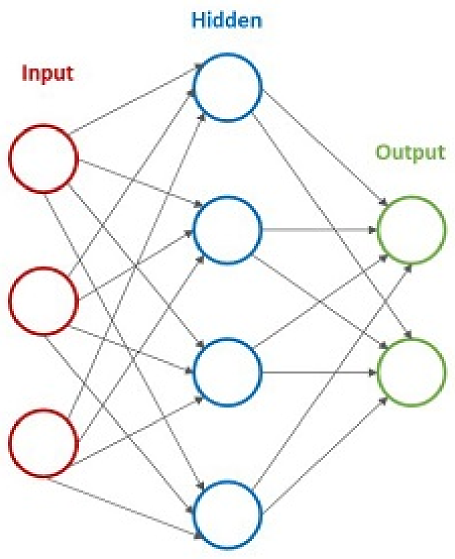
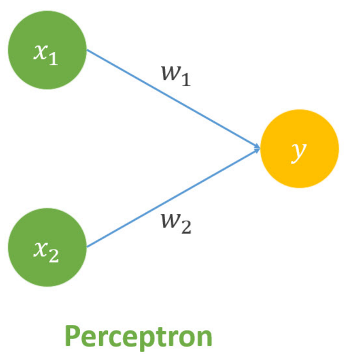
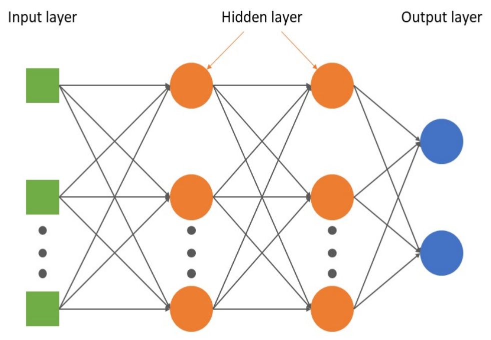


Figure 2.2: A simple artificial neural network.



(a)



(b)

Figure 2.3: Single perceptron (a) and multi perceptron (b).

body (soma) containing the nucleus, input signals through dendrites, and output signals through axons (axons) that connect to other neurons. Simply put, each neuron receives input data through the dendrites and transmits the output data through the axon to the dendrites of other neurons. It receives electrical impulses from other neurons via dendrites. If these electrical impulses are large enough to activate the neuron, the signal travels through the axon to the dendrites of other neurons. So each neuron needs to decide whether to activate that neuron or not.

The input data goes through different layers of artificial neurons that are stacked on top of each other to get the desired result. A neural network has the ability to adapt to any changes in input. So, it can deliver all the results in the best way possible without having to change the criteria for getting them.

Since its inception in 1957 by Frank Rosenblatt, the Perceptron algorithm has been widely adopted. Perceptron learning rules based on the original MCP neuron were devised. Perceptrons are binary classifiers that are trained using supervised learning. Using this method, neurons are able to learn and interpret each of the training components. There are two types of perceptrons: single-layer and multi-layered. Single-layer perceptrons can only learn linearly separable patterns. Two or more layers boost the processing power of multilayer perceptrons or feedforward neural networks. For this reason, it's dubbed the perceptron method: it learns to linearly weight the input signals. It is now possible to compute the $+1$ and -1 differences between the two linearly separable classes [6].

Application of Artificial Neural Network:

Artificial neural networks are employed in a broad range of disciplines, including banking, business planning, trading, business analysis, and product maintenance. Artificial neural networks are also commonly utilized in business for forecasting and problem solving, marketing research, fraud detection, and risk assessment. Based on historical data analysis, artificial neural networks can assess price data and uncover trading possibilities. Other technical analysis methods are unable to discern the nonlinear relationships of the inputs, whereas the artificial neural network can. In few years ago, the best-performing artificial-intelligence systems were speech recognizers on smartphones or Google translators. This is a result of a technique called "Deep Learning."

2.2 Deep Neural Networks and Deep Learning for Computer Vision

First of all, to understand Deep Learning, let's look back at some of the basics of Artificial Intelligence, Machine Learning, and Deep Neural Networks.

- Artificial intelligence refers to programs that are used to solve problems that are simple for humans but difficult for computers to solve.

- Machine Learning algorithms solve problems without specifying what the algorithm does, but instead create an algorithm by statistical method (or later called train-training), computer training.

- Deep Neural Network:

  A Deep Neural Network is a complex neural structure system consisting of many neural network units in which, in addition to input and output layers, there is more than one hidden layer. Each of these classes will perform a separate type of classification and sorting in a process we call "feature hierarchy" and each layer takes on a separate responsibility. The output of this class will be the input of the next class.

  Deep Neural Network is built with the purpose of simulating complex human brain activity and is applied in many different fields, bringing success and amazing results to humans.

- Deep Learning:

  Due to Deep Neural Network technology, a machine can learn very well by itself. It can process and solve data that is not known or structured.

  Deep Learning is a part of Machine Learning, which represents a specific form of this approach, where all types of technology will use all aspects of the field of artificial intelligence to find, classify, and organize information in ways that go beyond the original input and output protocols. We can imagine Deep Learning as the process of simulating our brain activity through mathematical models.

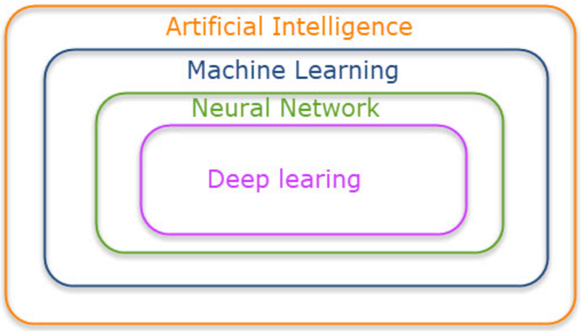


Figure 2.4: Relation between Artificial Intelligence, Machine Learning, Deep Neural Network and Deep Learning.

How the Deep Learning Works

Just like the brain has layers of neurons, neural networks have layers of nodes. Each layer's nodes are linked to those in the layers above and below it. The more levels there are in the network, the more complex it becomes. A single neuron in the human brain receives hundreds of inputs from other neurons. In an artificial neural network, signals travel between nodes and are given weights. The nodes below a node with a higher weight will have a bigger effect on it. In the last layer, the weighted inputs are assembled to produce an output. A significant amount of data and several complex mathematical calculations necessitate powerful hardware for Deep Learning systems. Even with today's most powerful computers, Deep Learning training can take weeks to complete.

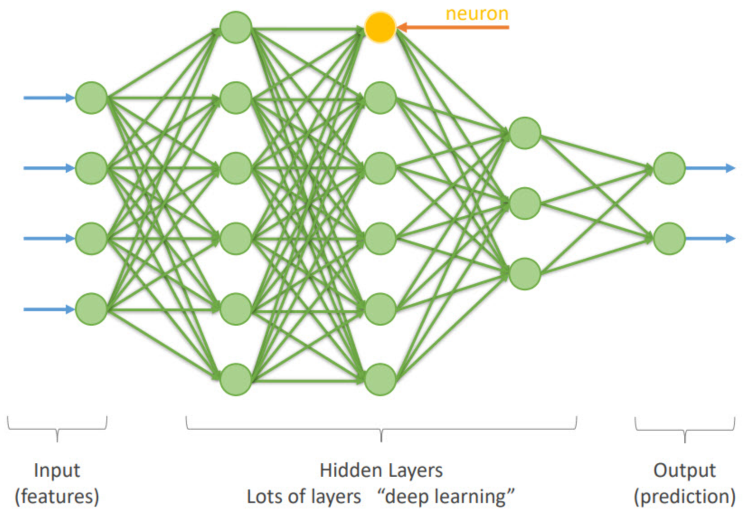


Figure 2.5: In addition to the input and output layers, there is more than one (hidden layer). Each of these classes will perform a separate type of classification and sorting in a process we call "feature hierarchy" and each layer takes on a separate responsibility, the output of this class will be the input of the next class.

As a result, Deep Learning algorithms are fed massive amounts of data. The answers to a series of binary yes-or-no questions can be used by artificial neural networks to classify data using extremely complex mathematical calculations while processing the input. For example, a facial recognition algorithm learns to identify and recognize faces' boundaries and lines, then the faces' more essential features, and lastly, the faces' overall representations, which the system can use to recognize them. The algorithm develops and learns over time, making it more likely to give correct responses. It will get better at recognizing faces in this situation as time goes on.

Application of Deep Neural Network:

Because it can be used and implemented in a wide range of application domains, Deep Learning is often referred to as universal learning. With Deep Learning's many successes in practically every area, one of the most well-known and visible applications of deep learning is in computer vision. Convolution Neural Network (CNN) is the type of DNN most commonly applied in computer vision to analyse images in a similar way as human visual sense does [7]. In 1988, the first Convolution Neural Network (CNN) was released, but it was not so popular at the time. In 1990, Yann LeCun [8] built a Convolutional Neural Network to recognize handwritten digits using backpropagation. This is a watershed moment in computer vision since it establishes the framework for modern computer vision employing Deep Learning. From the year 2000 onwards, particularly By 2011, GPUs had substantially improved in speed, allowing Convolutional Neural Networks to be trained "without" layer-by-layer pre-training. With the increase in processing speed, it became clear that deep learning provided considerable efficiency and speed advantages. One example is AlexNet [9], a Convolutional Neural Network that won multiple international competitions in 2011 and 2012, is one example. To improve the speed and dropout, rectified linear units were utilized.

2.3 Object Detection

An important computer vision task called "object detection" is to look for visual objects of a certain type (like humans, animals, or cars) in digital images. This task is called "detection". In recent years, the rapid development of deep learning techniques has brought new blood into object detection, leading to remarkable breakthroughs and pushing it forward to become a research hot-spot with unprecedented attention. Object detection is now used in a lot of real-world applications, such as self-driving cars, robot vision, video surveillance, and more. Figure 2.6 shows the growing number of "object detection" publications from 1998 to 2018.

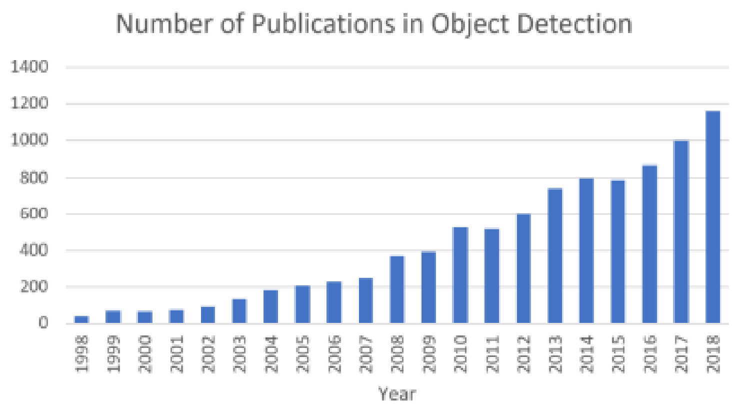


Figure 2.6: The increasing number of publications in object detection from 1998 to 2018. (Data from Google scholar advanced search: allintitle: “object detection” AND “detecting objects” [10].)

Over the past two decades, object detection has been divided into two phases: the "traditional object detection period (before 2014)" and the "Deep Learning-based detection period (after 2014)". The well-known object detection algorithms in the period before 2014 are as follows:

- ***Viola-Jones Detectors* [10]**. Paul Viola and Michael Jones made this object recognition framework in 2001. It lets you see people’s faces in real time. If an image has a lot of sliding windows, it can go through them all to see if any of them have a person’s face in them.

- ***HOG Detector*** [**11**]. Hog was first proposed by N. Dalal and B. Triggs in 2005. It is an improvement to the scale-invariant feature that changes and shapes the contexts of its time, and it was made even better. HOG works with blocks, which are like sliding windows. They are dense pixel grids in which gradients are formed by the magnitude and direction of change in the intensities of pixels in the block, which makes them look like gradients.

- ***Deformable Part-based Model (DPM)*** [**10**]. In 2008, P. Felzenszwalb came up with the idea of the DPM, which is an extension of the HOG detector. Later, R. Girshick made a lot of changes. Car detection can be done in a "divide and conquer" way by looking for its window, body, and wheels, which are all parts of the car.

Even though hand-crafted features have become more and more effective, object detection has hit a dead end since 2010. However, in 2012, the world saw a revival of convolutional neural networks. Deep convolutional networks were able to learn robust and high-level feature representations of an image with these networks. In 2014, people came up with the idea of the Regions with CNN features (RCNN) for object detection. People who use deep learning to find things divide it into two types: "two-stage detection" and "one-stage detection."

- ***RCNN*** [**12**] It starts with the extraction of a set of object proposals (object candidate boxes) by selective search. It then scales each proposal down to a fixed size image and feeds it into a pre-trained CNN model that can find features in the image. There are two more things that happen: linear SVM classifiers are used to predict the presence of an object in each region and to recognize different types of objects.

- ***SPPNet*** [**13**]SPPNet only processes the image once at the convolution layers, while R-CNN processes the image at the convolution layers as many times as there are region proposals. However, there are still some drawbacks: training is still a multi-step process, and SPPNet only fine-tunes its fully connected layers while ignoring all previous layers.

