Ministry of Higher Education and

Scientific Research

Babylon University

College of Engineering

Electrical Engineering Department

Lite Object Detection Model for Resource-Constrained Devices Using TinyML

A Thesis

Submitted to the Department of Electrical Engineering

University of Babylon

In partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering / Electronic

By

Ruqaya Alaa Ebrahim Issa

Under Supervision of

Prof. Dr. Ehab AbdulRazzaq Hussein

and

Assistant prof. Dr. Hilal Abdul Hussein Al-libawy

2024 A.D 1446 A.H

بسم الله الرحمن الرحيم

صدق الله العلي العظيم

المجادلة: 1

Supervisors Certification

I certify that this thesis titled **"Lite Object Detection Model for Resource-Constrained Devices Using TinyML"** was prepared by **Ruqaya Alaa** under my supervision at the Electrical Engineering Department, College of Engineering at the University of Babylon, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering/Electronics.

Supervisors

Signature:

Name*:* ***Prof. Dr. Ehab A. Hussein***

Date: / / 2024

and

Signature:

Name: Assistant prof. Dr. Hilal Al-Libawy

Date: / / 2024

Given the available recommendation, we forward this thesis for debate by the examining committee.

Head of the Electrical Engineering Department

Signature:

Name: ***Prof. Dr. Qais Kareem Omran***

Date: / / 2024

Abstract

Vehicle safety is an important aspect of modern transportation, aiming to protect passengers and reduce the risk of injury in the event of an accident. Technological advancements in recent years have resulted in the creation of Advanced Driver Assistance Systems (ADAS), which improve vehicle safety. An important function of the ADAS system is to identify blind spots in automobiles, and areas around the vehicle that are not visible to the driver. These blind spots pose significant risks and can lead to accidents, particularly while changing lanes.

This work will address the problem of blind spots because current detection solutions often rely on external sensors and complex infrastructure, such as radar, Light Detection and Ranging (LIDAR), or image classification, but with expensive processors and cameras, which limits their widespread adoption, especially in budget-constrained markets.

The proposed model will use a combination of computer vision and advanced computing techniques to detect vehicles in blind spot areas in real-time. A lightweight Convolutional Neural Network (CNN) model is deployed on an embedded device, a small computer built into a larger system to do a specific job such as a microcontroller, to process real-time photo captures from embedded cameras. The CNN model analyses the captured frames and identifies the motorcycle close to the host vehicle. This study compares the performance of lite algorithms that are suitable for embedded devices, including Single Shot Detector- Feature Pyramid Network (SSD FPN-Lite), You Only Look Once (YOLO), and Fast Objects More Objects (FOMO).

Two datasets were used, consisting of three-wheeled motorcycles, which are currently contributing to accidents in Iraq due to their small size

and limited blind spot visibility. The first dataset was collected from Karbala city center. The images were taken at different angles, dimensions, and times. The second is a Kaggle dataset to cover other shapes of vehicles in different countries.

The proposed system is successfully implemented for the first time for the our knowledge-on the limited resources (Arduino Nano 33BLE) and the board Portenta H7. The best results have been gained using DL FOMO model implemented on Portenta H7. Finally, the highest test accuracy 96.55% and the best inference time was achieved in time up to 30 milliseconds which helps to run the proposed blind spot system in real time.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
ADAS	Advanced Driver Assistance System
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CV	Computer Vision
DL	Deep Learning
FMCW	Frequency-Modulated Continuous Wave
FOMO	Faster Objects, More Objects
GPIO	general-purpose input/output
HAT	Hardware attached on top
IOT	Internet of Things
IDE	integrated development environment
IOU	Intersection over Union
LIDAR	Light Detection and Ranging
mAP	Mean Average Precision
ML	Machine Learning
RAM	Random Access Memory
SBC	single-board computer
SSD FPN-Lite	Single Shot Detector- Feature Pyramid Network
TinyML	Tiny Machine Learning
VBSDS	Vehicle Blind Spot Detection System
YOLO	You Only Look Once

List of Symbols

Abbreviation	Definition
D	represents the entire dataset.
D_{train}	Is the training set.
D_{val}	Is the validation set.
$\cup$	Denotes the union of sets.
$\cap$	Denotes the intersection of sets.
$\emptyset$	Represents the empty set.
T_{end}	the end time of inference.
T_{start}	the start time of inference.
R_n	Recall in threshold n
P_n	Precision in threshold n
TP	Correctly detected objects.
FN	Missed objects that the model failed to detect.
FP	Incorrectly detected objects or multiple detections of the same object.
N	N is the total number of classes or categories.
AP_i	AP_i is the Average Precision (AP) for each class or category i.
F1	F score
y	is the output vector
W	is the weight matrix

x	is the input vector
b	is the bias vector.
$F_{\mathrm{Shortcut}}(X)$	the original input .
$Fi(X)$	the output of the sequential operations.
K	the kernel size
C_{in}	the number of input channels.
C_{out}	the number of output channels.
H	the height of the feature map.
W	width of the feature map.

Chapter One
Introduction

Chapter One

Introduction

1.1 Vehicle safety

Vehicle safety is critically important, because it can save lives, stop injuries, and lower the financial and social costs linked to traffic accidents. A major source of death worldwide, road traffic accidents can be avoided in many cases by putting in place efficient safety features in cars. Road accidents not only result in fatalities but also in a great deal of injuries, from mild to serious. Through their protection during collisions, vehicle safety features reduce the possibility and severity of injuries to occupants [1].With regard to long-term disability, medical costs, and rehabilitation expenditures, this has major public health ramifications.

Healthcare systems and society at large can bear less of the load by enhancing automotive safety. Economic losses from traffic accidents are also high and include medical bills, property damage, lost output, and legal fees [2]. By reducing or eliminating accidents, improving vehicle safety helps to offset these expenses. In addition, aspects of vehicles that improve consumer confidence in automotive products. Safety-first manufacturers show their dedication to the welfare of their customers, which boosts brand loyalty and competitiveness in the market. For cars, governments and regulatory bodies impose safety rules and regulations to guarantee they satisfy the minimal safety standards [3]. Safer roads and communities benefit society as a whole by lowering the amount of accidents and the related human and financial losses [4]. Technologies in materials science, engineering, and automotive design are advanced in part because of the quest of vehicle safety. These advancements advance technology generally and add to safety as well. All things considered, car safety is a complex

problem that affects people personally [5], in communities, and in society at large. Working together, stakeholders can save lives, stop injuries, and make roads safer for everyone by giving safety top priority in vehicle design, manufacture, and regulation.

1.2 Blind Spot

In the context of vehicles, the word "blind spot" refers to the areas surrounding the vehicle that are not directly visible to the driver, either through the mirrors or by turning their head [6]. The blind areas might differ based on the vehicle's design, size, and configuration. Blind spots usually encompass the areas directly behind the vehicle (rear blind spot) and the sides of the vehicle, especially towards the rear (side blind spots). Blind spots present a safety hazard by potentially concealing other vehicles, pedestrians, bicycles, or obstructions. This significantly raises the chances of accidents, especially when making lane changes, merging, or executing turning manoeuvres [7].

The "2022 Road Traffic Injuries" study by the World Health Organisation reveals that over 50% of fatalities resulting from traffic crashes involve vulnerable individuals, including pedestrians, cyclists, and motorcyclists [8]. The blind area of vehicles remains a significant contributing factor in many collisions. Despite the recent introduction of modern safety systems that include costly technology such as Lidar or high-resolution cameras, it remains challenging to identify susceptible individuals, especially when the line of sight is blocked.

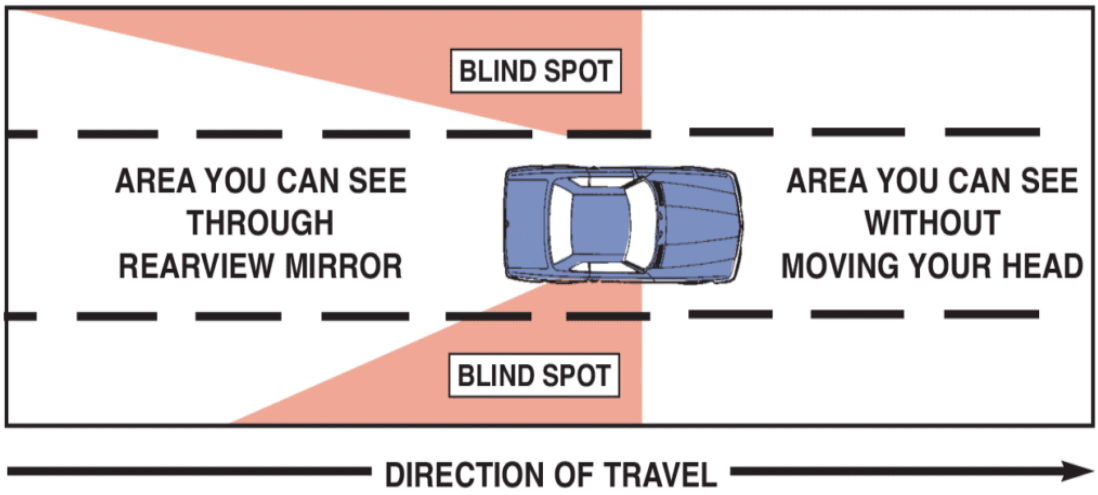

Figure.1.1 Blind spot [9]

1.3 Literature Review: Methods for Addressing Blind Spots in Vehicles

Blind spots in vehicles pose a significant safety concern, increasing the risk of accidents, particularly during lane changes, merging, or turning maneuvers. Over the years, researchers and automotive engineers have developed various methods to mitigate this problem and improve driver awareness of their surroundings. This literature review addresses the effectiveness of different approaches used to address blind spots in vehicles.

Bin-Feng Lin et al,. 2012 [10] presented a blind zone vehicle detection system to help drivers make safe lane changing decisions. The paper proposes a vision-based system to detect sedan vehicles in the blind spot zone during lane changes, using partial features and multiple machine learning models. The correct detection is determined by comparing the overlapping area between the filmed video and the system result. One of the limitations of this study is that it cannot detect when OverlapRatio occurs between vehicles, and the system cannot work in real time.

Din-Chang Tseng, et al,.2014[11] Proposed a lane change assistance system by mounting two cameras beneath the side mirrors of the host car. These cameras would record back side view photos in order to detect approaching vehicles. The suggested system has four stages: estimate of weather adaption threshold values, optical flow detection, static feature identification, and detection decision. The suggested system has the capability to identify adjacent cars that are travelling at varying speeds morever, the proposed system can also adapt to different weather conditions and environmental situations. The results showed a detection rate of 96%. However, the limits of this work were not done in real time in addition to the use of cameras with high resources.

Kiefer, et al,. 2017 [12] provide another common solution is the use of blind spot mirrors, which are additional convex mirrors mounted on the side mirrors to provide a wider field of view. Research suggests that blind spot mirrors effectively enhance visibility and reduce blind spot-related accidents .However, some drivers may find them challenging to adjust correctly or may experience distortion in the reflected image.

Zhao et al,.2019 [13] proposed in their Paper that an advanced deep learning technique using cameras is suggested for blind spot detection in Advanced Driver Assistance Systems (ADAS), to substitute the conventional radar-based approach. The process includes shifting blind spot detection to an image classification assignment and creating an effective neural network utilizing deep learning architectures. The objective is to get fast processing speed and minimal computational complexity for real-time embedded devices. Two deep learning models, YOLOv7 and Faster RCNN, were trained on a specifically gathered dataset to identify road characteristics. The models attained mean Average Precision (mAP) ratings of 87.20% and 75.64%, respectively. However

this work was done using Nvidia Quadro p6000 hardware, which is considered high in cost and resources.

Adnan, Z., et al. 2020 [14] presented a method that use an ultrasonic sensor to identify blind spots and employs automatic steering to manage the car's path. The ultrasonic sensor measures the time difference and calculates the distance to an impediment when it is detected in the blind region. Although ultrasonic sensors are efficient in detecting immobile things within their designated range, their ability to detect moving objects may vary. When it is crucial to accurately detect moving objects, it may be necessary to utilize additional sensors or algorithms to enhance precision and dependability. In addition, if both objects are rapidly approaching or receding from the ultrasonic sensor, the estimated distance may be imprecise as a result of the Doppler Effect.

M. Nor., et al. 2020[15] they demonstrated the design and development of a Vehicle Blind Spot Detection System (VBDS) using an OpenMV M7 with radar sensor. One of the most important limitations of radars is that they cannot detect more than one object at a time, and to improve the accuracy of multiple detections, it may require the integration of additional hardware or more advanced processing techniques, which can increase cost and complexity, in addition to 24 GHz, which may It is less accurate in locating small objects with high accuracy, especially in crowded environments or when objects are close together. Since 24 GHz is a common frequency, there is a possibility of interference with other devices operating on the same frequency. Interference can result in decreased detection accuracy or loss of signals.

Negishi et al,.2022 [16] designed and developed a wireless adjustable blind spot detection system using Lidar .The system starts by detecting the

presence of a vehicle in the blind spot area. Once the vehicle is detected, the Lidar sensor sends a signal to the microcontroller. In addition, data is transmitted wirelessly from the Arduino microcontroller to the BLiM application, where the alert appears. Data is transferred to the mobile device via the Bluetooth module HC-05. The system's reliance on a Lidar sensor to detect vehicles may be affected by environmental factors and may result in false alerts or missed detections. Wireless transfer of data to the BLiM app via Bluetooth may also cause delays, affecting system responsiveness.

Wang, et al. **2022[17]** A blind spot detection system was developed using numerous recognition techniques to enable automatic monitoring of blind spots inside a person's field of vision. The inspection system comprises a stationary depth camera, together with associated applications and operating stations. The design consists of three distinct modules: an object detection module, a blind spot analysis module, and an attention module. Nevertheless, the effectiveness of this work relies on the use of a costly camera and has not yet been tested in real-world conditions to determine its accuracy and effectiveness. It is important to note that this system solely identifies people and does not have the capability to detect vehicles.