- ***Fast RCNN*** [**14**] R-CNN models only use the proposed regions of an image as the input for a CNN. In a Fast R-CNN model, the whole image is used as the input for a CNN. To make Fast R-CNN run faster than R-CNN, we don't have to feed all the region proposals to the convolutional neural network at the same time each time we run it. When you do this, the convolution process is done only once for each image, and a map of its features is created. It doesn't matter that Fast-RCNN can combine the advantages of R-CNN and SPPNet, because proposal detection is still slow.

- ***Faster RCNN*** [**15**] In 2015, S. Ren et al. came up with a new detector called Faster RCNN. It came right after the Fast RCNN. It is the first deep learning object detector that goes from start to finish and is almost real-time. It doesn't matter that Faster RCNN breaks through the speed bottleneck of Fast RCNN because there is still a lot of computation redundancy at the next detection stage. Later, RFCN and light-headed RCNN have been suggested as ways to improve the system.

- ***Feature Pyramid Networks(FPN)*** **[16]** In 2017, T.-Y. Lin and other people came up with Feature Pyramid Networks. If we look into Faster RCNN, we see that it isn't very good at catching small things in the picture. There are now many detectors that use FPN as a building block.

- ***You Only Look Once (YOLO)*** **[17]** There are a lot of previous object detection algorithms that use regions to find the object in the image. The network doesn't look at the whole picture. Instead, it looks at parts of the image that have a good chance of having the object in them. YOLO is much faster than two-stage detectors at detecting things, but it has a lower level of accuracy when it comes to locating things, especially small things like coins. Later versions of YOLO (YOLO V2, YOLO V3, YOLO V4, and YOLO V5) and the SSD (Single Shot Multibox Detector) have paid more attention to this problem. This is why the SSD has been more popular.

- ***Single Shot MultiBox Detector (SSD)*** **[18]** In 2015, W. Liu et al. proposed SSD. Then, in November 2016, C. Szegedy et al. published SSD: Single Shot MultiBox Detector, which sets new benchmarks for object detection performance and precision. Like YOLO, it is a one-step object detector. SSD's most notable contribution is the development of multi-reference and multi-resolution detection techniques, which considerably increase the detection accuracy of a one-stage detector, notably for some small objects.

- ***RetinaNet*** **[19]** During the training of dense one-stage detectors, it was discovered that there was an extreme class imbalance problem. And it is believed that this is the main reason why one-stage detectors perform worse than two-stage detectors, despite their high speed and simplicity. RetinaNet is the best at what it does, and it beats well-known two-stage detectors like Faster R-CNN because it uses ResNet+FPN as a backbone for feature extraction and two task-specific subnetworks for classification and bounding box regression.

In this thesis, we want to apply YOLO (YOLOv4-tiny) for real-time object detection. When YOLO is compared to other detectors, we notice that it has a higher prediction accuracy, a better Intersection over Union in bounding boxes (when compared to real-time object detectors), and it is faster [20] [21].YOLO is a much quicker algorithm than its competitors, reaching speeds of up to 45 frames per second. While algorithms like Faster RCNN use the Region Proposal Network to detect possible regions of interest and then execute recognition on those regions independently, YOLO makes all of its predictions using a single fully connected layer. As a result, methods that use Region Proposal Networks have to execute numerous iterations for the same image, whereas YOLO only has to do one. The detector may be trained and utilized on a standard GPU, making it generalizable. New features in YOLOv4, especially YOLOv4-tiny, improve the classifier's and detector's accuracy, so they can be used in other research projects.

2.3.1 Detecting object with YOLO

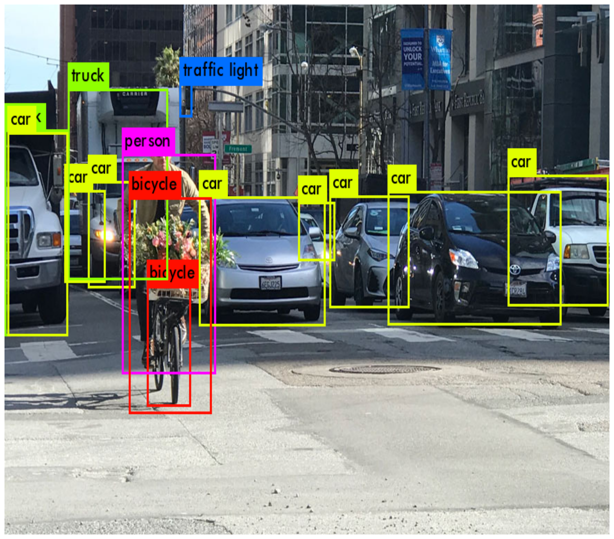


Figure 2.7: Objects detected by YOLO [17].

Object detection has perhaps become one of the most hottest topics in deep learning in recent years due to its wide applicability, easy-to-prepare data, and extremely large application results. New algorithms for object detection, such as YOLO and SSD, have quite fast speeds and high accuracy, so object detection can perform tasks that seem to be in real-time, even faster than humans. Accuracy is not reduced. Models are also becoming lighter so they can work on IoT devices to create smart devices.

In addition, an object detection algorithm can create very diverse applications such as: counting the number of objects, paying at the counter, automatic timekeeping, detecting dangerous objects such as guns, knives, etc., and many other applications. It can be said that any field can be applied to object detection.

Besides, the source of image data is extremely diverse and available because all you need is Google. It is also an advantage to train a model for object detection.

Because of its very high applicability, easy data preparation, and simple model training, in this article, I will introduce to you an object detection state-of-the-art algorithm, which is YOLO.

Many of you ask this question. YOLO sounds a bit like the slogan "You only live once" right? But YOLO in object detection means "You only look once". That is, we only need to look once to be able to detect the object. It's very fast and powerful, is not it?

Indeed, in terms of accuracy, YOLO may not be the best algorithm, but it is the fastest in the class of object detection models. It can achieve almost real-time speed without sacrificing accuracy compared to the top models.

YOLO is an object detection algorithm, so the goal of the model is not only to predict labels for objects like in classification problems but also to determine the location of objects. Thus, YOLO can detect many objects with different labels in an image instead of only classifying a single label in an image.

The reason YOLO can detect so many objects in such an image is that the algorithm has very special mechanisms that we will learn about below.

YOLO is a CNN network model for object detection, recognition, and classification. YOLO is created from the combination of convolutional layers and connected layers. In which case, convolutional layers will extract the features of the image, and full-connected layers will predict the probability and coordinates of the object.

YOLO history and milestones [22]

YOLO has come a long way since it was first proposed by Joseph Redmon in Darknet. Here are a few factors that helped YOLO's first iteration outperform R-CNN and DPM.:

- Frames are processed in real time at a rate of 45 frames per second.
- There are fewer false positives in the background.
- Detection accuracy gets better (although with lower accuracy on localization). The algorithm has changed a lot since it was first released in 2016. Both YOLOv2 and YOLOv3 were written by Joseph Redmon, who also wrote YOLOv3. It started with the third version of YOLO. After that, new authors started putting their own goals into each new version of YOLO.

YOLOv2: This version was released in 2017, and it was given an honorable mention at CVPR 2017 because of its big changes to anchor boxes and better resolution.

YOLOv3: When it came out in 2018, there was a new objective score for bounding box prediction and connections to the backbone network layers. It also worked better on small things because it could run predictions at three different levels of granularity [22].

YOLOv4: In April 2020, the first paper that wasn't written by Joseph Redmon was published. Here, Alexey Bochkovski came up with new ideas like activating the mind, better feature aggregation, and more.

YOLOv5: In his June 2020 release, Glenn Jocher focused on the architecture itself to make further improvements.

2.3.1.1 Network structure of YOLO

YOLO's structure includes base networks and convolution networks that perform feature extraction. The back part is the extra layers applied to detect objects on the feature map of the base network.

YOLO's base network uses mainly convolutional layers and fully connected layers. YOLO architectures are also quite diverse and can be customized into versions for many different input shapes.

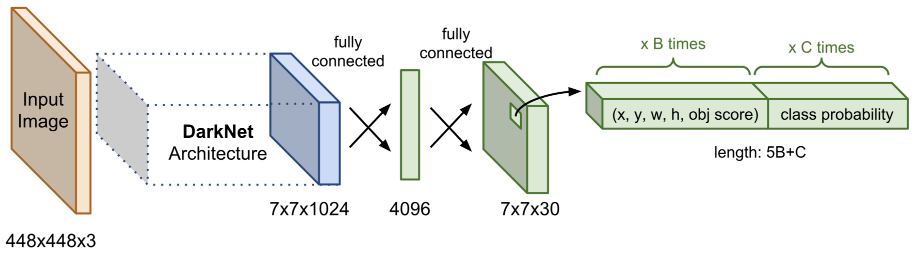


Figure 2.8: YOLO network architecture diagram. The Darknet Architecture component, called the base network, has a feature extraction effect. The output of the base network is a 7× 7× 1024 feature map that will be used as input for extra layers that predict the object's label and bounding box coordinates.

Image classification is the process of figuring out what kind of object is in an image. This is called "object localization." It means figuring out where one or more objects are in an image and drawing a box around them. Object detection does both of these things at the same time. It finds and classifies one or more objects in an image. To achieve these goals, two losses are calculated for the classification and regression units at the top of the feature extraction module, and the model weights are updated with respect to both losses.

In this work, we describe a computer vision algorithm for passively capturing devices. The algorithm detects objects in a scene together with information about the absolute distance from a camera.

For example, in figure 2.9, this way, the headlamp's control unit makes sure that the light-emitting parts aren't too bright so that the phone's camera can get a good picture of the scene in front of it. Algorithms need to know where the object is and how far it is away from them in order to make a good decision, so they need to know these things. As humans, we can tell and figure out how far away things are by using logic, like, "This is a bicycle, and it's small, so it's about 40 meters away." Even if we only use one eye to think this way, we still have some of the ability. The goal of the work is to come up with an extension that will not cost more to run than the original method, but the quality of the distance estimates will be similar to other state-of-the-art systems. This project aims to show how to figure out the absolute distance between two points. An algorithm called YOLOv4-tiny was used for this project.

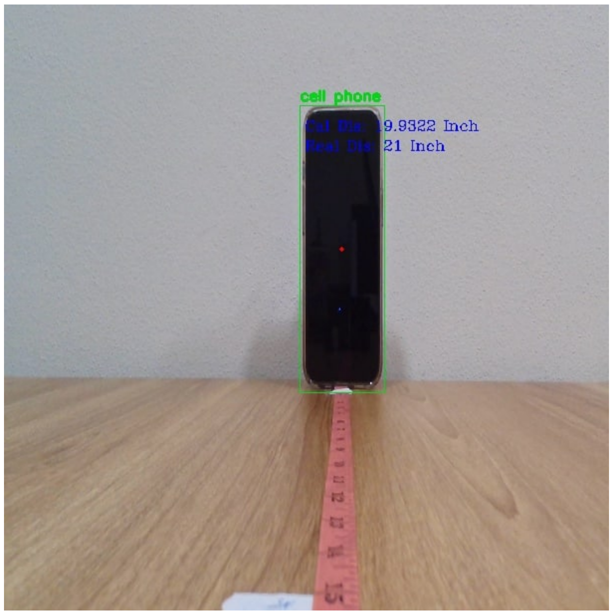


Figure 2.9: An experiment on estimating object distance using the YOLOv4-tiny and Triangle similarity.

2.3.1.2 YOLO Algorithm

The YOLO algorithm performs substantially better on all of the metrics we covered, as well as provides a high frame rate for real-time use. Instead of selecting the most intriguing section of an image, the YOLO method predicts classes and bounding boxes for the entire image in a single run of the algorithm[23].

To understand the YOLO algorithm, we should first know what is being predicted. And then, we want to be able to predict an object's class and the bounding box that specifies its location. Four descriptors can be used to describe each bounding box:

- Center of the box (cx, cy).
- Width (bw).
- Height (bh).
- Value c corresponding to the class of an object.

We also predict a real number pc, which is the probability that an object appears inside the bounding box.

Instead of looking for interesting regions in the input image that could contain an object, YOLO divides the image into SxS cells. Each grid cell predicts only one object. For example, in Figure 2.10 the yellow grid cell below tries to predict the "person" object whose center (the blue dot) falls inside the grid cell [17].

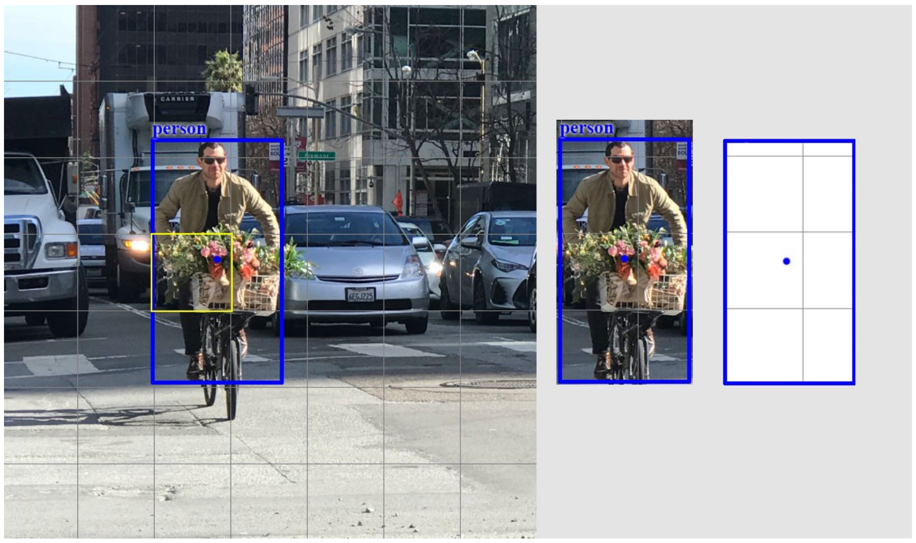


Figure 2.10: YOLO's Image Grid. Each grid cell detects only one object [17].

An object can only be in a cell if its center coordinates are in that cell. Since the center coordinates are always based on the cell, the height and width are always based on the whole picture's size.

YOLO determines the probability that a cell contains a specific class during the first pass of forward propagation. The following is the equation for the same:

$$score_{c,i} = p_c \times c_i \tag{2.1}$$

The class with the greatest probability is chosen and assigned to that grid cell. All of the grid cells in the image go through a similar process.

In Figure 2.11, using the given class probabilities, a possible image may look like this :

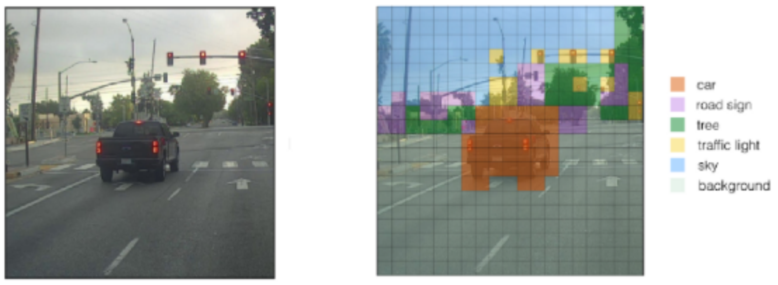


Figure 2.11: This pictures show the before and after of predicting the class probabilities for each grid cell [23].

Non-Max Suppression is the next step after forecasting class probabilities. It aids in the algorithm's efforts to eliminate redundant anchor boxes. As seen in the figure below, anchor boxes were created for each class based on how likely it is to occur.

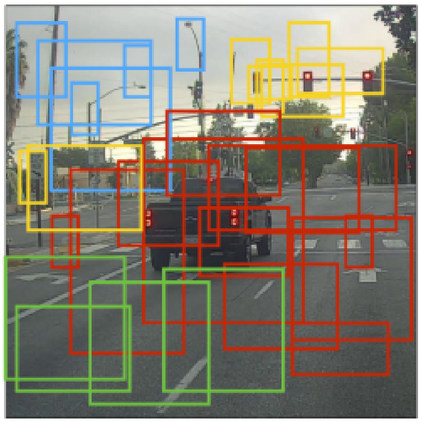


Figure 2.12: Anchor boxes[23].

To solve this problem, do the IoU (Intersection over Union) with the one that has the highest class probability among them. Non-Max Suppression removes the bounding boxes that are very close to each other.

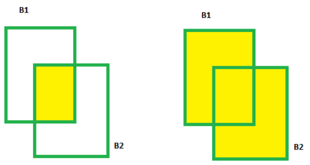


Figure 2.13: An example of intersection and union operations.

From the Figure 2.13, the IoU operation is defined:

$$IoU = \frac{B1 \cap B2}{B1 \cup B2} \tag{2.2}$$

The IoU operation calculates the value of IoU for all bounding boxes that correspond to the one with the highest class probability, then rejects those whose value of IoU exceeds a threshold. It means that those two bounding boxes cover the same item, but one of them has a low probability of being the same, so it is eliminated.

After that, the algorithm finds the bounding box with the next highest class probabilities and repeats the procedure until all of the different bounding boxes are found.

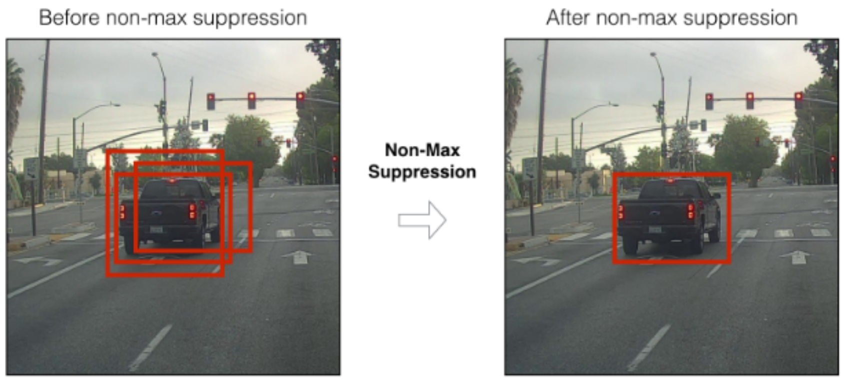


Figure 2.14: Before and After of Non-Max Suppression [23].

After almost all of our work is completed, the algorithm generates the required currency based on the bounding box details for the respective class. The algorithm's overall architecture is shown below.

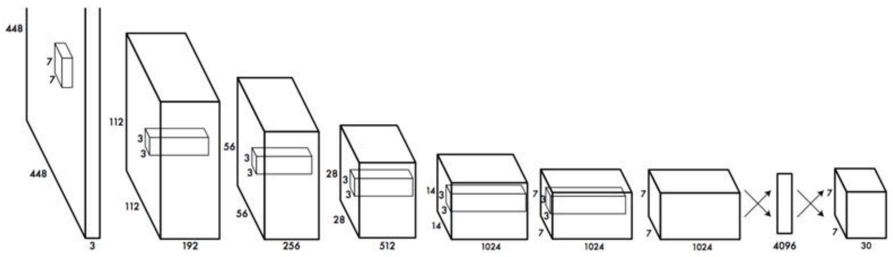


Figure 2.15: The Architecture. In the detecting network, 24 convolutional layers come before two fully linked layers. Preceding layers' feature space is reduced by alternating 1 to 1 convolutional layers. ImageNet classification is done at half resolution (224 x 224 input image) before being doubled for detection purposes [23].

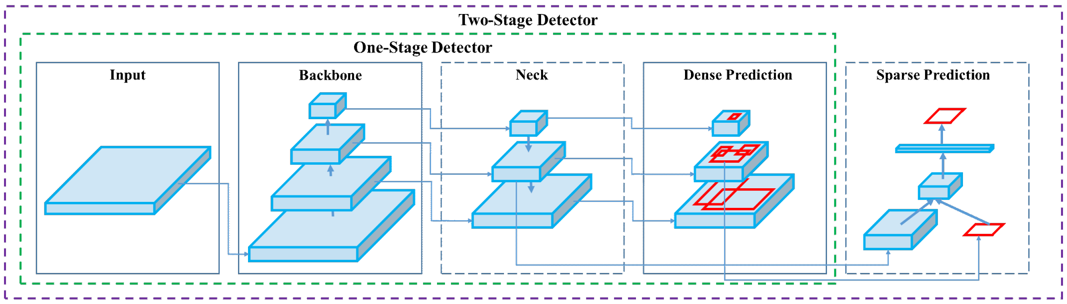
2.3.1.3 YOLOv4-tiny

YOLOv4 is an extension of the YOLOv3 paradigm. YOLOv4-tiny is one of the most common nowadays. Furthermore, it is more accurate than other detection methods and executes more quickly. The YOLO method outperforms all the measures we looked at while still delivering a high frame rate for real-time use. Instead of picking the most appealing part of an image, the YOLO technique predicts classes and bounding boxes for the entire image in a single algorithm run. Overall, the YOLO versions after v4 consist of three main pieces:

- Backbone - A convolutional neural network that aggregates and forms image features at different granularities.
- Neck - A series of layers to mix and combine image features to pass them forward to prediction.

- Head - Consumes features from the neck and takes box and class prediction steps.

Below is sub-modules options used inside that main architecture based on YOLOv4 [24]:



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 2.16: The overview of YOLOv4 Architecture [24].

- Input: Image, Patches, Image Pyramid.
- Backbones: VGG16 [25], ResNet-50 [26], SpineNet [27], EfficientNet-B0/B7 [28], CSPResNeXt50 [29], CSPDarknet53 [29].
- Neck:
  - Additional blocks: SPP [30], ASPP [31], RFB [32], SAM [33].
  - Path-aggregation blocks: FPN [33], PAN [34], NAS-FPN [35], Fully-connected FPN, BiFPN [36], ASFF [37], SFAM [38].
- Heads:
  - Dense Prediction (one-stage):
    - ∗ RPN [39], SSD [40], YOLO [41], RetinaNet [42] (anchor based).
    - ∗ CornerNet [43], CenterNet [26], MatrixNet [44], FCOS [45] (anchor free).
  - Sparse Prediction (two-stage):
    - ∗ Faster R-CNN [39], R-FCN [46], Mask R-CNN [47] (anchor based).
    - ∗ RepPoints [48] (anchor free).

Alexander Bochkovsky, Chien-Yao Wang, and Hong-Yuan Liao designed YOLOv4. The performance is comparable to EfficientDet. AP (Average Precision) and FPS (Frames Per Second) have both increased by 10% and 12% in YOLOv4 over YOLOv3. An extra module for spatial pyramid pooling is included, as well as a PANet path aggregation neck and a YOLOv3 head [49]. The following are some of YOLOv4's new features:

- *Self-adversarial-training (SAT).*
- *Weighted-Residual-Connections (WRC).*
- *Cross-Stage-Partial-connections (CSP).*
- *Cross mini-Batch Normalization (CmBN).*
- *Mish activation.*
- *Mosaic data augmentation.*
- *DropBlock regularization.*
- *Complete Intersection over Union loss (CIoU loss).*

The lightweight YOLO series methods are based on the full YOLO. They are made by taking out the entire network of the YOLO method. YOLOv4-tiny is one of the lightweight YOLO series methods [50].

Instead of the YOLOv4's CSPDarknet53 backbone network, it uses the CSPDarknet53-tiny backbone network. To reduce detection time, feature pyramid networks (FPN) can be used instead of spatial pyramid pooling (SPP) and path aggregation networks (PANet). Besides, instead of three scale predictions, it employs two-scale predictions (26x26 and 13x13). In comparison to the YOLOv3-tiny, the YOLOv4-tiny uses the CSPBlock network to extract features rather than conditional convolution networks, and employs complete intersection over union to choose bounding boxes. Fig 2.17 shows the YOLOv4-tiny network structure.

The YOLOv4-tiny method uses the same method as the YOLOv4 method. It also changes the size of each input image to make sure that they all have the same fixed size. Second, input images are divided into size $S \times S$ grids, with B bounding boxes used to detect objects in each grid. As a result, it will generate $S \times S \times B$ bounding boxes for an input image, and the bounding boxes generated will cover the entire input image. If the center of an object falls within a gird, the object's behavior will be predicted by the gird's bounding boxes.