Kim et al, 2023 [18] developed a radar blind spot detection system using frequency-modulated continuous wave (FMCW) technology to enhance driving safety. The study utilises specific methodologies to design antennas for the BSD radar system that operate between the frequencies ranges of 77 GHz and 79 GHz. The paper outlines the fundamental technologies of the Blind Spot Detection (BSD) radar system, including the signal processing and tracking algorithms. Additionally, it discusses the architecture of the BSD radar system.. However, it possesses specific

vulnerabilities that necessitate careful consideration. These factors encompass range and Doppler uncertainty in situations involving several targets, as well as vulnerability to interference.

Pourhasan Nezhad., et al. 2023 [19]. Their proposition entails an efficient blind spot warning system that operates with a single camera sensor on each side. The work comprises two components. Begin by doing a thorough examination of the categorization of hazardous and secure circumstances in a dynamic setting with items in motion. Furthermore, in order to differentiate between dangerous circumstances and secure environments, we implement a state-of-the-art (SOTA) object detector that has been previously trained. This detector is capable of monitoring cars in consecutive frames and subsequently calculating the distances of the tracked vehicles with an average margin of error of 6%. Furthermore, the suggested system employs the relative velocity of vehicles to alert drivers of hazardous circumstances, as well as to identify items located in areas that are not easily visible. This classification procedure does not occur in real time.

Table1.1 Compare between related researches on blind spot

Paper ID	Method Techniques	Hardware	Affected by climate	Limitation
[10]	partial features	Side-mounted camera	×	• Cannot detect when OverlapRatio occurs between vehicles. • The system cannot work in real time
[11]	Motion features and static features	Two cameras are mounted under side mirrors	×	This work was not implemented in real time, and high-cost resources were used. In addition, this system is an image

				classification system that cannot determine the type of vehicle.
Paper ID	**Method Techniques**	**Hardware**	**Affected by climate**	**Limitation**
[12]	convex mirrors	×	×	Some drivers may encounter difficulties in properly adjusting them or may perceive distortion in the reflected image.
[13]	DL model and Fully Connected Network	Camera-Based Blind Spot Detection and Nvidia Quadro p6000	×	• Nvidia Quadro P6000 are high-end graphics cards. It needs a power source, a motherboard to connect to, drivers and appropriate programs to work, and therefore it needs a computer system to operate, so this system cannot be generalized because it is impractical. • Works with image classification system
[14]	Ultrasonic Sensor	Arduino and an ultrasonic sensor	✓	• It typically has a limited range of a few meters . • It struggles with detecting soft, porous, or angled surfaces, and their performance can be hindered by interference from other sensors. • Ultrasonic sensor generally has lower resolution and accuracy, slower response times, and a blind zone close to the sensor.
[15]	Radar Sensors	24 GHz Radar Sensors with OpenMV M7	✓	24 GHz may be less accurate in locating small objects with high accuracy,

				especially in crowded environments or when objects are close together. Since 24 GHz is a common frequency, interference with other devices operating on the same frequency is likely. Interference can result in decreased detection accuracy or loss of signals.
Paper ID	**Method Techniques**	**Hardware**	**Affected by climate**	**Limitation**
[16]	Multi-LIDAR network.	Lidar sensor	✓	• Surface reflectivity: Dark or non-reflective surfaces can be difficult to detect accurately because they reflect less laser light. • Range limitations: Effective range is limited and can be affected by laser power and target reflectivity. • Power consumption: LiDAR systems can consume a lot of power and therefore require a battery. • Interference: Multiple LiDAR systems operating in close proximity can interfere with each other, causing inaccurate readings.
[17]	DL modal (YOLOv3)	Depth Camera D455	×	this system is not integrated with the vehicle itself, but instead it is an external device installed on street poles to notify the driver of the presence of people in the blind area. The system is prohibitively costly if it aims to be universally useful.

Paper ID	Method Techniques	Hardware	Affected by climate	Limitation
[18]	Frequency Modulated Continuous Wave (FMCW) radar technology	Blind Spot Detection BSD radar system	✓	• Accuracy limitations: The most important limitation of radar is the inability of the radar to distinguish between two objects that are very close to each other in terms of range, angle, or speed. • Ghost targets and multipath: Inaccurate readings can result from multipath effects, when the radar signal travels several ways to the target and back, or from reflections from buildings or other large objects that create phantom targets (ghost targets). • Doppler blindness: Targets moving perpendicular to the radar beam can be a problem for radar systems.
[19]	DL modal	High resolution single camera sensor on each side	×	Despite the efficiency of the work, it is difficult to implement and disseminate it in practice because it was implemented by taking a video using the side cameras and then processing it using algorithms on the computer.

1.4 The goal of the proposed system

After providing an overview of the key systems utilized in identifying blind spots in recent years, we aim to address the deficiencies of earlier research in this area, such as the problem of real-time detection and the inability to detect small or overlapping vehicles. . The goal of this thesis is to develop an effective and efficient blind spot detection system tailored for resource-constrained devices using Tiny Machine Learning (TinyML). This approach aims to improve vehicle safety through real-time blind spot detection, providing an affordable and scalable alternative to more resource-intensive traditional systems. Also to design and implement cost-effect system based on DL algorithm and limited resources hardware. Additionally, the implemented system has reliable detection score to be trusted in real time environment.

1.5 Thesis objective

The system design will prioritize accuracy, responsiveness, and minimal computational requirements to ensure optimal performance on limited hardware. Below are the most important detailed points that show the importance of this study:

1. Use object detection algorithms implemented on microcontrollers, prioritizing efficiency and affordability with focus on the TinyML implementation framework
2. Design, development and evaluation of a low-cost blind spot detection system for vehicles.
3. Investigating the feasibility of identifying the type of vehicles on the driver's side to enhance situational awareness.

4. Compare the cost and efficiency of the developed system with existing blind spot detection systems to evaluate its competitiveness and affordability.

1.6 Thesis Layout

This thesis consists of five chapters. Chapter one provides a comprehensive background on vehicle safety, specifically addressing blind spots in vehicles, in addition to an overview of literature review that focused on the same problem. The remaining chapters are structured as follows:

Chapter Two in this chapter an introduction of computer vision, its subfields, and its connection to deep learning is given. It describes the fundamental ideas of tinyML, together with its advantages in terms of hardware and software. Furthermore covered in this chapter are the ideas behind object detection and the TinyML-adapted methods employed in this thesis

Chapter Three explains the methodology used to detect blind spots. It describes the research design, the hardware used, data collection methods, and all the steps that enable the model to achieve the objectives.

Chapter Four the results were presented and the performance of the models was discussed on each algorithm used, in addition to comparing the model with other systems.

Chapter Five summarizes the contributions of this work and suggests future research paths for enhancing vehicle safety, therefore closing the thesis. Finally reference and appendix.

2

Chapter Two

Theoretical Background

Chapter Two

Theoretical Background

2.1 Introduction

In an era where technology is revolutionizing every facet of our lives, the realm of transportation stands at the forefront of innovation. As vehicles become increasingly autonomous and connected, the pursuit of safer roads has become an imperative. From advanced driver assistance systems to autonomous vehicles, the integration of artificial intelligence (AI) and machine learning has paved the way for unprecedented enhancements in vehicle safety. This chapter is separated into two parts, where the first part shows computer vision and its types, especially object detection. While the second part explains the meaning of the Tiny Machine Learning and all its related software and hardware.

2.2 Computer vision

In this thesis, blind spots in vehicles will be detected using deep learning algorithms to detect objects, which is considered one of the applications of computer vision. Therefore, it is necessary to know the concept of computer vision and where it begins.

Machine Learning is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and techniques that allow computers to learn from and make predictions or decisions based on data, without being explicitly programmed to perform specific tasks. In other words, the goal of machine learning is to enable computers to learn from experience (data) and improve their performance over time. It includes supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, algorithms learn from labeled data to make predictions

or decisions. Unsupervised learning involves discovering patterns and structures in unlabeled data. Reinforcement learning focuses on learning optimal actions based on feedback from the environment [20].

Though deep learning is a subfield of machine learning that specialises on building and training multi-layer artificial neural networks to extract knowledge from input. Deep learning aims to simplify the process by which computers may learn and comprehend complicated patterns or representations straight from unprocessed data without the requirement for feature extraction by hand [21].

In this thesis, "Computer Vision," term is also crucial since it deals with helping computers to perceive and comprehend visual data from the actual environment. It entails work like object detection, image segmentation, facial recognition, and video activity recognition. Computer vision algorithms interpret visual stimuli and extract relevant information from pictures or movies [22]. Figure 2.1 shows the relationship between these terms, Machine learning and deep learning are two subsets of artificial intelligence (AI), which deals with teaching computers to learn from data and make predictions or decisions. The section highlighted in the contrasting color represents the focus of our work, which involves the application of deep learning techniques in the field of computer vision.

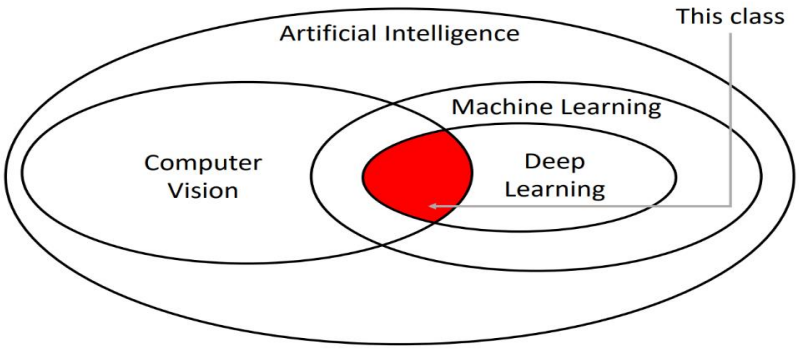

Figure 2.1. The relationship between machine learning, deep learning and computer vision

Computer vision is a field within artificial intelligence that focuses on enabling computers to interpret and understand visual information from the real world. In other words, machine learning and deep learning provide the tools and techniques necessary for computers to learn from visual data and deep learning [23]. Performing tasks in computer vision. They complement each other to enable the development of advanced vision systems capable of understanding and interpreting images and videos.

2.3 Different between Traditional Computer Vision workflow and deep learning workflow

Previously images are analyzed to extract as many features as possible. These features are then used to create a definition, known as a bag-of-words, for each object class. During the deployment phase, these definitions are queried within other images. If a substantial proportion of the characteristics present in one bag-of-words are also found in another image, the image is categorized as containing the particular object (e.g. chair, horse, etc.).

The challenge with this conventional method lies in the need to determine the crucial qualities in every individual image. As the quantity of classes to categorize grows, the process of extracting features gets increasingly burdensome. The determination of which traits most accurately characterize various classes of objects is reliant upon the discernment of the Computer Vision (CV) engineer and an extensive process of experimentation and refinement. Furthermore, each feature definition requires the control of several parameters, all of which the computer vision engineer has to carefully modify. While end-to-end learning, which is essentially given to the machine as a collection of photos annotated with the classes of objects found in each image, was presented

by deep learning [24]. On the given data, a deep learning model is developed, enabling neural networks to recognise the underlying patterns in image classes and pinpoint the most prominent and descriptive characteristics for every distinct object class. It is well acknowledged that Deep Neural Networks (DNNs) outperform standard methods, albeit they come with trade-offs in terms of computational demands and training duration. Due to the advancements in computer vision, the workflow of CV engineers has seen significant changes.

The traditional practice of manually extracting features has been replaced by the use of deep learning architectures, requiring engineers to possess knowledge and competence in iterating through these complex models as depicted in Figure 2.2.

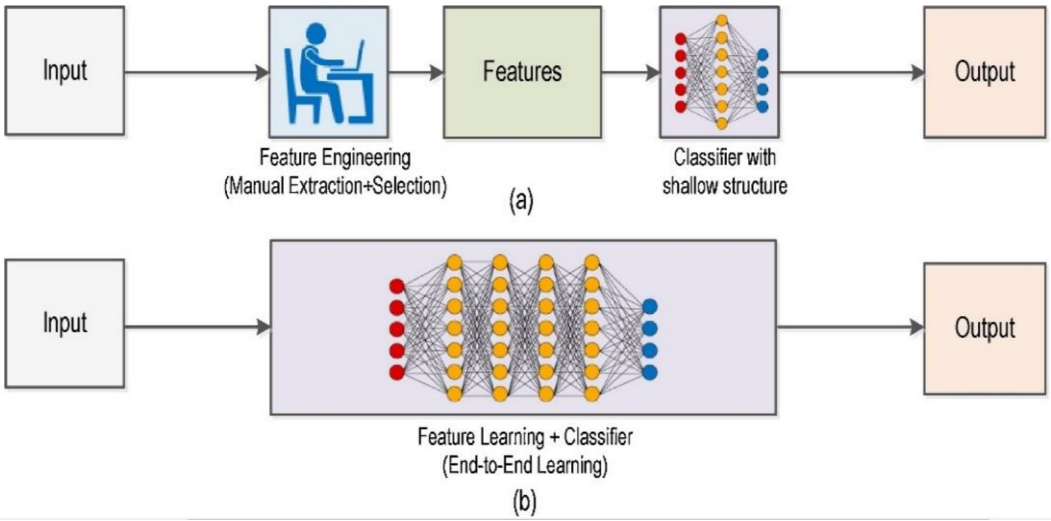

Figure 2.2 (a) Traditional Computer Vision workflow vs. (b) Deep Learning workflow. Figure from [25].