The confidence score of the bounding box can be obtained as follows:

$$C_i^j = P_{i,j} \times IoU_{pred}^{truth} \tag{2.3}$$

where C is the confidence score of the j^{th} bounding box in the i^{th} grid. $P_{i,j}$ is merely a function of the object. If the object is in the j^{th} box of the i^{th} grid, $P_{i,j} = 1$, otherwise. The IoU_{pred}^{truth}

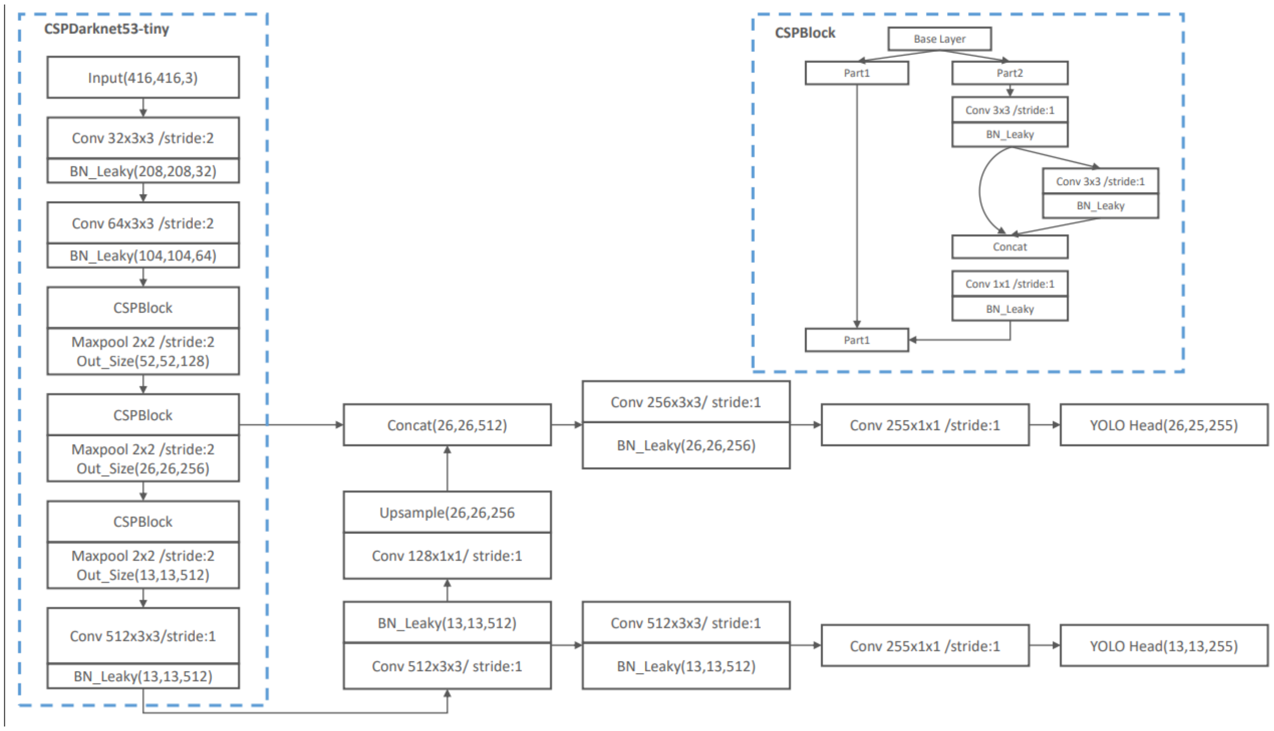


Figure 2.17: Structure of the YOLOv4-tiny network.

represents the intersection over union between the predicted box and ground truth box. The larger the abjectness score, the closer the predicted box is to the ground truth box [21].

YOLOv4-tiny has the same loss function as YOLOv4, which has three parts.

$$loss = loss_{confidence} + loss_{classification} + loss_{localization} \tag{2.4}$$

The confidence loss function is:

$$loss_{confidence} = -\sum_{i=0}^{S^2}\sum_{j=0}^{B} W_{ij}^{obj}[\widehat{C}_i^j log(C_i^j) + (1-\widehat{C}_i^j)log(1-C_i^j)] - \tag{2.5}$$

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}(1-W_{ij}^{obj})[\widehat{C}_i^j log(C_i^j) +$$

$$(1-\widehat{C}_i^j)log(1-C_i^j)] \tag{2.6}$$

where S^2 is the number of grid in input image, B is the number of bounding box in a grid, W_{ij}^{obj} is merely a function of the object. If the j^{th} bounding box of the i^{th} grid is responsible for detecting the current object, W_{ij}^{obj}=1 ,otherwise W_{ij}^{obj}=0. The C_i^j and $\widehat{C}_i^j$ are the confidence score of predicted box and confidence score of truth box, respectively. λ_{noobj} is a weight parameter [21].

The classification loss function is:

$$loss_{confidence} = -\sum_{i=0}^{S^2}\sum_{j=0}^{B} W_{ij}^{obj} \sum_{c=1}^{C}[\widehat{p}_i^j(c)log(p_i^j(c)) - (1-\widehat{p}_i^j(c))log(1-p_i^j(c))] \tag{2.7}$$

where p_i^j and $\widehat{p}_i^j$ are predict probability and truth probability to which the object belongs to c classification in the j^{th} bounding box of the i^{th} grid [21].

The localization loss function is:

$$loss_{localization} = 1 - IoU + \frac{\rho^2}{c^2}(b, b^{gt}) + \frac{16}{\pi^4}\frac{\left(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h}\right)^4}{1 - IoU + \frac{4}{\pi^4}\left(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h}\right)^2} \tag{2.8}$$

where IoU is intersection over union between the boxes that are predicted bounding box and truth bounding box. w^{gt} and h^{gt} are the truth width and height of the bounding box, respectively. w and h are the predicted width and height of the bounding box, respectively. $\rho^2(b, b^{gt})$ denotes the Euclidean distance between the center points of predicted bounding box and truth bounding box. c is the minimum diagonal distance of box that can contain the predicted bounding box and truth bounding box [21].

2.4 Vision-based Object Distance Estimation

Pose estimation and camera tracking are two important parts of robotics applications like localization, positioning tasks, and navigation that use vision-based tools. However, a single-camera-based distance estimation to the object is a very challenging task, and up to now, this topic has not been explored considerably. Two novel object distance estimation methods from a single camera are named DisNet (Distance Network) and YOLO-D. All of them estimate the distance of the objects detected by the object detector.

DisNet makes use of features extracted from the objects' bounding boxes and object class types to estimate the distance to the detected objects. The method can be used with any object detector that outputs an object bounding box and an object class. However, in this work, a state-of-the-art object detector, namely, You Only Look Once (YOLO), has been used.

The second method (YOLO-D) is another approach to estimating the distance to the detected object by modifying the architecture of the YOLO object detection model, a deep learning network. In this method, the modified architecture of YOLO predicts the distance to the detected object besides its primary function of the object bounding box and class prediction.

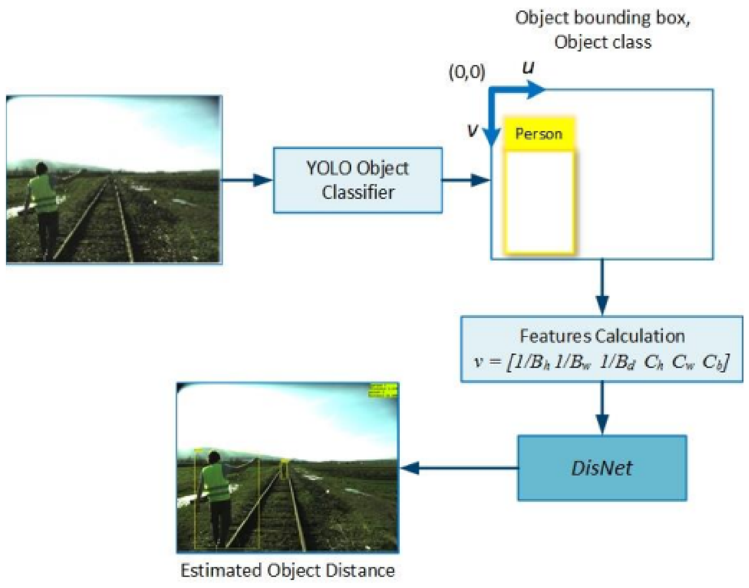


Figure 2.18: An example of the estimation of distances of two persons on the rail tracks is shown using DisNet [51].

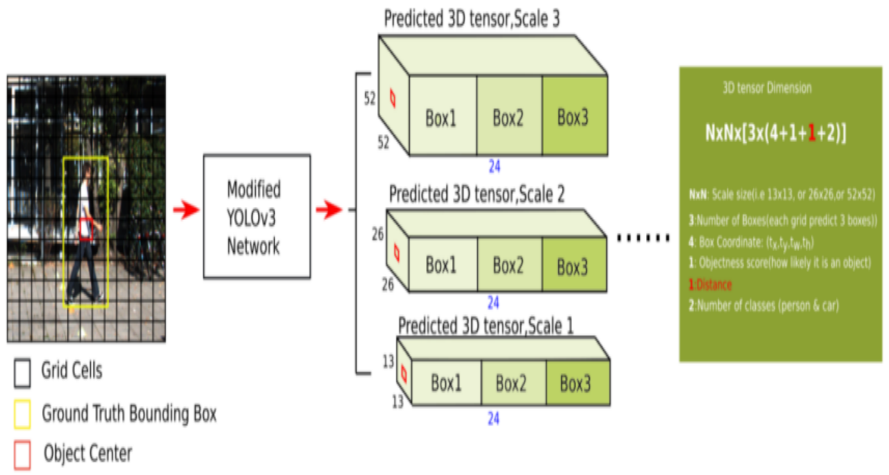

Predicted 3D tensor,Scale 3
Predicted 3D tensor,Scale 2
Predicted 3D tensor,Scale 1
Box1
Box2
Box3
52
26
13
24
Modified YOLOv3 Network
Grid Cells
Ground Truth Bounding Box
Object Center
3D tensor Dimension
NxNx[3x(4+1+1+2)]
NxN: Scale size(i.e 13x13, or 26x26,or 52x52)
3:Number of Boxes(each grid predict 3 boxes))
4: Box Coordinate: (tx,ty,tw,th)
1: Objectness score(how likely it is an object)
1:Distance
2:Number of classes (person & car)

Figure 2.19: The architecture of modified YOLOv3 (YOLO-D) [7].

2.4.1 Triangle Similarity

In this work, we used a combination of YOLOv4-tiny and the Triangle Similarity of the lens camera to estimate distance. It is divided into 2 parts:

- Detect an object and extract its position and width.
- Using the position and width of an object to estimate distance from the Triangle Similarity formula.

In order to determine the distance from our camera to a known object, first we introduce the Triangle Similarity using the basic lens of the camera.

Figure 2.20 shows the experimental setup and explains. We have:

$$\triangle OAB \sim \triangle Oab \tag{2.9}$$

By using the similar triangle method, we prove that:

$$X = \frac{f \times width}{\text{pixels per width}} \tag{2.10}$$

From the above formula (2.10), we can find the distance of the object. However, the focal length is unknown. There are many ways to find the camera's focus. The most common method is to adjust the camera using an image of a chessboard (or similar object) taken by the camera at many different tilt angles in the x, y, and z planes. An explanation of this method is beyond the scope of this article. This method is too technical and cumbersome.

Another easier method is to check the detailed specifications of the camera or the EXIF data of the image taken from the camera. However, these methods do not apply to all cameras.

In this thesis, we use the Triangle Similarity method to find the focal length of a camera. To do this, we need to know some details:

- The real distance of the camera from an object (D).
- The real width of this object (W). Here we use Inch in our calculations.
- The width of the object in the image (P). To detect the object in the photo, we use the Detect Object YOLOv4-tiny function and export the width of the object in pixels.

The formula can be used to calculate focal length [2]:

$$f = \frac{P \times D}{W} \tag{2.11}$$

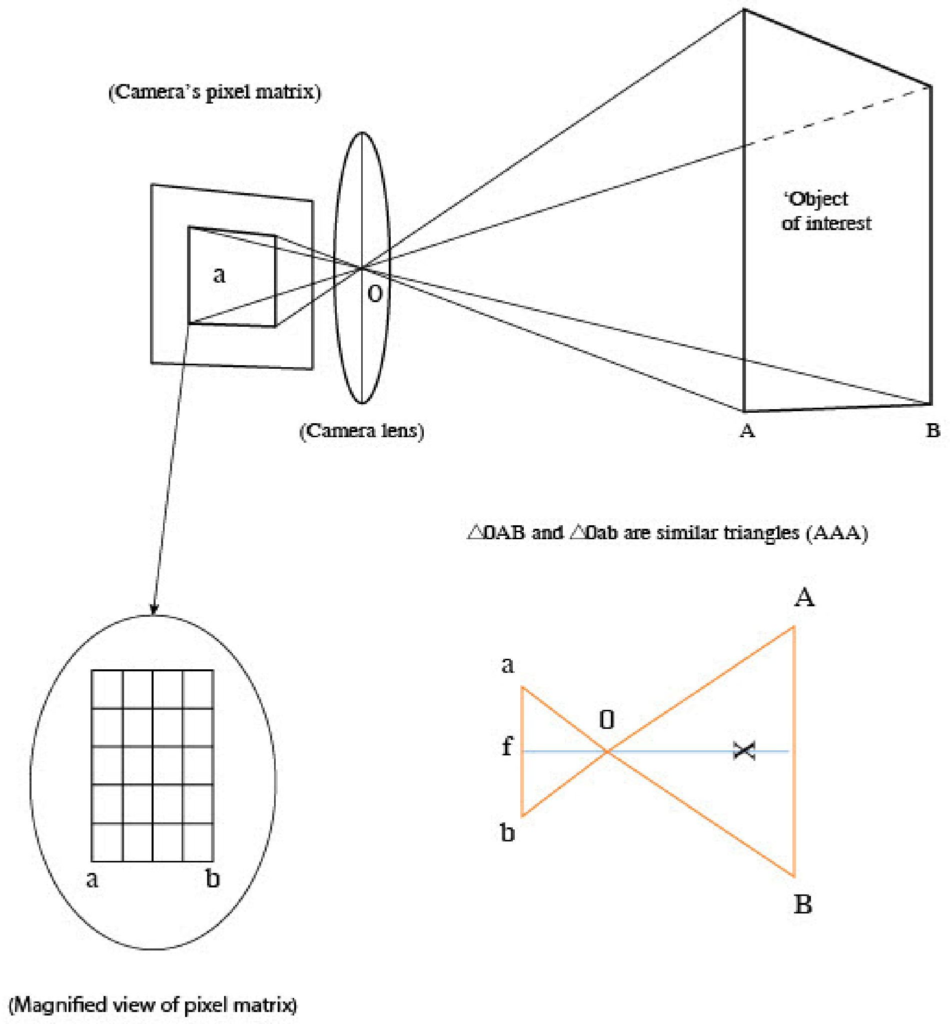

(Camera's pixel matrix)
a
O
(Camera lens)
'Object of interest
A
B
△OAB and △Oab are similar triangles (AAA)
A
a
O
f
X
b
B
a
b
(Magnified view of pixel matrix)

Figure 2.20: Illustration of the Triangle Similarity method [52].

As we continue to move objects both closer and further away from the camera, we can apply the triangle similarity to determine the distance of the object to the camera as [2]:

$$D' = \frac{W \times f}{P} \tag{2.12}$$

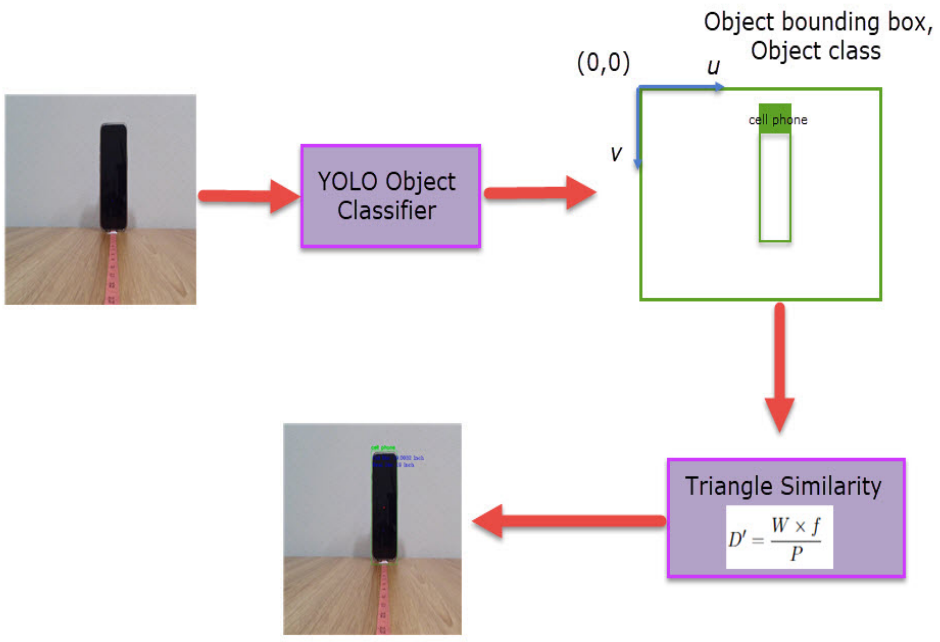


Figure 2.21: An example of the estimation of the distances of cell phone using the YOLOv4-tiny and Triangle Similarity Method.

2.4.2 Monocular Depth Estimation

For the second work, we estimate the distance using the Depth Estimation method. Let's talk about the idea of depth estimation so you can understand what it is.

2.4.2.1 An overview of Monocular Depth Estimation methods

Single-image depth estimation is an old problem in computer vision that can be used in robot navigation, augmented reality, 3D reconstruction, autonomous driving, and other fields.

According to the different mathematical models, Monocular Depth Estimation methods can be divided into traditional Machine Learning-based methods and Deep Learning-based methods.

Deep learning-based methods can be divided into supervised and unsupervised methods. Supervised learning methods require each RGB image to be labeled with its corresponding depth information and are therefore limited by the training set scenario. And depth information tags need to be collected by depth cameras or LIDAR, but the range of depth cameras is limited and LIDAR is expensive, so unsupervised estimation methods are cheaper. Therefore, unsupervised estimation methods without depth tags are the research trend. Monodepth2 is a very effective self-supervised monocular depth method that uses intrinsic geometric relations for supervised learning, and it is implemented via PyTorch, which is very user-friendly. As a result, we select Monodepth2 as the monocular depth estimation model for estimating the distance between the camera and the object. As shown in Figure 2.22, monodepth2 compares with several other monocular depth estimation methods on the KITTI dataset, and it can be seen that monodepth2 produces the clearest depth map.

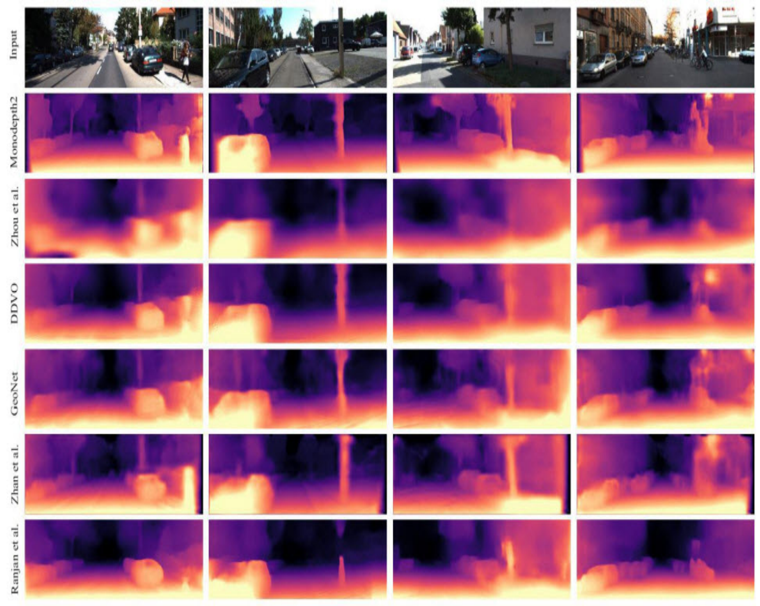


Figure 2.22: Comparison of the depth maps produced by Monodepth2 with other methods on the KITTI dataset. Depth from a single image. The self-supervised model, Monodepth2, produces sharp, high quality depth maps, whether trained with monocular (M), stereo (S), or joint (MS) supervision [53].

2.4.2.2 Monodepth2 model principle

The Monodepth2 method uses a combination of depth estimation and pose estimation networks to predict the depth in a single image frame. It does this by training an architecture based on a self-supervised loss function on a series of moving images. The architecture is made up of two networks, one for predicting depth on monocular images and the other for predicting pose between moving images. This method does not require labeling of the training dataset. Instead, it is trained using successive time frames in the image sequence and the reprojection relationship of the pose.

The training process of the Monodepth2 model is shown in Figure 2.23.

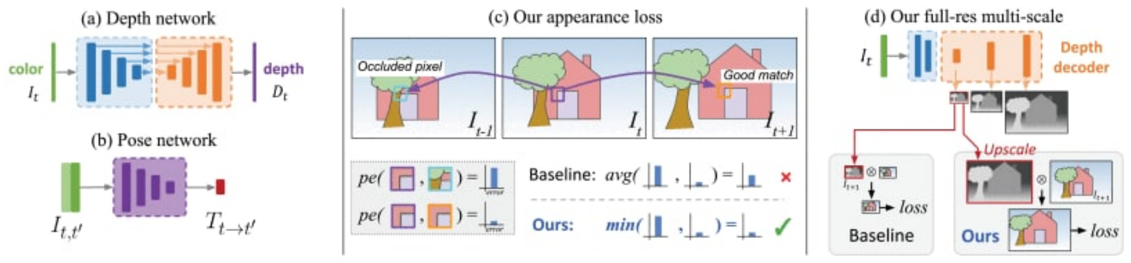


Figure 2.23: Monodepth2 model training process, overview. (a) Depth network: a standard and fully convolutional U-Net is used by Monodepth2 to make predictions about depth. (b) Pose network: a separate network is used to figure out the pose between two frames. (c) Per-pixel minimum reprojection: when correspondences are good, there shouldn't be much reprojection loss. But because of occlusions and disocclusions, pixels from the current time step don't show up in either the previous or next frame. The baseline average loss makes the network try to match pixels that are hidden, but our minimum reprojection loss only tries to match pixels that are visible, which gives sharper results. (d) Full-resolution multi-scale: Monodepth2 uses intermediate layers to upsample and calculate all losses. This keeps texture-copy artifacts to a minimum. [53].

- First, through the standard, fully convolutional U-Net depth network, the depth value of each pixel in the target image is predicted from the color.
- Then, two-color frames are used as input to predict the target image by means of a positional network. Measure a single relative pose or rotation and translation parameter of the target image that has 6 degrees of freedom.
- Finally, the obtained pose matrix is projected into the camera with a fixed internal reference matrix K to obtain the reconstructed target image, calculate the reprojection error loss function for each pixel value, and minimize this loss function to reduce the difference between the target image and the reconstructed target image.

The reprojection error is calculated as follows [53]:

$$L_p = \sum pe(I_t, I_{t'\to t}) \tag{2.13}$$

$$I_{t'\to t} = I_{t'} \langle proj(D_t, T_{t\to t'}, K)\rangle \tag{2.14}$$

$$pe(I_a, I_b) = \frac{\alpha}{2}(1 - SSIM(I_a, I_b)) + (1-\alpha)\left\|I_a - I_b\right\|_1 \tag{2.15}$$

Where pe is the reprojection error, D_t is the projected depth map, $proj()$ is the 2D coordinates of the projected image, $T_{t\to t'}$ is the transformation mapping, K is the internal parameter of the camera, $\langle\rangle$ is a bilinear sampling operator, I_t is the target image and $I_{t'}$ is an adjacent image [53].

2.4.2.3 YOLOv4-tiny and Monodepth2 combination for estimating distance

By using two parallel deep networks YOLOv4-tiny and Monodepth2: one for object detection and the other for depth estimation. The predicted depth is extracted from Monodepth2. In turn with YOLOv4-tiny, the objects inside the image is localized and classified. Furthermore, the localization of each object defined by bounded boxes is detected on the estimated depth image. Finally, the relevant distance of an object is calculated by the median estimated distance of all pixels inside the defined bounded box.

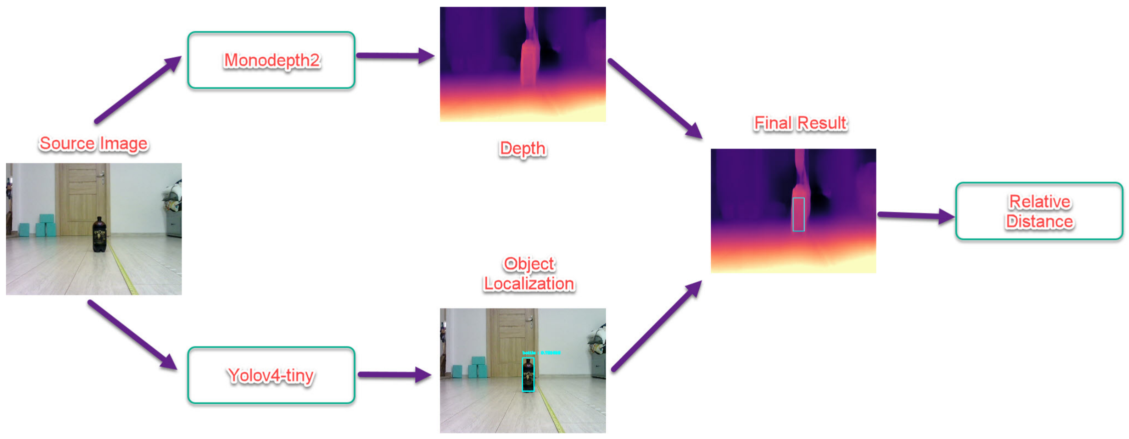


Figure 2.24: An illustration of the overall framework.

Chapter 3

Implementation of the described algorithms in the form of an application

Firstly, We use the Python programming language and build the code structure as follows:

```
.
    Database
        bottle.jpg
        phone.jpg
    Models
        coco.names
        yolov4-tiny.cfg
        yolov4-tiny.weights
    mono_models
        models
            depth.pth
            encoder.pth
    networks
        depth_decoder.py
        resnet_encoder.py
    object_detection.py
    initWindow.py
    main.py
    utils.py
    layers.py
    options.py
    requirements.txt
```

Listing 3.1: Structure Folder For My Application.

- The file *main.py* is the main file to run the app.
- The file *initWindow.py* is the file create the UI.