2.4 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) enhance prediction accuracy by leveraging large datasets and computational power, expanding the limits of what was achievable previously intractable problems are

currently being resolved with superhuman precision. Classification of images is a prime illustration of this. Since its resurgence in 2012 [26] by Krizhevsky, Sutskever, and Hinton, DL has dominated the field due to its significantly superior performance in comparison to conventional methods.

A significant increase in the capacity to recognize objects can be attributed to the development of CNNs, which has had a significant impact on the area of CV in recent years and is responsible for the development of CNNs [27]. This surge in development has been made possible by an increase in the amount of data that is accessible for training neural networks as well as an increase in the amount of computing power that is available. Both of these factors have contributed to the growth of neural networks.

2.4.1 CNNs Layers:

1. Convolutional Layer: is the core building block of a Convolutional Neural Network (CNN). It is designed to automatically and adaptively learn spatial hierarchies of features from input images. The components of the Convolutional Layer is **Filters/Kernels:**

- **Definition:** A filter, also known as a kernel, is a small matrix of weights used to extract features from the input image [28].
- **Operation**: The filter slides over the input image (or previous layer's output) and performs element-wise multiplications followed by summation. This process is known as convolution.
- **Output:** The result of the convolution operation is a feature map (or activation map), which highlights the presence of certain features in the input.

2. Activation Layer (ReLU): The ReLU (Rectified Linear Unit) layer is an essential component of many neural networks, particularly Convolutional Neural Networks (CNNs) [29]. It introduces non-linearity into the network, enabling it to learn complex patterns and relationships. Neural networks without non-linear activation functions can only model linear relationships, regardless of their depth. By applying ReLU, the network can model complex, non-linear relationships.

The ReLU function is defined as :

$$\mathbf{ReLU(x) = max\ (0, x)} \qquad \mathbf{(2.1)}$$

This means that if the input *x* is positive, it returns x; otherwise, it returns 0.

3. Pooling Layer: is a fundamental component of Convolutional Neural Networks (CNNs) that performs a form of reducing the spatial dimensions of the input feature maps. This helps to achieve spatial invariance and reduces the computational complexity of the network, as well as the risk of overfitting. Also it helps the network recognize features regardless of their location in the input image. The most common types of pooling layers is:

- **Max Pooling:** selects the maximum value from each patch of the feature map, capturing the most prominent features such as edges or corners and making the network robust to small translations.
- **Average Pooling:** calculates the average value within each patch, retaining more contextual information and providing a smoother representation of the input.

4. Fully Connected layers (Dense): are critical in neural networks, especially for tasks requiring high-level reasoning. They are used extensively in both the middle and final stages of networks, transforming

the learned features from convolutional and pooling layers into the final output, such as class scores. The input to a fully connected layer is typically a flattened vector of features from the previous layer, whether it be a convolutional, pooling, or another fully connected layer. The layer performs a linear transformation on the input vector using its weights and biases. The transformation can be expressed as:

$$y=Wx+b \qquad (2.2)$$

Where: y is the output vector, is the weight matrix, x is the input vector, is the bias vector.

5. **Batch Normalization (BN):** is a technique used in neural networks to improve training speed, stability, and performance. It normalizes the activations of each layer to have a consistent distribution, which helps mitigate issues related to internal covariate shift.

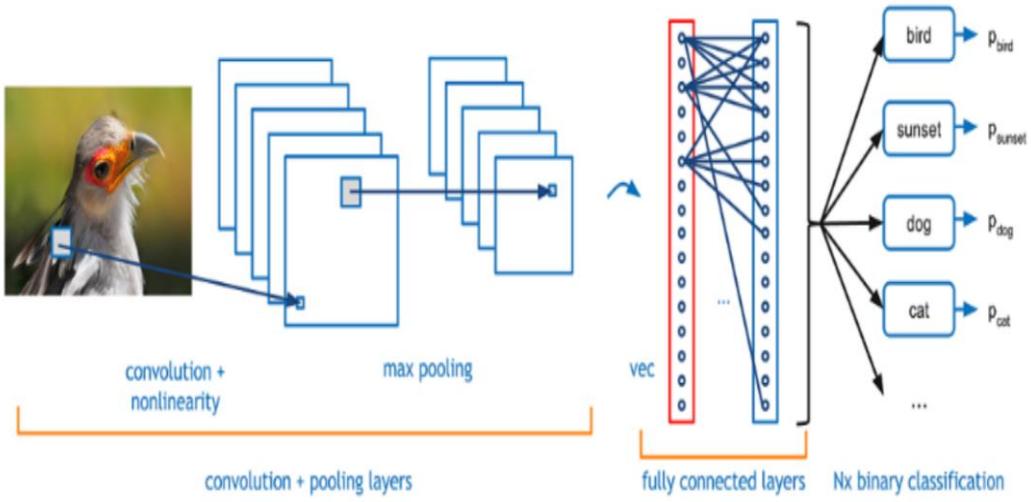

Figure 2.3 Building blocks of a CNNs [30]

2.4.2 Hyperparameter of CNNs

- **Batch Size**

The context of machine learning, particularly deep learning, and the "batch size" refers to the number of training examples utilized in one iteration of training. When training a neural network, you don't typically feed the entire dataset into the network at once, especially if the dataset is large. Instead, you divide the dataset into smaller batches, and each batch is passed through the network one at a time. The network computes the gradients of the loss function with respect to the parameters (weights and biases) using these examples, and then updates the parameters accordingly [31].

- **Learning rate**

This is the most important hyperparameter because it can greatly change the accuracy of the model. In general, a small learning rate takes a very long time to converge, while a large learning rate may perform well at the beginning, but does not converge as a result of overshooting [32]. The use of learning rates varied across research (e.g. 0.01, 0.0001, 0.001, etc.)

- **Epoch**

Within a neural network, a period signifies a comprehensive learning iteration for a given training data set. The number of epochs is being augmented, and the learning process for the identical training data set is also reiterated. The findings indicate that increasing the number of epochs leads to a higher prediction accuracy [33]. This is observed through the neural network's field loss, which quantifies the absolute inaccuracy between the model's output value and the actual value of the real-world

data. Choosing the number of epochs is not solely based on accuracy, but also on minimising the loss value.

2.5 Tasks of computer vision

Deep learning has revolutionized the field of computer vision, enabling machines to understand and interpret visual data with remarkable accuracy and efficiency [34] .Main three fundamental tasks within computer vision that have seen significant advancements through deep learning are Object Detection, Image Classification, and Image Segmentation [35] ,as shown in table 2.1 and Figure 2.4.

Deep learning has propelled these tasks to new heights of performance and applicability, enabling a wide range of real-world applications from autonomous vehicles to medical diagnosis [36]. Continuous research and development in deep learning algorithms and architectures continue to push the boundaries of what's possible in computer vision.

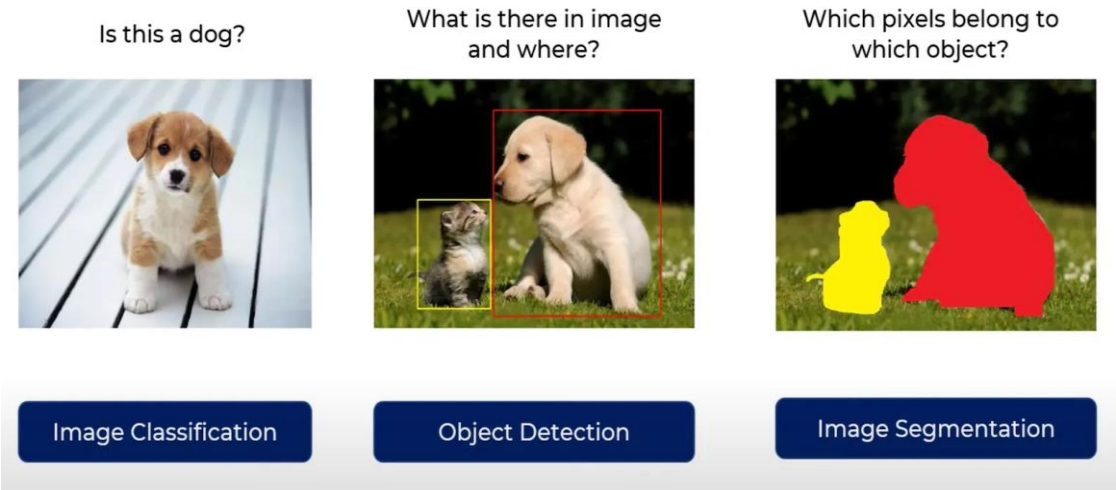

Figure 2.4 different between image classification and object detection and segmentation [37]

Table2.1 Comparison between types of computer vision

Aspect	Object Detection	Image Classification	Image Segmentation
Task Description	Identify and localize objects within an image or video frame.	Assign labels or categories to entire images.	Divide images into semantically meaningful regions or segments.
Objective	Detect and localize multiple objects, often with bounding boxes.	Determine the main category or class of the entire image.	Assign a class label to each pixel in the image, outlining object boundaries.
Output	Multiple bounding boxes around detected objects, with associated labels.	Single label or category representing the predominant content of the image.	Pixel-wise segmentation mask indicating the class of each pixel.
Complexity	Higher complexity due to the need to locate and classify multiple objects simultaneously.	Moderate complexity, focusing on overall content analysis.	High complexity, requiring detailed pixel-level predictions.
Applications	Autonomous vehicles, surveillance systems, object tracking.	Image search engines, content filtering, quality control in manufacturing.	Medical imaging, autonomous robotics, scene understanding.

Aspect	Object Detection	Image Classification	Image Segmentation
Notable Techniques	Region-based CNNs (R-CNN), You Only Look Once (YOLO), Single Shot Multibox Detector (SSD).	Convolutional Neural Networks (CNNs), transfer learning, fine-tuning.	Fully Convolutional Networks (FCNs), U-Net, Mask R-CNN.

In computer vision tasks, precision metrics are crucial for evaluating the performance of deep learning models in tasks like object detection, image classification, semantic segmentation, and more. Here's an introduction to some of the most important precision metrics used in computer vision deep learning:

1. Validation Set

In deep learning, a validation set is an essential component used to evaluate a learned model with data that it was not exposed to during training. It stops overfitting and helps assess how well the model generalizes. Typically, the validation set is used all during training to assess the model's performance on fresh data and, if necessary, modify hyper parameters or early stopping conditions. The relationship between the training set, validation set, and the overall dataset can be mathematically represented by the following equations [38].

$$\boldsymbol{D} = \boldsymbol{D}_{\mathbf{train}} \cup \boldsymbol{D}_{\mathbf{val}}$$
$$\boldsymbol{D}_{\mathbf{train}} \cap \boldsymbol{D}_{\mathbf{val}} = \emptyset \qquad \textbf{(2.3)}$$

Where D represents the entire dataset, D_{train} is the training set, D_{val} is the validation set, ∪ Denotes the union of sets, ∩ Denotes the intersection of sets and ∅ Represents the empty set.

2. Inference Time

Inference time, as used in deep learning discussions, refers to the duration required for a machine learning model to process new data and provide a forecast [39]. Complexity of the model, available computing resources, and processing power of the devices are a few of the factors influencing this important component of real-time applications. The duration of time a model needs to go from receiving input samples to receiving an output forecast is known as inference time. It has a mathematical representation as:

$$\textbf{Inference Time} = T_{\text{end}} - T_{\text{start}} \qquad (2.4)$$

Where T_{start} is the start time of inference and T_{end} is the end time of inference.

3. Test accuracy

In deep learning, test accuracy is a model's performance evaluation on a specific test dataset. Normally, it is assessed by comparing the test dataset's ground truth labels with the model's predictions. Evaluation of the model's capacity to accurately categorise or forecast the intended outputs for the supplied inputs is the aim. Given that it offers information on the model's performance and dependability, test accuracy is a crucial parameter in deep learning. It facilitates the understanding of academics and practitioners of how well the model generalises to unknown data and can be used to compare numerous models or evaluate the effectiveness of different testing techniques [40].

By dividing the number of correct predictions by the total number of predictions, the mathematical formula for determining the accuracy of a test can be achieved.

$$\boldsymbol{Test\ Accuracy} = \frac{\boldsymbol{Number\ of\ Correct\ Predictions}}{\boldsymbol{Total\ Number\ of\ Test\ Instances}} \times \mathbf{100}\% \qquad \textbf{(2.5)}$$

Where Number of Correct Predictions" refers to the number of instances in the test set that are correctly classified by the model and Total Number of Test Instances" is the total number of instances in the test set.

2.6 Object Detection

Object detection is a crucial task in the field of computer vision [41]. Computer vision primarily entails the recognition of several objects inside an image or a video frame. A typical pipeline consists of the acquisition, preparation, analysis, and evaluation of the data [42]. Object detection introduces an additional level of intricacy by not only recognizing items within an image or video but also precisely determining their location using bounding boxes (shown Figure 2.5) [43]. The bounding boxes function as rectangular boundaries that encompass the things of interest. Intelligent systems can gain a comprehensive understanding of their visual environment and make informed decisions and take appropriate actions by utilizing object detection, similar to how humans view the world [44]. This feature enables a diverse array of uses, including autonomous driving, surveillance systems, augmented reality, and robotics.

Deep neural networks have revolutionized object detection tasks, leading to significant progress and enhanced capabilities. Deep neural networks, in contrast to traditional methods that depend on manually designed features and shallow learning models, allow for end-to-end

learning [45]. This means that the networks can automatically extract features from raw pixel data. The shift in paradigm has resulted in notable enhancements in accuracy, as deep architectures have the ability to record complex patterns and variances in the appearance of objects across various scales, positions, and environmental conditions. Furthermore, deep neural networks have exceptional versatility and adjustability, enabling them to be trained on extensive datasets that include a wide range of object categories and scenarios.