- The *utils.py* file contains functions using for Triangle Similarity method and Monodepth2 method.
- The folder *Models* contains the pre−trained model file of YOLO.
- The folder *mono_models* contains the pre−trained model file of Monodepth2.
- The folder *networks* contains functions used to decode and encode Monodepth2 network.
- The folder *Database* contains the images used for calculating the focal length of the camera for each object.
- The file *options.py* contains the setting of Monodepth2.
- The file *layers.py* contains functions method for Monodepth2 network.
- A function for detecting objects is included in the file *object_detection.py*.
- The file *requirements.txt* includes the library needed for this program. To make sure the machine has all the libraries, you need to run the command to install them before running the main program. Using this command:

```
        pip install -r requirements.txt

```

- To run this process, use the command:

```
        python main.py

```

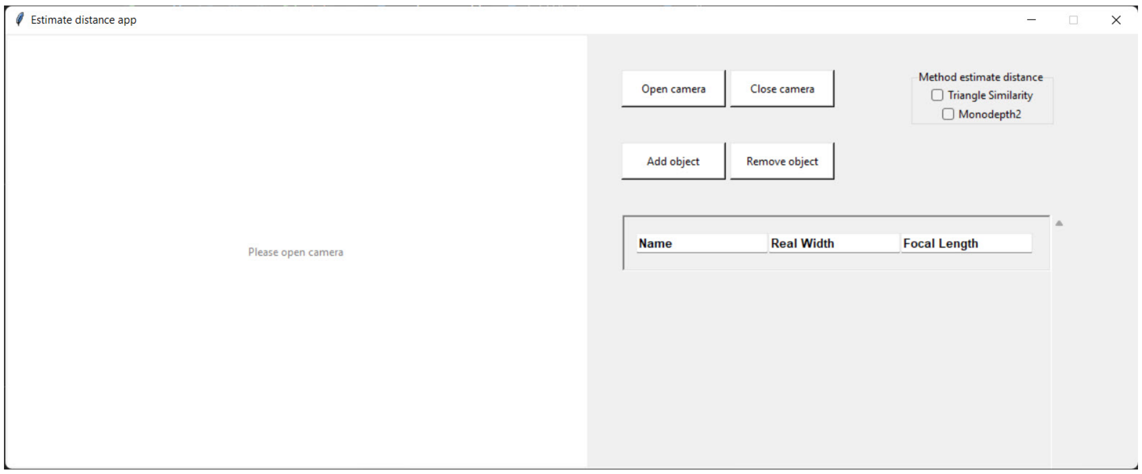


Figure 3.1: An application interface.

3.1 Object detection and estimated distance algorithms

3.1.1 Object detection

In order to detect objects. First we need to load the model files for Yolov4-tiny. This code follows the file *object_detection.py*

```
        # load model
        # (might need to change to absolute path rather than relative path)
        weights_path = "Models/yolov4-tiny.weights"
        config_path = "Models/yolov4-tiny.cfg"
        yoloNet = cv2.dnn.readNetFromDarknet(config_path,weights_path)
```

Listing 3.2: Load model files.

The model we use here is stored in 2 files. Information about the structure of the neural network (config) is stored in 1 file (.cfg) and information about the connection level between neurons (weights) is stored in 1 file (.weights). Here we use a model based on Darknet, so the *readNetFromDarknet* function is used.

If you want to use GPU instead of CPU, you can change your code to enable GPU detection:

```
        yoloNet.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
        yoloNet.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA_FP16)
```

The next step is to apply the model to the image:

```
1       # run model
2       model = cv2.dnn_DetectionModel(yoloNet)
3       model.setInputParams(size=(416, 416), scale=1/255, swapRB=True)
```

Listing 3.3: Run model.

Line 2 in Listing 3.3 is to create a model using the cv2.dnn_DetectionModel function.

Line 3 in Listing 3.3 sets parameters related to image preprocessing, where:

- *scale*: Pixels' normalization scale
- *size*: The resolution of the input image of the model
- *swapRB*: This parameter decides whether to switch between red and green channels because openCV stores images in the BGR order in which the model is trained on RGB images, so the parameter must be set to True in this method.

The next step is to create a detect function:

```
        def object_detector(image, classIN):
            classes, scores, boxes = model.detect(image, CONFIDENCE_THRESHOLD, NMS_THRESHOLD)
            # creating empty list to add objects data
```

```
4           data_list =[]
5           for (classID, score, box) in zip(classes, scores, boxes):
6               # define color of each, object based on its class id
7               color= COLORS[int(classID) % len(COLORS)]
8 
9               label = "%s : %f" % (CLASSES[classID], score)
10              # getting the data
11              # 1: class name  2: object width in pixels, 3: position where have to draw text(distance), 4: bouding box information
12              if classID in classIN:
13                  data_list.append([CLASSES[classID], box[2], (box[0], box[1]-2), box])
14                  #draw bounding box
15                  draw_bounding_box_with_label(image, label, box, color)
16 
17          return data_list
```

Listing 3.4: Object detection function.

Lines 2 (Listing 3.4): Running the object detection model

- *CONFIDENCE_THRESHOLD*: An object with a higher probability score than this threshold will be retained. In this program, *CONFIDENCE_THRESHOLD* is set at 0.3.

- *NMS_THRESHOLD*: Threshold for the Non-Max Suppression (NMS) algorithm used in YOLO. This algorithm filters out the noise contours and preserves the best path for each object. In this program, *NMS_THRESHOLD* is set at 0.5.

Line 5 (Listing 3.4): The output of the model consists of three components:

- *classID*: It contains the id of the found object.

- *score*: The certainty of the model corresponds to the above objects.

- *boxes*: It contains the bounding box coordinates.

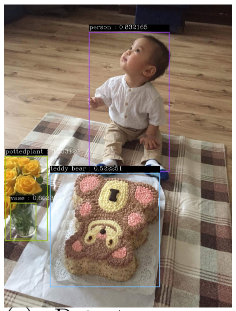
Line 13 (Listing 3.4): Append the object data to the list. It contains the class name, the object width in pixels, the position where we have to draw the text distance and the bounding box information for the depth method. After that, return it for calculation of distance.

Line 14 (Listing 3.4):*draw_bounding_box_with_label* is a function that draws a bounding box along with the corresponding object name.

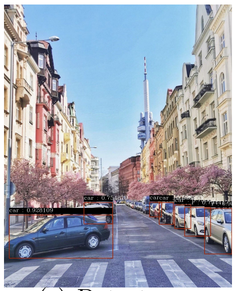
In Fig 3.2. We use the YOLOv4-tiny function to detect some objects in six images. For each different type of image, such as color, position, number of objects in the image,... the function to detect objects can detect the main object in the image with high accuracy.

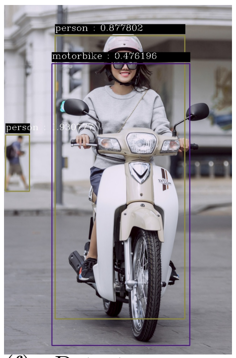
(a) Detect bicycle and person [54].

(b) Detect person, dinner table and bowl.


(c) Detect person, vase, teddy bear and potted plant.

(d) Detect person, sofa and cell phone.


(e) Detect cars



(f) Detect person and motorbike [55].

Figure 3.2: Six experiments in object detection by real images.

3.1.2 Triangle Similarity for distance estimation

By using the algorithm to estimate the distance we mentioned above. We have implemented the two functions in the *util.py* file. One function is to determine the focal length of the camera. Another function is for estimating the object's distance from the camera.

```
        #calculate focal length of camera
        def focal_length_finder (real_distance, real_width, width_in_frame):
            focal_length = (width_in_frame * real_distance) / real_width
            return focal_length

```

```
    # distance finder function
    def distance_finder(focal_length, real_object_width, width_in_frame):
        distance = (real_object_width * focal_length) / width_in_frame
        return distance
```
Listing 3.5: Function to determine focal length and estimate distance.

In function *focal_length_finder*:

- *real_distance* is the real distance of an input object from the database. Here is an inch
- *real_width* is the real width of an input object from the database. Here is an inch
- *width_in_frame* is the width of the input object in the image. Get it from the detect object function. Here is a pixel

A note for using this function is that the real distance and real width of the input object must be in the same units and not scaled by a factor.

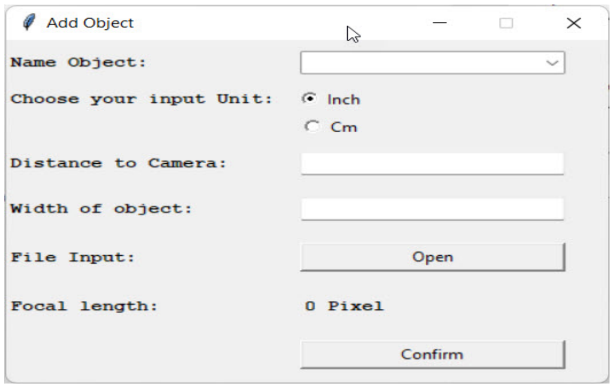
In function *distance_finder*:

- The focal length of the camera is represented by the variable *focal_length*.

Before using these functions. We need to prepare the input data. In this paper, we will prepare the image of two objects. It is a cell phone and a bottle. So we need to take two pictures of these objects with the define distance of the object from the camera and measure the width of these two things.

Here, we take two photos of a cell phone and a bottle and put them all in the *Database* folder like Listing 3.1

After that, we select the type of object, select the input unit, put the real distance and real width of each object, and upload the file image prepared in the previous part into the input interface. The result of focal length will be shown after the input file is uploaded and confirmed by clicking the button to add the object to the system. The UI input object is shown in Figure 3.3 :


Add Object

Name Object:

Choose your input Unit: (● Inch) ○ Cm

Distance to Camera:

Width of object:

File Input: Open

Focal length: 0 Pixel

Confirm

Figure 3.3: The UI input object.

3.1.3 Monodepth2 for distance estimation

In order to use the following algorithm method for measuring distance, all that is required of us is to enter the type of object into the panel that is seen in Figure 3.3. We will break the source code into some parts to describe them.

The first part is preparing the model. In this paper, we use the *mono* pre-train model with size 640×192 and put all of them into the folder models of mono_models.

The next part is to create a function using monodepth2 to estimate distance.

```python

def get_depth(frame, d):
    with torch.no_grad():
        input_color = frame

        input_color = cv2.resize(input_color, (640, 192))/255
        input_color = np.transpose(input_color, (1, 0, 2))

        input_color = input_color.astype(np.float32)

        input_color = np.reshape(input_color,(input_color.shape[0],input_color.
    shape[1],input_color.shape[2],1))
        input_color = np.transpose(input_color)

        input_color = torch.from_numpy(input_color)

        #Predict the depth map
        #opt.post_process = 1
        if opt.post_process:
            # Post-processed results require each image to have two forward passes
            input_color = torch.cat((input_color, torch.flip(input_color, [3])), 0)

        output = depth_decoder_(encoder(input_color))

        pred_disp, _ = disp_to_depth(output[("disp", 0)], opt.min_depth, opt.
    max_depth)

        pred_disp = pred_disp.cpu()[:, 0].detach().numpy()

        if opt.post_process:
            N = pred_disp.shape[0] // 2
            pred_disp = batch_post_process_disparity(pred_disp[:N], pred_disp[N:,
    :, ::-1])

        #Evaluating

        pred_depth = 1 / pred_disp
```

```
35
36         #Multiply the depth map with scale factor
37         pred_depth *= opt.pred_depth_scale_factor
38
39         pred_depth[pred_depth < MIN_DEPTH] = MIN_DEPTH
40         pred_depth[pred_depth > MAX_DEPTH] = MAX_DEPTH
41
42         pred_depth = np.reshape(pred_depth,(192,640))
43
44
45         pred_depth = cv2.resize(pred_depth,(frame.shape[1],frame.shape[0]))
46
47         #Get the depth map inside the bounding box
48         dist_matrix = pred_depth[d[1]:d[1]+d[3], d[0]:d[0]+d[2]]
49         #calculate the mean of all pixels inside the bounding box
50         dist = np.mean(dist_matrix)
51
52
53     return dist
```

Listing 3.6: Depth estimate distance function.

After the frame of the camera and information about the box of the detected object are input through parameters, before the frame is sent to the monodepth2 model to predict the depth map, the function process resizes, transposes, reshapes, and converts it to a torch tensor type (line 6 to 14 in listing 3.6). Then the depth map is evaluated and multiplied with the scale factor value. The default value of the scale factor used in this paper is 5.4. The next step is to extract the depth value from within the bounce box and average it to get the estimated distance (line 48 and 50 in listing 3.6).

3.2 Experiments

3.2.1 Object estimation distance using the Triangle Similarity method

For experiments in object distance estimation, we use the bottle (3.4). Its width is 9.87cm, and we put the bottle away from the camera at 150cm to make a photo for calculating focal length.

We put the phone away from the camera, starting at 0.5 meters and then gradually increasing by 0.1 meters until the distance reached 3 meters. For each distance, we record 300 frames to calculate the distance. Each frame will output the estimated distance and error by calculating the absolute value of the difference between the actual distance and the estimated distance.

$$error = |Real\_distance - Estimate\_distance| \tag{3.1}$$

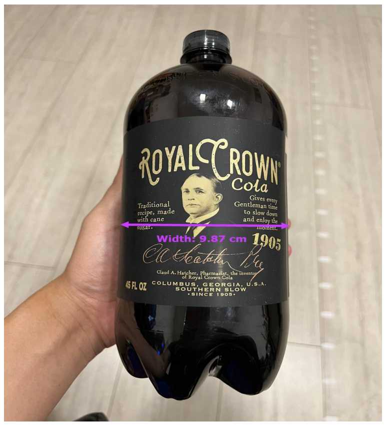


Figure 3.4: The bottle has a width of 9.87cm.

After 300 times record the distance and the error of each distance. We will find the mean error of 300 records to find the best estimated distance.

By following all the steps, we have obtained the results shown in 3.1.

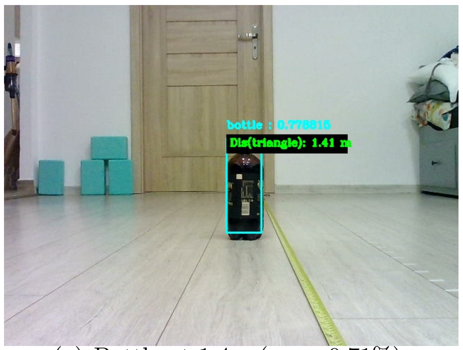
Table 3.1: Result from 26 different distances using Triangle Similarity.

Real	**Estimate**	**Error**	**Real**	**Estimate**	**Error**	**Real**	**Estimate**	**Error**
0.5	0.49648	0.00352	1.4	1.41	0.01	2.3	2.27419	0.02581
0.6	0.57317	0.02683	1.5	1.5	0.0	2.4	2.43103	0.03103
0.7	0.64091	0.05909	1.6	1.60227	0.00227	2.5	2.51786	0.01786
0.8	0.71212	0.08788	1.7	1.71951	0.01951	2.6	2.61111	0.01111
0.9	0.85976	0.04024	1.8	1.80769	0.00769	2.7	2.71154	0.01154
1.0	0.99296	0.00704	1.9	1.90541	0.00541	2.8	2.82	0.02
1.1	1.08462	0.01538	2.0	1.95833	0.04167	2.9	2.9375	0.0375
1.2	1.19492	0.00508	2.1	2.07353	0.02647	3.0	2.9375	0.0625
1.3	1.30556	0.00556	2.2	2.20312	0.00312			

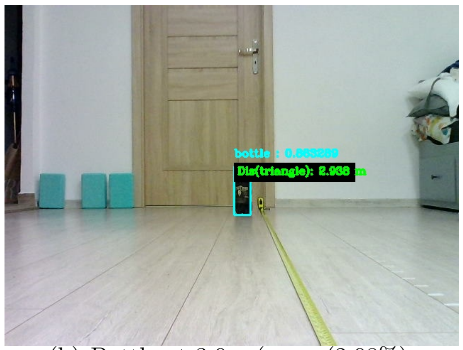
3.2.2 Object estimation distance using the Monodepth2 method

For the experiments in this section, we use the bottle in the above section and use the Monodepth2 to estimate distance.

We put both objects away from the camera, starting at 1 meter and then gradually increasing by 0.5 meter until the distance reached 3 meters. We also do all the steps in section 3.2.1 for each object. Finally, we have the obtained results shown in Table 3.2.



(a) Bottle at 1.4m (error 0.71%).

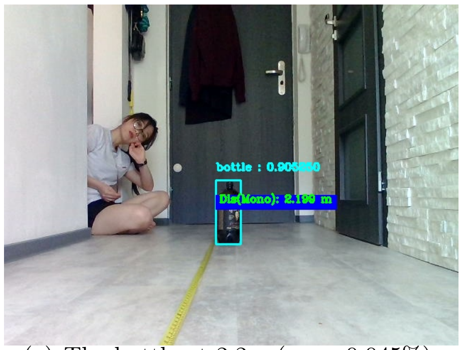


(b) Bottle at 3.0m (error (2.08%).

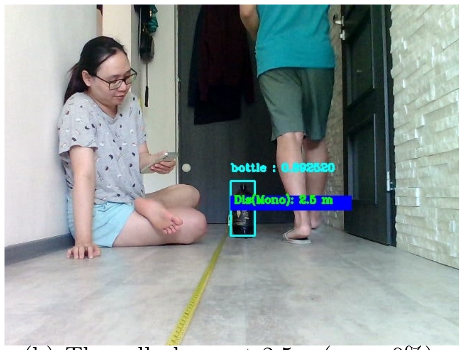
Figure 3.5: We have the results of the distance estimation of the bottle at 1.4m (a) and 3m.(b) in real time.

Table 3.2: Result from 26 different distances using Monodepth2.

Real	Estimate	Error	Real	Estimate	Error	Real	Estimate	Error
0.5	0.82464	0.32464	1.4	1.4574	0.0574	2.3	2.29716	0.00284
0.6	0.90727	0.30727	1.5	1.66969	0.16969	2.4	2.39922	0.00078
0.7	1.13022	0.43022	1.6	1.68637	0.08637	2.5	2.50095	0.00095
0.8	1.2122	0.4122	1.7	1.79789	0.09789	2.6	2.59657	0.00343
0.9	1.05791	0.15791	1.8	1.90424	0.10424	2.7	2.69657	0.00343
1.0	0.9596	0.0404	1.9	2.07904	0.17904	2.8	2.79336	0.00664
1.1	1.11033	0.01033	2.0	2.04307	0.04307	2.9	2.90213	0.00213
1.2	1.35265	0.15265	2.1	2.13781	0.03781	3.0	2.93608	0.06392
1.3	1.38166	0.08166	2.2	2.19957	0.00043			



(a) The bottle at 2.2m (error 0.045%).



(b) The cell phone at 2.5m (error 0%).

Figure 3.6: Results of estimating the distance of a bottle in real time using the Monodepth2 method.

Chapter 4

Discussing the performance of the implemented algorithms

The estimated distance was already shown in real time in the preceding section. Fig 4.1 shows the fit between estimated and true distances, ranging from $0.5m$ to $3m$ from data in two tables 3.1 and 3.2.

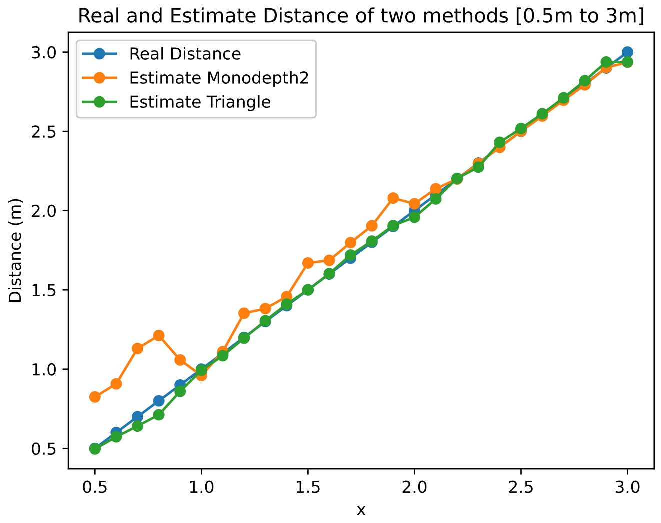


Figure 4.1: Estimate distance in 0.5m to 3m.

Estimated Monodepth2 method is denoted by the orange line in this diagram. The estimated distance at close range using this method, which can range from 0.5 meters to 2 meters, is not very accurate and has a significant margin of error in comparison to the true distance (blue line). When used from a distance of at least 2 meters, technique a has provided estimates that are quite accurate, coming very close to matching the actual distance. The Estimated Triangle method, which is represented by the green line, yields pretty spectacular results. Almost exactly, the anticipated distance corresponds to the real distance; there are some inaccuracies, but they are not very significant. To provide further clarity, we make use of a regression model to figure out the precise error associated with each strategy.

Linear Regression Model

To assess the suggested method's effectiveness, we utilize basic linear regression to determine the experimental error. Linear regression is one of statistics and Machine Learning's most well-known and well-understood algorithms, see more details at [56]. Using a linear regression model, we can anticipate how one variable will change in response to the value of another. The dependent variable is the variable that you want to forecast. As the name suggests, this is the variable you will use to make a prediction about the other variable. This type of linear regression equation serves as the basis for the model's construction: $y = ax + b$. A linear equation including one or more independent variables that best predict the value of the dependent variable is used in this type of analysis to estimate its coefficients, which is a form of analysis known as coefficient estimation. When using linear regression, a straight line or surface is fitted into the data, reducing the differences between projected and actual output values. There are simple linear regression calculators that employ the "*least squares*" method, see more details at [57] to find the best-fit line for a set of paired data, and there are more complex linear regression calculators. After that, you calculate the value of x (the dependent variable) based on the value of y. (independent variable).

The root mean squared error ($RMSE$) is the most commonly used metric for evaluating the effectiveness of a linear regression model. The primary idea is to compare the predicted values ($\hat{y}$) to the actual values (y) observed to see how inaccurate the model's predictions are. In other words, "poor" $RMSE$ is high, and "good" $RMSE$ is low. $RMSE$ was defined by:

$$RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2} \tag{4.1}$$

Some discussion from object distance estimate experiments

- Using the data in Table 3.1 from estimated Triangle method, a linear regression model from 26 pairs of variables (x, y of the following form is generated, the line for this linear regression see Figure 4.2:

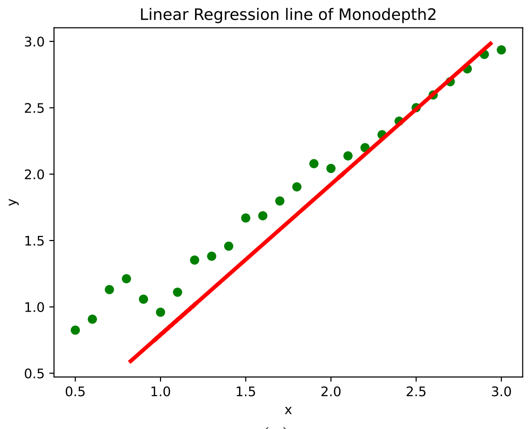
$$y = 0.98297007 \cdot x + 0.0380786$$

  or [estimate distance] = 0.98297007 $\cdot$ [real distance] +0.0380786

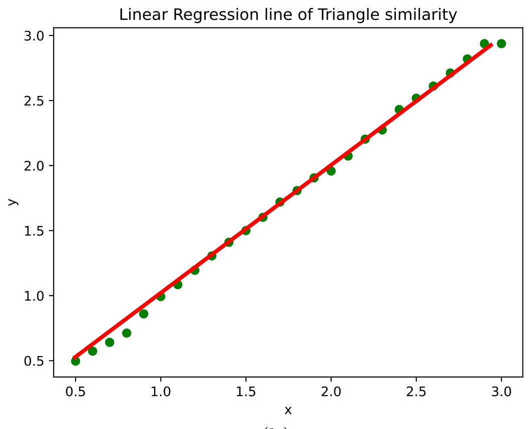
- Using the data in Table 3.1 from estimated Monodepth2 method, a linear regression model from 26 pairs of variables (x, y of the following form is generated, the line for this linear regression see Figure 4.2:

$$y = 1.13283233 \cdot x - 0.34284673$$

or [estimate distance] = 1.13283233 · [real distance] −0.34284673

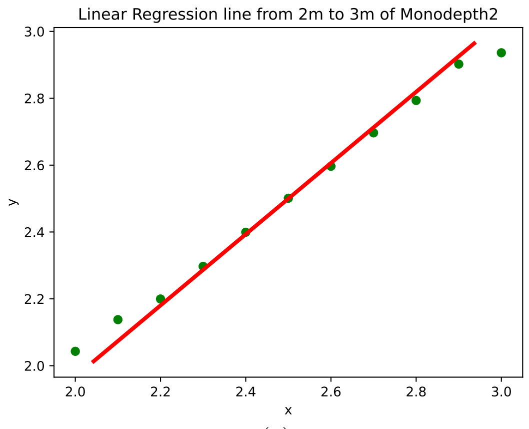


(a)

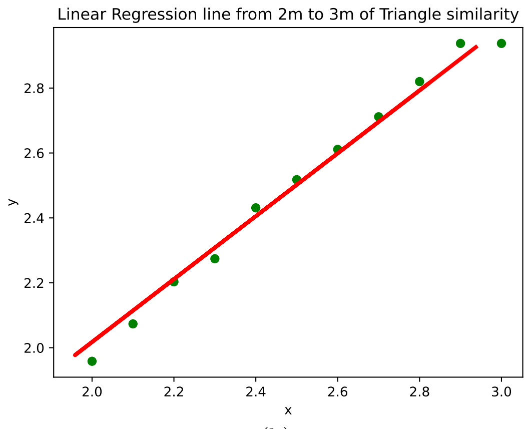


(b)

Figure 4.2: Linear Regression line from 0.5m and 3m of both methods.



(a)



(b)

Figure 4.3: Linear Regression line from 2.0m and 3m of both methods.

- A linear regression model from estimated Triangle method (2.0m to 3.0m) from 11 pairs of variables (x, y of the following form is generated, the line for this linear regression see Figure 4.3:

$$y = 0.96899027 \cdot x + 0.07966402$$

  or [estimate distance] = 0.96899027 · [real distance] +0.07966402

- A linear regression model from estimated Monodepth2 method (2.0m to 3.0m) from 11 pairs of variables (x, y of the following form is generated, the line for this linear regression see Figure

(4.3):

$$y = 1.06466776 \cdot x - 0.16191041$$

or [estimate distance] = 1.06466776 · [real distance] −0.16191041

Table 4.1: *RMSE* parameters for linear models

Models	RMSE
Monodepth2 (26 Observations)	0.1013
Triangle (26 Observations)	0.0267
Monodepth2 (11 Observations)	0.0176
Triangle (11 Observations)	0.0289

From table 4.1, the error values from models are displayed. Models are made up of pairs of values ranging from $0.5m$ to $3m$. Monodepth2 method, which is the total result from 0.5m to 3m distance, produces a lot more accurate model than Triangle method, which is formed from the estimation results of Monodepth2. However, Triangle method's error is considerably bigger. On the other hand, when the distance is between 2 and 3 meters, the error of model Monodepth2 method is significantly lower than the error of model Triangle method. That indicates that any method can provide an approximation of the distance between the two points, but Triangle method can only measure the distance between the two points when they are relatively close to one another. The greater the distance between the two points, the more accurately Monodepth2 method provides the estimated result.