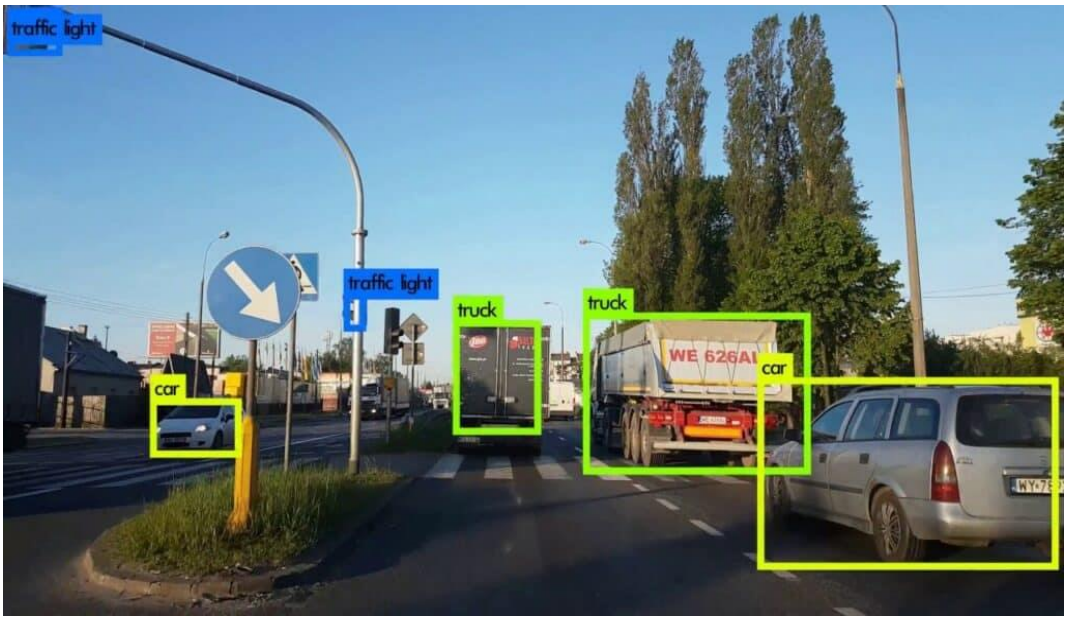

Figure 2.5 object detection in streets [46]

2.7 Performance Evaluation parameters

2.7.1 Confusion matrix

In object detection, a confusion matrix is a useful tool for evaluating the performance of a model. It shows the relationship between the ground truth (actual values) and the model's predictions [47]. While a confusion matrix in classification typically has a fixed structure with rows representing actual classes and columns representing predicted classes, in

object detection, it’s slightly more complex due to the nature of multiple objects and possible overlapping detections.

	Predicted: Positive	Predicted: Negative
Actual: Positive	True Positive (TP)	False Negative (FN)
Actual: Negative	False Positive (FP)	True Negative (TN)

Here's a basic breakdown of how the confusion matrix components apply to object detection:

1. True Positives (TP): The number of correctly detected objects, where the detected bounding box matches the ground truth box (usually defined by an Intersection over Union (IoU) threshold, often 0.5 or higher).
2. False Positives (FP): The number of detected objects that do not match any ground truth box (e.g., wrong detections or duplicates).
3. False Negatives (FN): The number of ground truth objects that were missed by the model (no detected box matches the ground truth box).
4. True Negatives (TN): In object detection, true negatives are less commonly discussed because the focus is on the presence and localization of objects rather than the absence of objects.

2.7.2 Intersection over Union (IoU)

The IoU metric quantitatively measures the similarity between two bounding boxes and is commonly employed to evaluate the accuracy of object detection algorithms. IoU is computed by dividing the area of overlap between the ground truth and the anticipated bounding boxes by the area of union of these boxes.

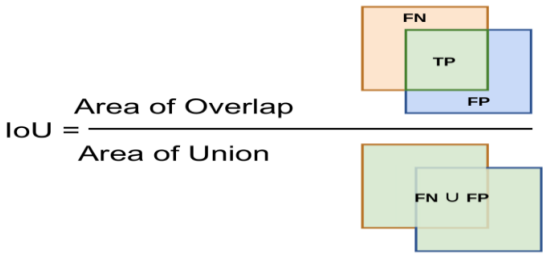

Figure 2.6 The Intersection over Union (IoU) is computed by dividing the overlapping area between the bounding boxes by the union area [48]

As shown in figure 2.6, the region of intersection is the specific area where an object detection algorithm accurately finds an object that precisely matches the ground truth box. This specific choice is commonly referred to as true positives (TP). The denominator's union area merges the detection outcomes and subsequently deducts the true positives. This is done to avoid duplicating the counting of those things. The objects spotted in the blue zone are false positives generated by the model. These are referred to as false positives (FP). The system failed to detect the objects within the orange region, which should have been identified based on the ground truth box. The term used to refer to these missed pixels is "false negatives" (FN). A perfect prediction is denoted by an Intersection over Union (IoU) value of 1, indicating that the anticipated bounding box precisely matches the ground truth bounding box. Therefore, the values of

FP, TP , and FN are all zero. On the other hand, when the IoU value is zero, it indicates that the prediction is erroneous because there is no overlap between the predicted bounding boxes and the actual bounding boxes, shown Figure 2.7. The given expression can be assessed as the subsequent equation:

$$IOU = \frac{TP}{TP+FP+FN} \qquad (2.6)$$

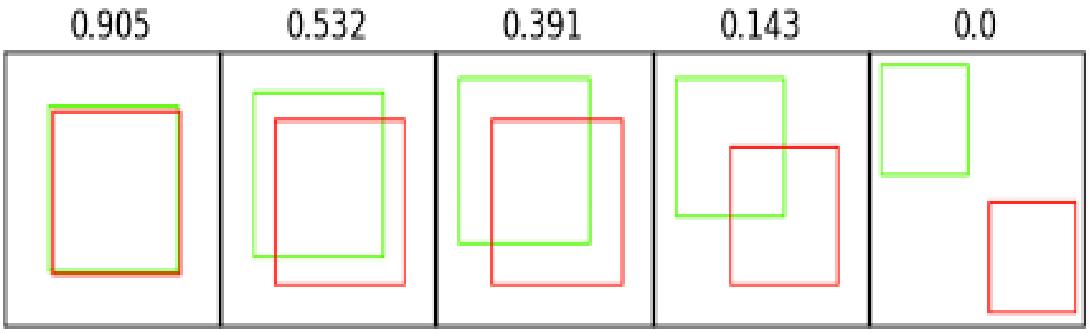

Figure 2.7 How to display intersection over union (IoU) values for object detection [49]

2.7.3 Average Precision (AP)

When evaluating a model's ability to forecast the behaviour of a certain class of objects, the average precision (AP) metric is employed. Our first stop will be at precision so that we can better understand average precision. The ratio of correctly predicted positive instances (TP) to the total number of positive instances (TP + FP) is what the term "precision" means when discussing the accuracy of the model's predictions. These considerations lead to the following conclusions:

$$Precision = \frac{TP}{TP+FP} \qquad (2.7)$$

When calculating average precision, it is important to take into account the precision values at various degrees of recall. Recall, on the

other hand, is a measurement that determines the proportion of all real positive instances (true positives) in the dataset that correspond to the proportion of correctly predicted positive instances. It reveals the extent to which the model is able to locate all instances of positive outcomes.

$$\boldsymbol{Recall = \frac{TP}{TP+FN}} \qquad \textbf{(2.8)}$$

Average Precision (AP) is the area under the precision-recall curve. It is calculated by taking the simple average of the maximum precision values at each recall level.

$$\boldsymbol{AP = \sum_n (R_n - R_{n-1}) P_n} \qquad \textit{(2.9)}$$

where $\boldsymbol{P_n}$ and $\boldsymbol{R_n}$ are the precision and recall at step $\boldsymbol{n}$.

2.7.4 Mean Average Precision (mAP)

The accuracy with which a model predicts a certain object class is expressed by the mean average precision (mAP) measure. The mAP computes the average precision for every class and then averages these values for all classes [50]. It offers a single summary score that represents how well the model performed overall across several classifications.

$$\boldsymbol{mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i} \qquad \textbf{(2.10)}$$

Where $N$is the total number of classes or categories and AP_i is the Average Precision (AP) for each class or category i.

2.7.5 F1 score

The F1 score is the harmonic mean of precision and recall. Mathematically, it is calculated as:

$$\textbf{F1}=\boldsymbol{\frac{TN+TP}{TN+FN+FP+TP}} \qquad \textbf{(2.11)}$$

2.8 Machine learning framework for embedded system (TinyML)

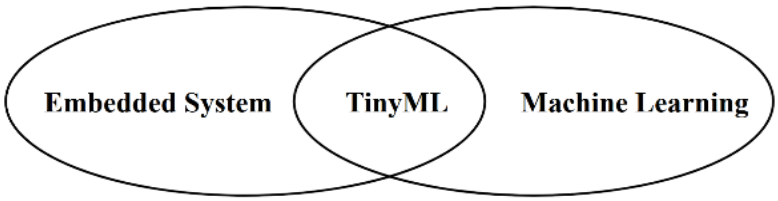

Figure 2.8 Definition of TinyML

Tiny Machine Learning (TinyML) enables cognitive functionalities on low-resource IoT devices, specifically Microcontroller units. To refer to machine learning applications that utilize compact models for very efficient inference with minimal energy consumption [51]. TinyML is an emerging topic that focuses on implementing machine learning algorithms, such as object detection and classification, specifically for microcontrollers (MCUs). The goal is to leverage the large number of MCUs available to do machine learning tasks at the extreme edge, as shown in Figure 2.8, with the help of interfacing systems like sensors. Most microcontrollers (MCUs) operate within the milliwatt power range, which means that an MCU with similar power consumption levels can run on a coin cell battery for more than a year [52].

TinyML will enable the development of innovative services and applications at the edge, relying on distributed edge inference and autonomous decision-making instead of server computation [53]. Specifically, the origins of TinyML may be traced back to the emergence of Edge ML, which refers to the practice of deploying machine learning models directly on edge devices, such as IoT devices, microcontrollers, and other embedded systems, rather than relying on centralized cloud servers.

This approach leverages local computation and storage capabilities, enabling real-time data processing and decision-making closer to where the data is generated [54]. However, while TinyML offers several advantages, it also comes with certain limitations. Table 2.2 outlines the advantages and disadvantages of TinyML to provide a comprehensive overview of its strengths and challenges:

Table 2.2 Advantages and disadvantages of TinyML

Advantages	Disadvantages
Extremely low power usage, suitable for battery-operated devices	Limited processing power due to energy constraints
Real-time processing with minimal latency as computation occurs on the edge	Limited computational capabilities for complex tasks
Enables deployment of AI models on a wide range of small, low-cost devices	Challenging to deploy complex models that require significant memory and processing
Data processing occurs locally on the device, enhancing privacy and security	Limited resources may lead to compromises in data handling and storage
Can operate without constant internet connection, suitable for remote locations	May lack the ability to access large datasets or perform cloud-based updates

While TinyML presents immense potential, it also faces several challenges that must be addressed to fully realize its capabilities:

a. Model Optimization: There is still a substantial obstacle to overcome in the form of the development of highly efficient models that are capable of completing difficult tasks with low resources [55]. The importance of conducting additional research into architecture design and model compression methodologies cannot be overstated.

b. Hardware Limitations: TinyML is still in its infancy compared to the development of dedicated hardware accelerators that are low-power. The advancement of TinyML capabilities will be extremely dependent on the continuation of innovation in hardware design.

c. Energy Efficiency: Managing the amount of electricity that is consumed by gadgets is becoming even more important as technologies continue to advance. Building machine learning algorithms and hardware that are efficient with energy will be essential to TinyML's success in the long run.

Table2.3 compares three machine learning platforms: CloudML, MobileML, and TinyML. CloudML involves deploying machine learning models on powerful cloud servers using GPUs like Nvidia Volta, offering high performance and scalability for complex tasks, but at high costs ($9K) and power consumption (250W). MobileML refers to running models on mobile devices using mobile CPUs, balancing performance and portability for real-time tasks like image recognition, with moderate costs ($750) and power usage (~8W). TinyML deploys models on microcontrollers (e.g., Arm4, Arm7) with extremely low power (0.1-0.3W) and cost ($3-$8), ideal for simple, energy-efficient AI applications at the edge.

Table 2.3 Comparison of hardware for CloudML, MobileML and TinyML [56]

Platform	Architecture	Memory	Storage	Power	Price
CloudML Nvidia V100	GPU Nvidia Volta	HBM 16GB	SSD/Disk TB~PB	250W	$9K

MobileML Cell Phone	CPU Mobile CPU	DRAM 4GB	Flash 64GB	~8W	~$750
Platform	**Architecture**	**Memory**	**Storage**	**Power**	**Price**
TinyML	MCU Arm4 Arm7 Arm7+ Arm4	SRAM 128KB 320KB 521KB	eFlash 0.5MB 1MB 2MB	 0.1W 0.3W 0.3W	 $3 $5 $8

2.9 Software

In TinyML software refers to the lightweight, specialized programs and frameworks designed to run machine learning models on microcontrollers and low-power edge devices. This software includes optimized machine learning libraries (e.g., TensorFlow Lite for Microcontrollers, MicroPython, and an online platform for TinyML)

2.9.1 TensorFlow

TensorFlow is a comprehensive and powerful framework that supports a wide range of machine learning and deep learning applications, making it a popular choice among developers and researchers. TensorFlow is an open-source machine learning framework developed by Google. Here are some features of TensorFlow:

- **Application**: TensorFlow can be used for a variety of tasks, from training models for image recognition and natural language processing to

deploying models on different platforms, including mobile devices and cloud infrastructure [57].