Chapter 5

Conclusion

This thesis develops a framework for combining the tasks of object detection and object distance estimation using a single camera. This work includes two approaches for detecting objects: with YOLOv4-tiny algorithm and the Triangular similarity and Monodepth2 methods to estimate their distances. The thesis provides an overview of neural networks, deep neural networks, and deep learning for computer vision. The YOLOv4-tiny method, which is utilized in object identification, is described in detail. We show the methodologies for both tasks, object identification and distance estimation, by running a series of experiments with varying distances and object sizes. During the mission to detect objects, YOLO did an excellent job quite rapidly. While the YOLO algorithm performs substantially better on all of the measures we examined, it also gives a high frame rate for real-time applications, which is an important consideration. To avoid having to choose just the most visually appealing area of a picture, the YOLO technique predicts classes and bounding boxes for the whole picture during one single run of the algorithm. The most important contribution made by this thesis is the illustration of two different combinations. The first combination is YOLOv4-tiny combined with Triangular Similarity , while the second combination is YOLOv4-tiny combined with Monodepth2. The outcomes of each expression are distinct, and the efficiencies achieved at various distances are similarly variable. However, in general, we are able to show that Monodepth2 is more accurate in determining the distance to objects at greater distances. The implementation of the specified algorithms in the form of an application is covered in Chapter 3 and the performance of the algorithms are discussed. We describe the findings of our experiments in terms of accuracy in Chapter 4. In real-world object detection, the suggested thesis technique demonstrates its capacity to recognize objects both rapidly and correctly. The accuracy of the particular computed distances was tested, and the findings revealed a low error, further demonstrating the efficiency and usefulness of the combination of detecting and estimating object distance in this application.

References

1. LEE, Jeonghun; HWANG, Kwang-il. YOLO with adaptive frame control for real-time object detection applications. *Multimedia Tools and Applications*. 2021-09. Available from DOI: 10.1007/s11042-021-11480-0.
2. ROSEBROCK, Adrian. *Find distance from camera to object/marker using Python and OpenCV*. 2015-01. Available also from: https://pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/.
3. SANKAR, Sanjay; SS, Vasanthakumar; RIFAYEE HUSSAIN Z, Mohamed; KUMAR K, Saravana; SARAVANAN, Dineshkumar; PD, Rathika. Advanced driver assistant system. *Indian Journal of Computer Science*. 2021-08, vol. 6, p. 35. Available from DOI: 10.17010/ijcs/2021/v6/i3-4/165410.
4. BARTOLOZZI, Chiara; MANDLOI, Neeraj; INDIVERI, Giacomo. Attentive motion sensor for mobile robotic applications. *Proceedings - IEEE International Symposium on Circuits and Systems*. 2011-05, pp. 2813–2816. Available from DOI: 10.1109/ISCAS.2011.5938190.
5. NORDICCODER. *Deep Neural Network*. 2019-10. Available also from: https://nordiccoder.com/blog/deep-neural-network/.
6. SIMPLILEARN. *What is Perceptron: A Beginners Guide for Perceptron*. 2022-02. Available also from: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron.
7. HASEEB, Muhammad Abdul. Machine Learning Techniques for Autonomous Multi-Sensor Long-Range Environmental Perception System. 2021. Available from DOI: 10.26092/elib/465.
8. LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998, vol. 86, no. 11, pp. 2278–2324. Available from DOI: 10.1109/5.726791.
9. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. 2012, vol. 25. Available also from: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

10. ZOU, Zhengxia; SHI, Zhenwei; GUO, Yuhong; YE, Jieping. *Object Detection in 20 Years: A Survey*. arXiv, 2019. Available from DOI: `10.48550/ARXIV.1905.05055`.

11. MALLICK, Satya. *Histogram of Oriented Gradients explained using OpenCV*. 2016-12. Available also from: `https://learnopencv.com/histogram-of-oriented-gradients/`.

12. GIRSHICK, Ross B.; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, vol. abs/1311.2524. Available from arXiv: `1311.2524`.

13. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *CoRR*. 2014, vol. abs/1406.4729. Available from arXiv: `1406.4729`.

14. GIRSHICK, Ross B. Fast R-CNN. *CoRR*. 2015, vol. abs/1504.08083. Available from arXiv: `1504.08083`.

15. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross B.; SUN, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, vol. abs/1506.01497. Available from arXiv: `1506.01497`.

16. HUI, Jonathan. *Understanding Feature Pyramid Networks for object detection (FPN)*. 2018-03. Available also from: `https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c`.

17. HUI, Jonathan. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. 2018-03. Available also from: `https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088`.

18. LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian; REED, Scott E.; FU, Cheng-Yang; BERG, Alexander C. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, vol. abs/1512.02325. Available from arXiv: `1512.02325`.

19. LIN, Tsung-Yi; GOYAL, Priya; GIRSHICK, Ross B.; HE, Kaiming; DOLLÁR, Piotr. Focal Loss for Dense Object Detection. *CoRR*. 2017, vol. abs/1708.02002. Available from arXiv: `1708.02002`.

20. REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. *CoRR*. 2018, vol. abs/1804.02767. Available from arXiv: `1804.02767`.

21. JIANG, Zicong; ZHAO, Liquan; LI, Shuaiyang; JIA, Yanfei. Real-time object detection method based on improved YOLOv4-tiny. *CoRR*. 2020, vol. abs/2011.04244. Available from arXiv: `2011.04244`.

22. LEARN. *The evolution of YOLO: Object detection algorithms*. 2021. Available also from: `https://blog.superannotate.com/yolo-object-detection/`.

23. MANISHGUPTA, Jebaseelan. *YOLO — You Only Look Once*. Towards Data Science, 2020-05. Available also from: `https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4`.

24. BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*. 2020.

25. SIMONYAN, Karen; ZISSERMAN, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. Available from eprint: `arXiv:1409.1556`.

26. DUAN, Kaiwen; BAI, Song; XIE, Lingxi; QI, Honggang; HUANG, Qingming; TIAN, Qi. Centernet: Keypoint triplets for object detection. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6569–6578.

27. DU, Xianzhi; LIN, Tsung-Yi; JIN, Pengchong; GHIASI, Golnaz; TAN, Mingxing; CUI, Yin; LE, Quoc V; SONG, Xiaodan. Spinenet: Learning scale-permuted backbone for recognition and localization. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11592–11601.

28. TAN, Mingxing; LE, Quoc. Efficientnet: Rethinking model scaling for convolutional neural networks. In: *International conference on machine learning*. 2019, pp. 6105–6114.

29. WANG, Chien-Yao; LIAO, Hong-Yuan Mark; WU, Yueh-Hua; CHEN, Ping-Yang; HSIEH, Jun-Wei; YEH, I-Hau. CSPNet: A new backbone that can enhance learning capability of CNN. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 390–391.

30. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*. 2015, vol. 37, no. 9, pp. 1904–1916.

31. CHEN, Liang-Chieh; PAPANDREOU, George; KOKKINOS, Iasonas; MURPHY, Kevin; YUILLE, Alan L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*. 2017, vol. 40, no. 4, pp. 834–848.

32. LIU, Songtao; HUANG, Di, et al. Receptive field block net for accurate and fast object detection. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 385–400.

33. WOO, Sanghyun; PARK, Jongchan; LEE, Joon-Young; KWEON, In So. Cbam: Convolutional block attention module. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

34. LIN, Tsung-Yi; DOLLÁR, Piotr; GIRSHICK, Ross; HE, Kaiming; HARIHARAN, Bharath; BELONGIE, Serge. Feature pyramid networks for object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

35. LIU, Shu; QI, Lu; QIN, Haifang; SHI, Jianping; JIA, Jiaya. Path aggregation network for instance segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8759–8768.

36. GHIASI, Golnaz; LIN, Tsung-Yi; LE, Quoc V. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 7036–7045.

37. LIU, Songtao; HUANG, Di; WANG, Yunhong. Learning spatial fusion for single-shot object detection. *arXiv preprint arXiv:1911.09516*. 2019.

38. ZHAO, Qijie; SHENG, Tao; WANG, Yongtao; TANG, Zhi; CHEN, Ying; CAI, Ling; LING, Haibin. M2det: A single-shot object detector based on multi-level feature pyramid network. In: *Proceedings of the AAAI conference on artificial intelligence*. 2019, vol. 33, pp. 9259–9266. No. 01.

39. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*. 2015, vol. 28.

40. LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian; REED, Scott; FU, Cheng-Yang; BERG, Alexander C. Ssd: Single shot multibox detector. In: *European conference on computer vision*. 2016, pp. 21–37.

41. REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

42. LIN, Tsung-Yi; GOYAL, Priya; GIRSHICK, Ross; HE, Kaiming; DOLLÁR, Piotr. Focal loss for dense object detection. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

43. GKARAGKOUNIS, Athanasios K. *The consumer society and the Mediterranean town of Rethemnos, Crete, southern Greece*. Swansea University (United Kingdom), 2010.

44. RASHWAN, Abdullah; KALRA, Agastya; POUPART, Pascal. Matrix Nets: A new deep architecture for object detection. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019.

45. TIAN, Zhi; SHEN, Chunhua; CHEN, Hao; HE, Tong. Fcos: Fully convolutional one-stage object detection. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9627–9636.

46. DAI, Jifeng; LI, Yi; HE, Kaiming; SUN, Jian. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*. 2016, vol. 29.

47. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross. Mask r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

48. YANG, Ze; LIU, Shaohui; HU, Han; WANG, Liwei; LIN, Stephen. Reppoints: Point set representation for object detection. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9657–9666.

49. TECHZIZOU. *YOLOv4 vs YOLOv4-tiny*. medium, 2021-02. Available also from: `https://medium.com/analytics-vidhya/yolov4-vs-yolov4-tiny-97932b6ec8ec`.

50. ZHAO, Haipeng; ZHOU, Yang; ZHANG, Long; PENG, Yangzhao; HU, Xiaofei; PENG, Haojie; CAI, Xinyue. Mixed YOLOv3-LITE: A Lightweight Real-Time Object Detection Method. *Sensors*. 2020, vol. 20.

51. HASEEB, Muhammad Abdul. DisNet: A novel method for distance estimation from monocular camera. In: 2018.

52. MEGALINGAM, Rajesh Kannan; SHRIRAM, Vignesh; LIKHITH, Bommu; RAJESH, Gangireddy; GHANTA, Sriharsha. Monocular distance estimation using pinhole camera approximation to avoid vehicle crash and back-over accidents. In: 2016-01, pp. 1–5. Available from DOI: `10.1109/ISCO.2016.7727017`.

53. GODARD, Clément; MAC AODHA, Oisin; FIRMAN, Michael; BROSTOW, Gabriel. *Digging Into Self-Supervised Monocular Depth Estimation*. arXiv, 2018. Available from DOI: `10.48550/ARXIV.1806.01260`.

54. HANG, Diem. *Ho Ngoc Ha thuot tha trong ta ao dai trang tren duong pho Ha Noi*. 2016-07. Available also from: `https://bloganchoi.com/ho-ngoc-ha-thuot-tha-trong-ta-ao-dai-trang-tren-duong-pho-ha-noi/`.

55. TRE, Tri thuc. *Hay yeu mot co gai di xe so*. 2016. Available also from: `https://afamily.vn/hay-yeu-mot-co-gai-di-xe-so-20160128032818252.chn`.

56. KUMARI, Khushbu; YADAV, Suniti. Linear regression analysis study. *Journal of the Practice of Cardiovascular Sciences*. 2018-01, vol. 4, p. 33. Available from DOI: `10.4103/jpcs.jpcs_8_18`.

57. PAVELESCU, Florin Marius. Features of the ordinary least square (ols) method. implications for the estimation methodology. *Journal for Economic Forecasting*. 2004-01, vol. 1, pp. 85–101.

58. VERSCHAE, Rodrigo; RUIZ-DEL-SOLAR, Javier. Object Detection: Current and Future Directions. *Frontiers in Robotics and AI*. 2015, vol. 2. ISSN 2296-9144. Available from DOI: `10.3389/frobt.2015.00029`.

59. TAN, Mingxing; PANG, Ruoming; LE, Quoc V. Efficientdet: Scalable and efficient object detection. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10781–10790.

60. SOLAWETZ, Jacob. *YOLOv5 New Version Explained [May 2022]*. Roboflow Blog, 2020-06. Available also from: `https://blog.roboflow.com/yolov5-improvements-and-evaluation/`.