- **Ecosystem**: TensorFlow has a rich ecosystem that includes libraries for data preprocessing, model building, and deployment. Some notable components are TensorFlow Lite (for mobile and embedded devices), TensorFlow.js (for running models in the browser) [58].

- **Keras Integration**: TensorFlow integrates with Keras, a high-level neural networks API, which makes it easier to build and train deep learning models with a user-friendly interface.

Overall, TensorFlow is a versatile tool for anyone looking to develop and deploy machine learning models.

2.9.2 TensorFlow Lite

A Google-developed software framework called TensorFlow Lite (TFL) allows machine learning models to be run on edge devices including mobile devices and microcontrollers. Lite ML applications fit it well because of its lightweight and efficient design. This is why TFL was created especially to support edge-based machine learning. As so, a wide range of effective algorithms can be carried out on peripheral devices with limited resources, such as microcontrollers, cellphones, and other circuits [59]. Without requiring a continuous internet connection or depending on cloud-based services, it allows jobs like image categorization, object identification, and natural language processing to be completed directly on the device.

TFL is a framework made expressly to do machine learning on embedded devices with few kilobytes of memory. Both Android and

smartphones as well as embedded Linux and microcontrollers are among the many platforms with which the program is compatible.

Figure 2.9 shows the process of deploying a machine learning model on a microcontroller using TensorFlow Lite for Microcontrollers. First, a model is trained using TensorFlow and then converted into a lightweight TensorFlow Lite (.tflite) format, optimized for low-resource environments. This .tflite model is further converted into a C array to be integrated into a microcontroller's firmware. The microcontroller uses this model to perform real-time inference on incoming data, allowing for low-latency and energy-efficient machine learning at the edge without needing cloud connectivity.

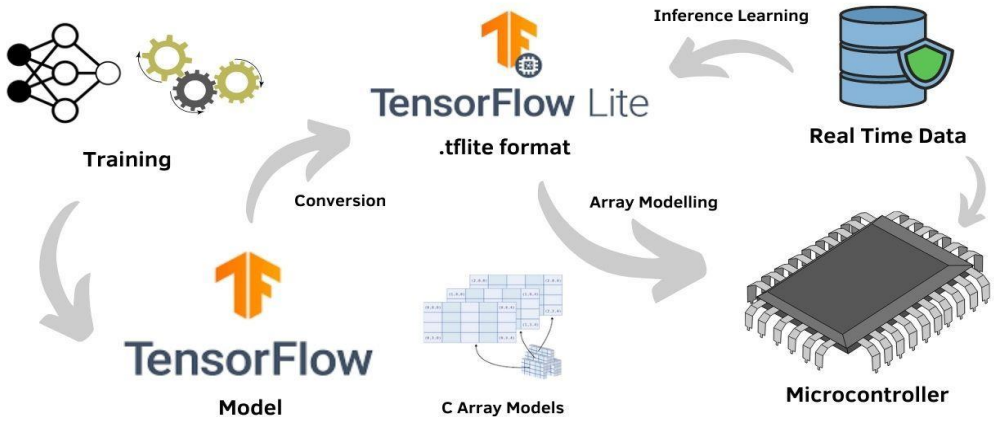

Figure2.9 TensorFlow Lite and it Framework [60]

2.9.3 MicroPython

MicroPython is a small embedded systems and microcontroller-compatible open-source programming language and software implementation. The goal behind MicroPython was to offer a high-level programming language for quick development, prototyping, and deployment on tiny devices. By offering a language as familiar and user-friendly as Python, MicroPython sought to make this process easier.

Furthermore supported are a large number of microcontrollers from different manufacturers, including Raspberry Pi Pico, Arduino, ESP8266/ESP32, STM32 boards, and more [61]. It offers a script mode for executing standalone applications together with an interactive shell for real-time experimentation and debugging. MicroPython has been continuously improved upon and developed since its inception [62].

Today, MicroPython is widely used in various applications such as Internet of Things (IoT) devices, robotics projects, home automation systems, sensor networks, educational platforms, and more. Its simplicity and versatility make it an attractive choice for developers looking to work with microcontrollers without sacrificing ease-of-use or power [63] .

2.9.4 Online Platform For TinyML

An online platform for TinyML is a cloud-based service that simplifies the development, training, optimization, and deployment of machine learning models on microcontrollers and low-power edge devices. These platforms provide tools for training models on various datasets, optimizing them for minimal memory and power usage through techniques like quantization, and converting them into formats suitable for microcontrollers, such as TensorFlow Lite . They also offer features for data collection, model management, and integration with a wide range of hardware,Such as Edge Impulse platform ,which is a service that simplifies the process of generating TinyML models that are specifically targeted for edge devices since it streamlines the process.

2.10 Hardware

TinyML focuses on deploying machine learning models on small, resource-constrained devices, balancing computational power, energy efficiency, and size constraint [64].

Microcontrollers (MCUs) are small, low-power integrated circuits that perform basic computing tasks and execute lightweight ML models. Examples include ARM Cortex-M series (e.g., STM32, Nordic nRF) and ESP32. Microprocessors (MPUs) are more powerful than MCUs and handle more complex models or multitasking, with examples like ARM Cortex-A series and Intel Atom. Field-Programmable Gate Arrays (FPGAs) are customizable integrated circuits used for hardware acceleration of ML tasks, such as Xilinx Zynq and Intel FPGA [65].

Low-power sensors, such as accelerometers and temperature sensors, provide data for ML models and enable smart applications. Memory constraints, including SRAM, Flash, limit the size and complexity of deployable models, making efficient memory management crucial [66], shown Table 2.4. Lastly, power management is essential for battery-operated devices, necessitating careful design to ensure efficient energy usage.

Overall, TinyML hardware advances continue to expand the possibilities for running sophisticated ML models on small, energy-efficient devices.

Table2.4 Difference between Flash memory and RAM memory

Aspect	Flash Memory	RAM (Random Access Memory)
Volatility	Non-volatile: Retains data even when power is off	Volatile: Loses data when power is turned off
Purpose	Stores program code (firmware) and static data	Temporarily stores data and program instructions
Persistence	Retains data long-term	Stores data temporarily during program execution
Access Speed	Generally slower access speed compared to RAM	Faster access speed compared to flash memory
Capacity	Typically larger capacity compared to RAM	Typically smaller capacity compared to flash memory
Usage Examples	Storing program code, configuration data	Storing variables, data structures during execution
Optimization	Often requires efficient programming practices	Memory management is crucial for efficient operation
Power Consumption	Consumes less power compared to RAM	Consumes more power due to constant refresh operations

Overall, embedded devices play a crucial role in enabling automation, connectivity, and intelligence in various industries by providing dedicated functionality in a compact form factor. Also, object detection on embedded devices requires a careful balance between accuracy and resource efficiency to enable real-time performance within the limitations of the hardware. These are most used types of embedded devices that are used in TinyML application:

2.10.1 Raspberry Pi

The Raspberry Pi, depicted in figure 2.12, is an affordable single-board computer (SBC) created by the Raspberry Pi Foundation (raspberrypi.org). The three variants, commonly referred to as 'Raspberry Pi's', are built on a system-on-a-chip architecture. Each model includes an integrated CPU and GPU, on-board memory, and a power input of 5 V DC. Additionally, all models are equipped with a connector specifically designed for connecting a dedicated camera. They also feature a set of general-purpose input/output (GPIO) pins that enable communication with various electrical components, such as LEDs, buttons, servos, motors, power relays, and an extensive selection of sensors. Hardware attached on top (HAT) refers to specialized expansion boards that can be connected to the GPIO pins [67].

Raspberry Pi, depending on the model, consumes between 2.5W to 7W of power. For example, the Raspberry Pi 4 Model B has a 64-bit quad-core ARM Cortex-A72 processor running at 1.5GHz and offers up to 8GB of RAM. It supports high-speed data transfer with USB 3.0 and Gigabit Ethernet. Storage is managed via a microSD card, which can support capacities up to 1TB. Overall, the Raspberry Pi is a compact, energy-efficient computer suitable for a range of applications, from simple tasks to more demanding workloads like light computing and multitasking [68].

Figure 2.10 Raspberry Pi [69]

2.10.2 Arduino

Arduino is a customized microcontroller that is open-source and designed for physical computing. It operates on a single board. The Arduino board possesses an exceptional processing capability of up to 16 MHz, which may vary based on the specific board being used. Arduino made significant progress by introducing a user-friendly integrated development environment (IDE) and standardized hardware. Arduino will come in handy for controlling motors, LEDs, or interfacing sensors [70]. There are a huge variety of Arduino boards, all with different capabilities, as shown in Table 2.5. Additionally, Arduino's platform is widely used in education, making it an ideal tool for introducing students to computer vision concepts. Moreover, the rapid prototyping capabilities of Arduino facilitate quick iteration and development cycles, fostering innovation in computer vision across domains such as robotics, IoT, and automation.

Table 2.5 Some devices that can detect objects and support TinyML

Hardware	Processor	CPU Clock	Flash Memory	SRAM Size	Power	Organization
Alif Ensemble E7	Cortex-M55 with Ethos-U55 microNPUs	400MHz	4MB	4MB	1.71-3.6V	Alif Semiconductor
Arduino Nicla Vision	Dual Ann Cortex M7ZM4	M7: 480MHz and M4: 240MHz	2MB	1MB	3.7V Li-po battery	Arduino

Seeed Grove Vision AI Module	Hi max HX6537-A	400MHz	2MB	2MB	5V	Seed Studio
Portenta H7 Lite	Dual Cortex M7+M4	M7: 480MHz and M4: 240MHz	**16MB**	8MB	3.7V	Arduino
Nano 33 BLE	Nordic nRF52840 Cortex-M4F	M4 64 MHz	1MB	256KB	3.3V	Arduino

2.11 Overview of Object Detection Algorithms

Object detection is a key area of computer vision that involves identifying and locating objects within an image or video by providing both the class label and the bounding box coordinates for each object [71]. Object detection algorithms can be broadly categorized into two main types: two-stage detectors and one-stage detectors [72]. Each type has its strengths and weaknesses in terms of accuracy, speed, and computational requirements, which determines their suitability for real-time applications.

Two-stage detectors, such as R-CNN (Region-Based Convolutional Neural Networks), Fast R-CNN, Faster R-CNN, and Mask R-CNN, work by first generating region proposals that are likely to contain objects and then classifying these proposals and refining the bounding boxes. These methods are highly accurate due to their two-step approach, but they are

relatively slow because they require separate steps for region proposal and classification [73].

One-stage detectors, such as YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), and RetinaNet, perform object localization and classification in a single pass through the network, making them significantly faster and more suitable for real-time applications [74].

2.11.1 YOLO (You Only Look Once)

YOLO represents a singular approach to object detection, swiftly analyzing entire images via a neural network to yield immediate forecasts regarding bounding boxes and class likelihoods. This methodology strikes an optimal balance between accuracy and speed, rendering it well-suited for applications requiring instantaneous responsiveness [75].

YOLOv5 is one-stage object detection algorithm known for its speed and accuracy. It builds on the YOLO family of models, which are designed for real-time object detection tasks. YOLOv5 offers a good balance between computational efficiency and detection performance, making it highly suitable for real-time applications such as video. The YOLOv5 architecture comprises three essential components: backbone, neck, and head, which collectively enable accurate and dense predictions, as depicted in the Figure2.11. The backbone refers to a network that has been trained in advance to extract detailed and meaningful features from images. This process decreases the spatial resolution of the image while simultaneously enhancing the resolution of the features (channels). The second model is the neck model, which is employed to extract the unique pyramids. This model exhibits strong generalization capabilities among objects of varying sizes and scales. Ultimately, the head model is employed to execute the concluding stage procedures. The anchor boxes are utilized

on the feature maps to generate the ultimate output, which includes classes, topicality scores, and bounding boxes. [70].

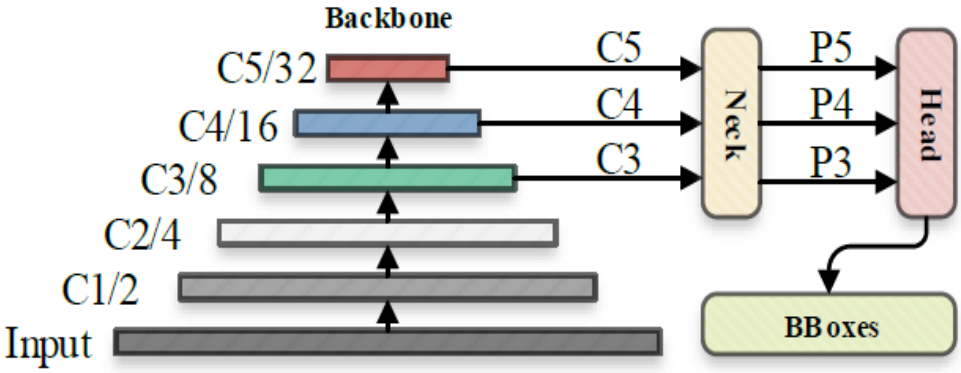

Figure 2.11 YOLO5 Architecture [76]

2.11.2 MobileNetV2

MobileNetV2 architecture was introduced and significantly improved upon the original MobileNet architecture as has been rigorously designed for low-power, low-latency operations for mobile and edge devices. Lightweight design, low memory footprint, real-time inference capabilities, compatibility with TinyML frameworks, versatility, and the balance between accuracy and efficiency make it a suitable choice for deployment in TinyML applications, especially on microcontrollers and other resource-constrained devices.[77]. Figure 2.12 displays that in MobileNetV2, there are two types of blocks. One is a Inverted Residual Block with step 1. The other is a Standard Convolutional block with step 2 to reduce the size to make it suitable for running on low-resource devices.

a. In blocks with stride = 1, the input starts and goes through a 1×1 convolution layer with ReLU6, which increases the number of channels. The output goes through a 3×3 depthwise convolution layer with ReLU6. Then, the output goes through a 1×1 convolution layer without Linear, which reduces the number of channels to the original number. Finally, the

output is added to the original input using fast add. The final output is the sum of the original input and the output of the sequential operations as follows:

$$F(X) = F_{Shortcut}(X) + Fi(X) \qquad (2.12)$$

Where $F_{Shortcut}(X)$ is the original input and $Fi(X)$ is the output of the sequential operations.

In blocks with stride = 2: it reduces spatial dimensions which helps reduce computational cost and increase efficiency. The spatial dimensions of the final output are smaller than the spatial dimensions of the original input. The original input cannot be added to the final output directly because the dimensions are different, making the addition impossible without modification.

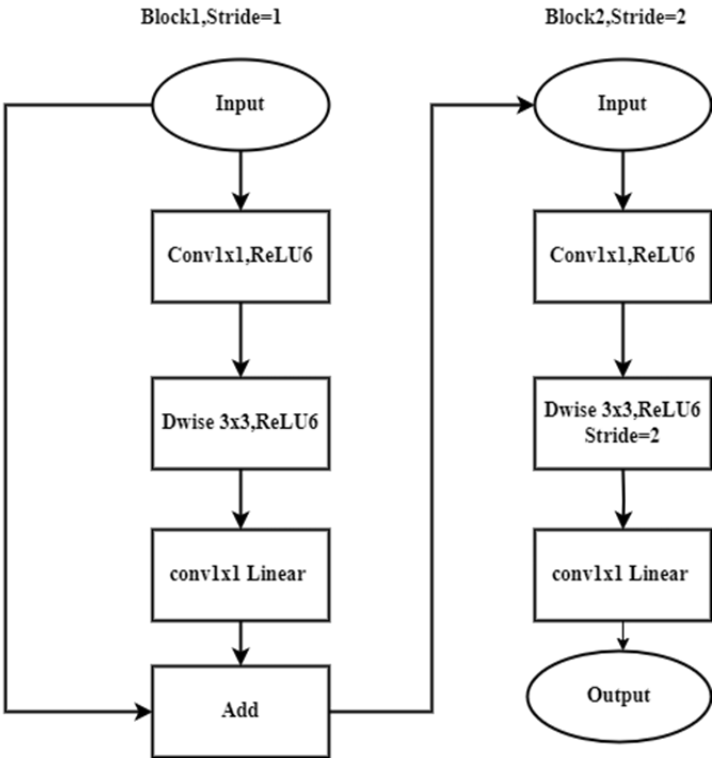

Figure 2.12 MobileNetV2 architecture where Stride block 1 Inverted Residual Block and Stride block 2 Standard Convolutional Block

- **Depthwise Separable Convolution**

Depthwise separable convolutions are a key component of MobileNetV2, helping to make it efficient and lightweight .Its decompose a standard convolution into two separate layers:

1. **Depthwise Convolution**:
   - Applies a single convolutional filter per input channel (input depth).
   - Each filter is applied independently to each channel.
   - This reduces the computational complexity significantly compared to standard convolutions.
2. **Pointwise Convolution**:
   - Uses 1×1 convolutions to combine the outputs of the depthwise convolution.
   - This step effectively merges the output channels from the depthwise convolution, increasing the depth of the feature maps.

- **Comparative between Standard Convolution and Depthwise Separable Convolution Mathematically**

For a standard convolutional layer, the computational cost is given by:

$$cost_{standerd} = K^2 * C_{in} * C_{out} * H * W \qquad (2.13)$$

Where K is the kernel size , C_{in} the number of input channels, C_{out} the number of output channels, H the height of the feature map and Wwidth of the feature map.

Depthwise separable convolutions split this into two operations: depthwise convolution and pointwise convolution.

1. **Depthwise Convolution**:
    - Applies a single filter per input channel.
    - The computational cost is:

$$cost_{\text{Depthwise}} = K^2 * C_{in} * H * W \quad (2.14)$$

2. **Pointwise Convolution**:

- Applies a 1×1 convolution to combine the outputs of the depthwise convolution.
- The computational cost is:

$$cost_{\text{Pointwise}} = 1 * 1 * C_{in} * C_{out} * H * W \quad (2.15)$$

$$cost_{\text{Pointwise}} = C_{in} * C_{out} * H * W \quad (2.16)$$

The total computational cost of a depthwise separable convolution is the sum of the two operations:

$$Cost_{total} = cost_{\text{Pointwise}} + cost_{\text{Depthwise}} \quad (2.17)$$

In summary, Standard Convolution applies each filter to all input channels, resulting in high computational cost and many parameters, making it less efficient [78]. Depthwise Separable Convolution splits into depthwise and pointwise convolutions, significantly reducing computational cost and parameter count, making it more efficient and ideal for lightweight models.

Here are some types of networks that are included in MobileNetV2:

1. MobileNetV2 Single Shot Detector- Feature Pyramid Network (SSD FPN-Lite)

The FPN Lite module is employed to integrate the outputs of the MobileNetv2 and SSD layers, hence enhancing the accuracy of object

detection by including contextual information and merging data from various scales [79]. This model utilizes the Feature Pyramid Network (FPN) to extract features from the input image, while employing the ResNet50 Convolutional Neural Network (CNN) as the underlying network, as shown in Figure 2.13 MobileNetV2 SSD FPN-Lite is a lightweight object detection architecture designed for efficiency on mobile and edge devices.

It uses the MobileNetV2 backbone, which consists of layers such as Inverted Residual Blocks and depthwise separable convolutions to reduce computational complexity while maintaining accuracy. Layers like Conv2D and Bottleneck layers extract multi-scale features. The FPN-Lite (Feature Pyramid Network Lite) component includes layers like 1x1 Convolutions and 3x3 Convolutions to merge higher-level semantic features with lower-level spatial details, enhancing multi-scale feature representation. The SSD Detection Head comprises Convolutional layers for predicting bounding box offsets and class scores, Classification layers, and Regression layers to handle various object scales and shapes. This combination of layers allows MobileNetV2 SSD FPN-Lite to achieve a good balance of speed and accuracy, making it ideal for real-time applications on resource-constrained devices [80].

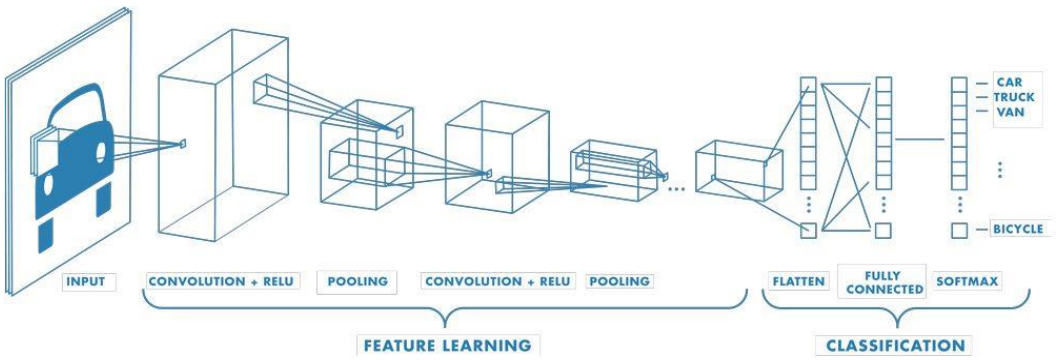

Figure 2.13 MobileNetV2 SSD FPN-Lite Architecture [81]

2. FOMO (Faster Objects, More Objects)

FOMO is a lightweight object detection algorithm designed specifically for efficient object detection on edge devices with limited computational resources. FOMO is particularly well-suited for deployment on devices with have restricted computing capabilities, such as mobile phones, Internet of Things devices, and embedded systems. In these contexts, the ability to quickly and accurately identify objects is of utmost importance.

The adaptability of this technology will offer advantages to applications such as object detection, scene comprehension, and monitoring [82]. It addresses the need for efficient and scalable object detection solutions that can handle large-scale datasets and real-time processing requirements. The system can monitor numerous objects simultaneously in real time with significantly lower processing power and memory use than MobileNet SSD or YOLOv5.The system integrates an image feature extractor, often a shortened portion of MobileNet-V2, with a fully convolutional classifier to approach object detection as object classification across a grid, executed simultaneously, as shown in Figure 2.14. FOMO's primary design choices prioritize object identification based on item position rather than object size. Classical bounding boxes are no longer necessary based on this fact. A detection relying on object centroids is sufficient. It is important to maintain interoperability with other models, thus the training picture input continues to utilize bounding boxes.

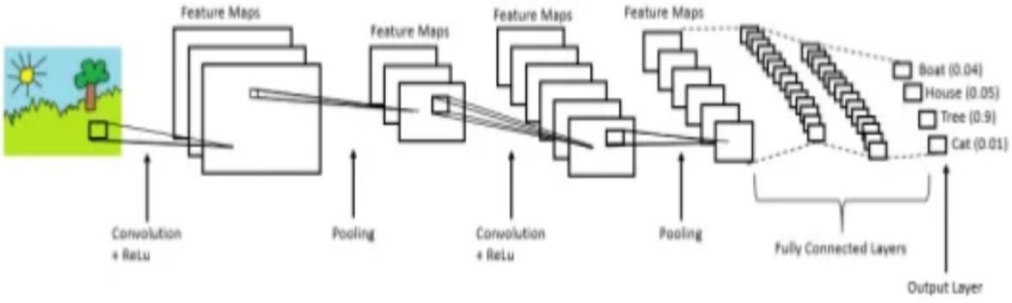

Figure2.14 FOMO Architecture [83]

When FOMO first began to design, one of its main objectives was to implement object detection on microcontrollers, which frequently have very little RAM and flash. The tiniest FOMO version (96x96 grayscale input, MobileNetV2 0.05 alpha) operates on a Cortex-M4F at 80MHz with ~10 fps and <100KB RAM. Additionally, the entire MobileNet network is not used for FOMO, it is cut off near the top so you only use a portion of it. FOMO utilizes MobileNetV2 as the foundational model for its trunk and typically reduces the spatial dimensions from input to output by 1/8th (e.g., a 96x96 input yields a 12x12 output), as shown in Figure 2.15. MobileNet is truncated at the intermediate layer block_6_expand_relu. Modifying the cut point will lead to a varied spatial reduction. For instance, cutting at a higher position like block_3_expand_relu will result in FOMO only reducing the spatial dimensions by 1/4 (e.g. a 96x96 input will provide a 24x24 output).

There is also an important parameter in this algorithm which is Alpha, in the FOMO method refers to the scaling factor that regulates the network's breadth, which influences the number of channels in each layer about the infrastructure. Alpha **0.1** is setting alpha to 0.1 further reduces the number of channels in each layer of the MobileNetV2 network. With alpha 0.1, each layer will have just 10% of the channels as opposed to the

basic architecture [84]. Alpha 0.1 produces a highly lightweight network design, drastically lowering computational overhead and memory use. However, this significant drop in network width may result in a bigger loss of accuracy than utilizing a higher alpha value. When alpha is set to **0.35**, each layer in the network will have fewer channels than if alpha was set to 0.5. This minimizes the network's computational cost and memory footprint, making it even more lightweight. However, selecting a lower alpha value may result in a bigger drop in accuracy when compared to the standard MobileNetV2 architecture.

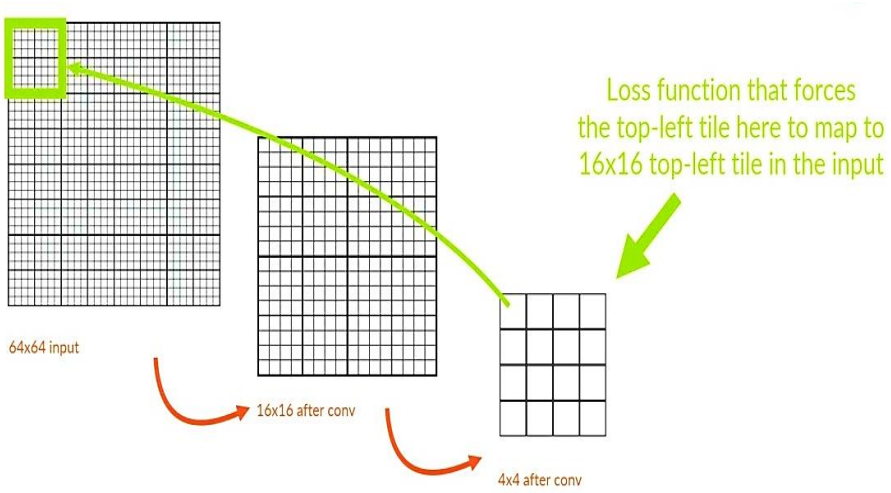

Figure 2.15 Explain how the FOMO algorithm works [85]

- **Understanding Heatmaps in FOMO**

Heatmaps in FOMO provide a simplified and efficient way to detect objects by highlighting regions where objects are likely present(shown Figure 2.16). This solution utilizes the decreased computing burden and memory demands, rendering it well-suited for implementation on microcontrollers and other devices with limited resources [86]. Heatmaps enable FOMO to achieve practical object detection in scenarios where traditional methods would be too demanding.

- **Heatmap Generation**

  1-Feature Extraction: The backbone network (a lightweight CNN) processes the input image and extracts important features. These features are passed through subsequent layers to produce a lower-resolution feature map.

  2-Heatmap Creation:

i. The feature map is processed by a series of convolutional layers to generate a heatmap. Each pixel in this heatmap corresponds to a specific region in the input image [87].

ii. The value of each pixel in the heatmap represents the confidence score or probability that an object is present at that location. Higher values indicate a higher likelihood of an object's presence.

- **Interpreting Heatmaps**
  - **i. Visual Representation:** Heatmaps can be visualized as images where the intensity or color of each pixel indicates the confidence level of object presence.
  - **ii. Localization:** The heatmap provides an approximate location of objects. Instead of bounding boxes, the peak (highest value) regions in the heatmap indicate where objects are likely to be located.

- **Advantages of Using Heatmaps**
  - i. **Simplicity:** The model complexity is much decreased by emphasizing the existence and approximate position of items rather than exact bounding boxes. Deploying on microcontrollers with little memory and processing capacity requires this simplicity.

ii. **Efficiency:** Heatmaps are more efficient and appropriate for real-time applications on limited devices since they require less calculations than bounding box prediction.

iii. **Resource Optimization:** Further lowering the memory and processing needs are heatmaps' decreased size with relation to the original input image. This improvement makes FOMO operate effectively on devices with extremely few resources.

- **Post-Processing of Heatmaps**

i. **Thresholding:** After generating the heatmap, a threshold can be applied to filter out low-confidence detections. Only pixels with values above a certain threshold are considered potential object locations [88].

ii. **Non-Maximum Suppression (NMS):** it can be used to make sure that just the most certain detections are retained. NMS keeps just the peak (highest confidence) values in close proximity, therefore suppressing overlapping detections.

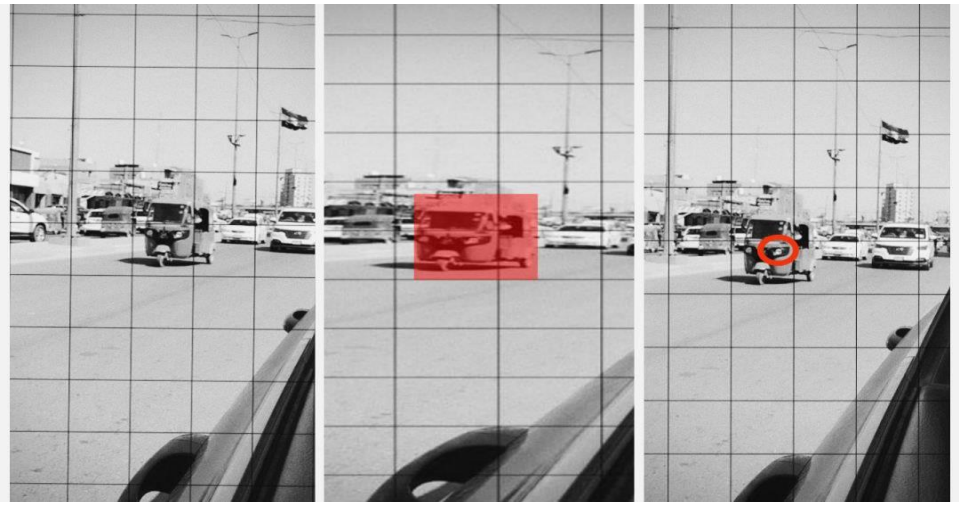

Figure 2.16 Heatmap generation of FOMO algorithm

2.12 Types of training according to the size of the data

2.12.1 Active Learning

In machine learning, active learning is a method applied when it is either time-consuming or impossible to obtain huge volumes of precisely labelled data. Still, an active learning method can be applied to train a model. The procedure begins with training a model on a little dataset. Subsequently, the model can be applied to gather fresh data and assist in choosing new records that it believes to be most promising [89]. After this, the model can be retrained on a bigger dataset and the recommended records can be tagged by a domain expert. An active learning algorithm allows a model to be quickly installed and then actively upgraded while in use. Figure 2.17 shows the whole active learning system.

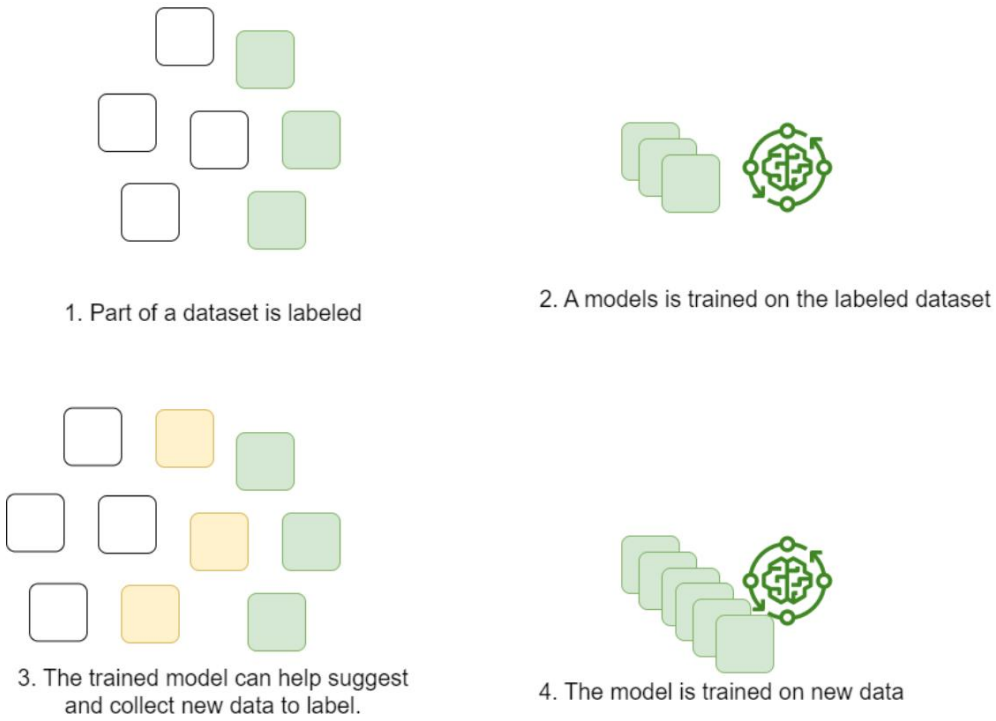

Figure 2.17 Active learning process [90]

2.12.2 Transfer learning

Transfer learning is a machine learning technique where a model trained on one task is reused or adapted for another related task. Instead of training a model from scratch, transfer learning leverages knowledge gained from solving one problem to improve performance on a different, but related, problem [91]. In other word, transfer learning is a powerful technique that accelerates the training process and improves the performance of machine learning models by transferring knowledge from one task to another. It is widely used in various domains to leverage the benefits of pre-existing models and adapt them to new tasks with minimal effort.

A common phenomena observed in deep learning networks trained on images is that in the initial layers of the network, the model attempts to learn low-level properties such as edge detection, colour recognition, and fluctuations in intensities. These features do not seem to be exclusive to a single dataset or task, as they may be applied to any form of image processing, whether it is for spotting a lion or an automobile [92]. In both instances, it is necessary to identify these fundamental characteristics. These traits are present irrespective of the specific cost function or image dataset. Therefore, acquiring knowledge of these characteristics in the context of lion detection can also be used to other tasks, such as human detection.

Here are some key points about transfer learning:

- **Pre-trained Models**: In transfer learning, a pre-trained model is used as the starting point for training on a new task. These pre-trained models are typically trained on large datasets for general tasks such as image classification or natural language understanding [93].

- **Fine-tuning**: After initializing the model with pre-trained weights, the model is fine-tuned on a smaller dataset specific to the new task. Fine-tuning involves updating the model's parameters using the new dataset while retaining the knowledge learned from the pre-trained model [94].
- **Adaptation:** Transfer learning allows models to adapt to new tasks with less data and computational resources compared to training from scratch. It is particularly useful when the new task has limited labeled data or when training from scratch would be prohibitively expensive. Figure 2.18 shows the whole transfer learning system.

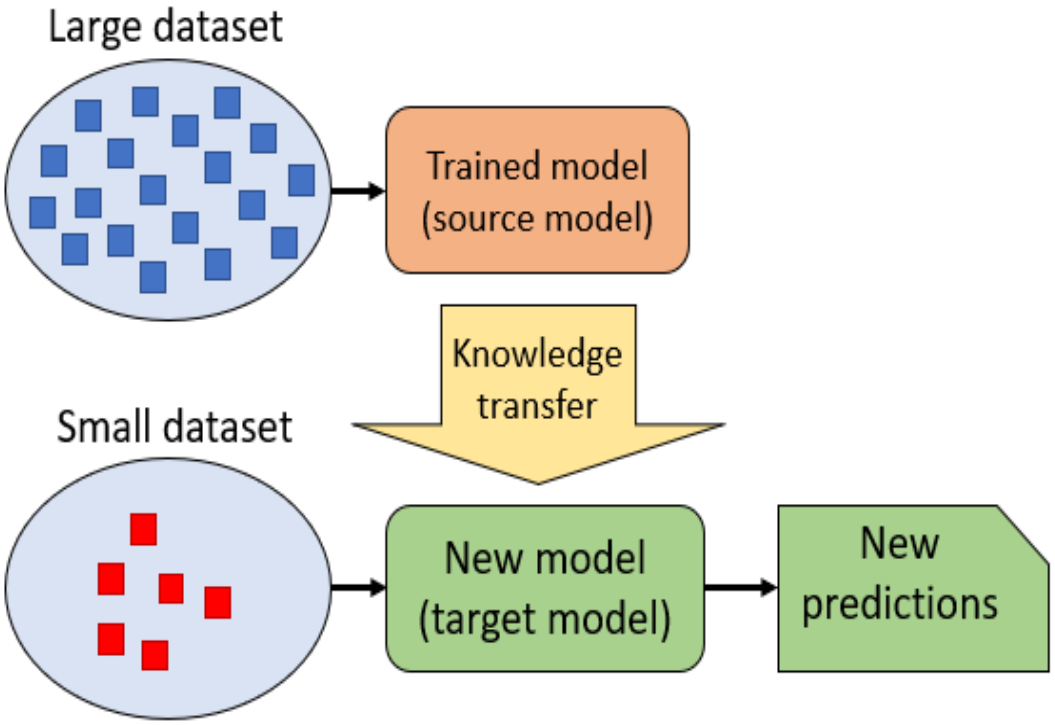

Figure 2.17 Transfer learning process [94]

3

Chapter Three

Proposed System Design & Implementation

Chapter Three

Proposed Model

3.1 Introduction

Object detection on embedded system opens new opportunities for edge computing and IoT applications. By overcoming the challenges of limited resources and leveraging powerful software frameworks, we as users can create innovative solutions that bring the power of machine learning to the edge, enabling real-time object detection in a wide range of scenarios.

In this work, we will find solutions to the blind spot problem in vehicles by taking advantage of embedded devices such as microcontrollers and cameras, along with advanced signal processing and machine learning algorithms. Blind spot detection systems enhance driver awareness and reduce the risk of accidents.

In this chapter, we will explain the propose system and the steps we followed to build the model, in addition to the algorithms used in the training process and even transferring the model to the embedded device.

3.2 Description of the System

A driver's blind spot is the region that is not visible to them when looking through the side and rear mirrors. This thesis has explored the innovative application of tiny machine learning (TinyML) in detecting blind spots using the Arduino Nano 33 BLE and Arduino Portenta H7 , addressing a crucial aspect of automotive safety. By leveraging the power of edge computing and compact hardware, we have demonstrated the feasibility of deploying intelligent blind spot detection systems in resource-constrained environments such as three wheels motorcycle with accuracy in real-time. We will provide a comprehensive description of the model in terms of hardware and framework.

3.2.1 Hardware

Below is a general description of the devices used:

1. **Arduino Nano 33 BLE**

The main challenge in this work was to implement the model on this type of Arduino (as shown Figure 3.1) due to its limited capabilities and resources as shown in tabel 3.1. Since there are various boards, a clear distinction is needed. Nano33 BLE Sense is Nordic Semiconductors' nRF52840, 32-bit ARMCortex-M4 CPU running at 64MHz Equipped with 1MB (nRF52840) CPU Flash Memory. It can be used in a variety of ways, including ML (Machine Learning). As the name of the Nano 33 BLE Sense board suggests, BLE is stands for Bluetooth Low Energits specifications are shown in the following table:

Table 3.1 Specification of Arduino Nano 33 BLE[95]

Specification	Details
Microcontroller	Nordic nRF52840 Cortex-M4F
Operating Voltage	3.3V
Digital I/O Pins	14 (including 6 PWM outputs)
Analog Input Pins	8 (8-bit resolution)
Flash Memory	1MB
SRAM	256KB
Clock Speed	64MHz
Wireless Communication	Bluetooth Low Energy (BLE)
USB Interface	Micro USB
Dimensions	45mm x 18mm
Weight	5g

2. **Arduino Portenta H7**

The Arduino Portenta H7(Figure 3.1) is a powerful board featuring dual-core ARM Cortex-M7 and Cortex-M4 architecture, making it suitable for a wide range of applications including machine vision tasks. With advanced computing capabilities and comprehensive I/O options, it brilliantly meets high-end computing applications. The following table shows the device specifications:

Table 3.2: Specification of Arduino Portenta H7 [96]

Specification	Details
Microcontroller	STM32H747XI dual-core Cortex-M7 @ 480 MHz, Cortex-M4 @ 240 MHz
Operating Voltage	3.3V
Digital I/O Pins	39
Analog Input Pins	16
Flash Memory	16 MB
SRAM	2 MB
Clock Speed	480 MHz (Cortex-M7), 240 MHz (Cortex-M4)
Wireless Communication	Wi-Fi 802.11 a/b/g/n/ac, Bluetooth 5.1
USB Interface	USB-C (host and device)
Dimensions	63.5mm x 25mm
Weight	Approximately 14g

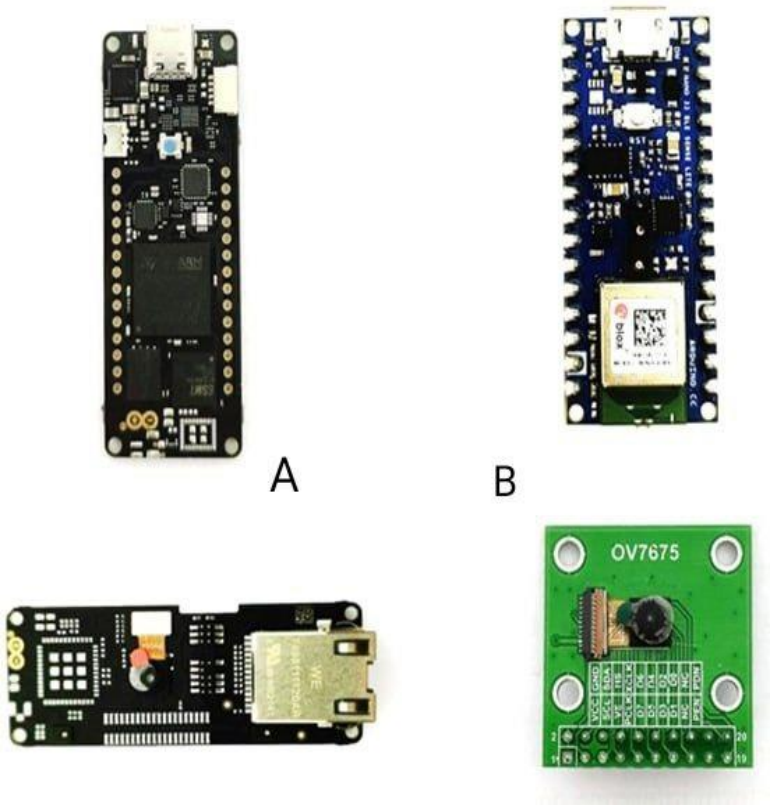

Figure3.1 A- Arduino Portenta H7 with vision shield and B- Arduino Nano 33 BLE with ov7675 camera

The Vision Shield for Portenta H7 is an integrated solution designed for computer vision tasks, featuring Himax HM01B0 image sensor and ST VL53L0X ToF ranging sensor. It connects directly to the Portenta H7, simplifying integration, and is compatible with Arduino IDE for ease of programming. In contrast, the OV7675 Camera Module is a stand-alone module requiring interfacing with microcontrollers such as Arduino nano 33 BLE , offering basic image capture capabilities with its OV7675 CMOS sensor.

3.2.2 Software

In this work, we utilized an online platform called Edge Impulse to develop and deploy machine learning models on edge devices. Edge Impulse provides an accessible and comprehensive environment for building, training, and optimizing machine learning models specifically

tailored for edge applications. The platform simplifies the process of data acquisition, feature extraction, model training, and deployment, making it particularly suitable for rapid prototyping and real-time inferencing on low-power devices.

To enhance the performance of our object detection model, we employed Transfer Learning, a powerful technique that leverages pre-trained models on large datasets to fine-tune and adapt them to specific tasks with smaller, domain-specific datasets. Transfer learning significantly reduces the time and computational resources required for training, while also improving model accuracy and robustness by using learned features from well-established architectures.

3.3 Proposed System

Figure 3.2 show typical workflow for deploying a machine learning model on an embedded device. It starts with data collection and data preprocessing, followed by model selection and training. The next steps involve model optimization and quantization to ensure efficiency on resource-constrained devices, leading to deployment to the embedded device. Finally, the process includes evaluation of the model's performance and device inference for real-time decision-making.

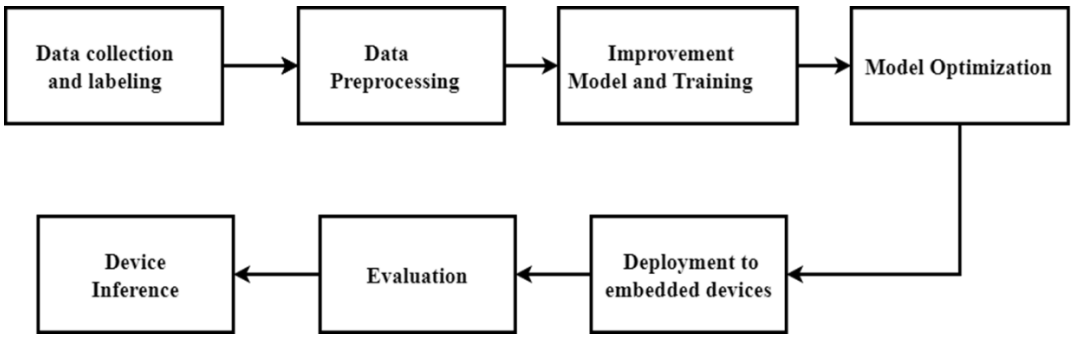

Figure 3.2 Proposed Framework

3.3.1 Dataset

Two types of data were used in this work:

a) **Kaggle** (Figure 3.3) dataset was used.

- Image size and resolution: Image size refers to the dimensions of an image in pixels, usually represented as width x height. In this data type the image was 640 pixels wide and 480 pixels high, indicating a resolution of 640 x 480 pixels. Higher resolution images contain more detail but may require more processing power and storage.
- Object Dimensions: When describing an object within an image, we can specify its dimensions in relation to the size of the image. If we detect the vehicle in the image, for example, we describe its dimensions as occupying approximately 1/4 of the image's width and 1/5 of its height. This gives an idea of the size of the object within the image. The size of objects relative to the image dimensions can greatly affect object detection algorithms. Larger objects may be easier to detect, especially if they take up a large portion of the image. However, smaller objects or objects with lower contrast relative to the background may be difficult to accurately detect.
- The presence of more than one vehicle in the image was also taken into consideration, as well as the event that the vehicles overlapped with each other and the colors of the vehicles had different backgrounds.

Figure 3.3 Sample of Kaggle dataset

b**) Live images were collected** Video data captured in Karbala, Iraq, provides a vivid depiction of urban life, characterized by busy streets, diverse vehicles, and many colors (Figure 3.4). Segmenting the video into images reveals the complexities of the scene, including overlapping vehicles and the challenge of distinguishing between different colors. We captured images at different times of day and night, as well as at different angles, and mounted the camera on the vehicle's side mirror to obtain data where the three-wheeled vehicle was actually in the blind spot area.

Figure 3.4 Dataset collected from the streets of Iraq

Although the number of data is small, we only train the last layer of the network layer because we are using a previously trained network. This process is called transfer learning.

i.Data Splitting

Data sets are then typically divided into three main sub-sets: training, validation, and test sets. Each subgroup serves a specific purpose in developing and evaluating machine learning models. The types are as follows:

a) Training Set

The training set, which is the largest part of the dataset, is dedicated to training the ML model and in this work it amounts to 80% of the total data used, shown figure 3.5. It includes labeled examples used to teach the model basic patterns and relationships within the data. Ensuring that the training set represents the overall distribution of the data is critical for the model to effectively learn generalizable patterns.

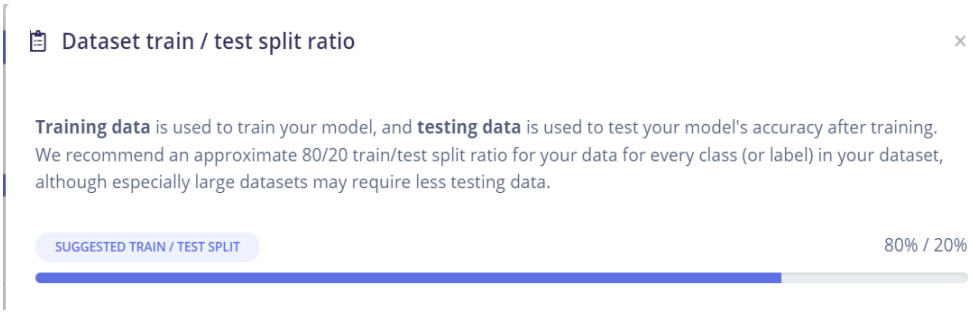

Figure 3.5 Ratio of training data to test data

b) Test Set

Test Set is a completely independent subset of the dataset, merely serves to assess the completed performance of the trained model. It serves as a proxy for real-world model performance on new, unseen data. Keeping the test set separate from the training and validation sets ensures an unbiased assessment of the model's generalization ability. Once the model is trained and fine-tuned using training and validation sets. To give an idea of practical effectiveness, the performance is evaluated by applying the model to the test set. In this work, its percentage was 20% of the total data, as shown in the Figure 3.5.

c) Validation Set

Validation Set is a smaller subset of the dataset only 15% of training set, serves the purpose of tuning hyperparameters and evaluating training model performance. It acts as an independent evaluation set, assessing how well the model generalizes to unseen data and helping prevent overfitting. Hyperparameters like learning rate, regularization strength, or network architecture are adjusted based on the model's performance on the validation set.

ii. Labeling

To keep the data interoperability with other models, labelling the data is required before to beginning a neural network training. While the detection technique will involve marking the centers of the objects it

wishes to detect, the bounding boxes will be used as input for training the image. Following that, all of the data is categorized and the labels which are displayed in Figure 3.6 are given the name "Tuktuk."

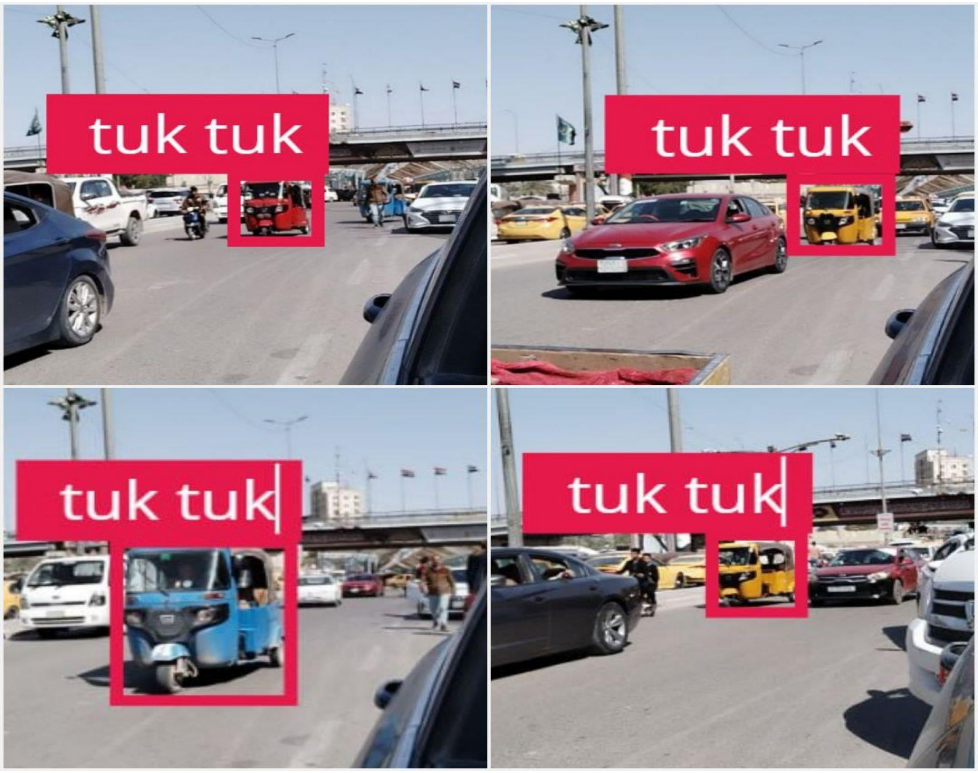

Figure 3.6 Labeling proses

3.3.2 Data preprocessing

a) **Image resizing:** it involves resizing the collected images to a uniform resolution suitable for model training and inference while balancing computational efficiency and information preservation. We trained the model on several sizes starting from 70*70 to 360*360 to get the best results.

b) **Image color selection:** In this part, the appropriate image color is also chosen according to the capabilities of the device used, whether Grayscale or RGB (shown Table 3.4). The choice between grayscale and RGB images can have a significant impact on memory usage, particularly in the context of machine learning models where memory

resources may be limited, such as on edge devices or embedded systems.

Table 3.3 Differences between RGB and Grayscale

Aspect	RGB	Grayscale
Color Model	Uses three primary colors (Red, Green, Blue)	Uses a single channel representing intensity
Components	Consists of three channels (R, G, B)	Consists of a single channel
Color Depth	Each channel typically has 8 bits (0-255)	Each pixel typically has 8 bits (0-255)
Memory Usage	Requires more memory due to multiple channels	Requires less memory compared to RGB
Computational Efficiency	More computationally intensive due to multiple channels	More computationally efficient due to single channel

c) Feature explorer: The Visual Features Explorer can be employed to quickly identify outliers or improperly classified data. The latter facilitates the recognition of unclassified data. The data explorer first endeavours to extract the suitable characteristics from the data by analysing the signals and integrating them into a neural network. Subsequently, it utilises a dimensionality reduction mechanism to convert these characteristics into a two-dimensional domain. This provides a thorough overview of the entire dataset in a single glance.

This visualization efficiently assesses the degree of separation in the data. In this case, the Feature Explorer displays data points that differ greatly from the main cluster in the dataset. This implies the potential for misnaming of data.

Figure 3.7 displays a portion of the dataset utilized in the study, whereas (Figure3.7 A) presents the data obtained from the collected dataset. The quantity of image in the dataset will have a significant impact. Despite the presence of outliers, the explorer determined that they were acceptable after conducting a thorough examination. The variation in this dispersion is probably caused by the diverse composition of the background. Derived from (Figure3.7 B), a subset of the Kaggel dataset. Based on the graphic, our initial expectation is that the combined data will yield superior results compared to the Kaggel data. The rationale behind this is that the photographs he gathered exhibit a higher degree of similarity and consistency, as they originate from comparable streets and surroundings, in contrast to Kaggel's dataset.

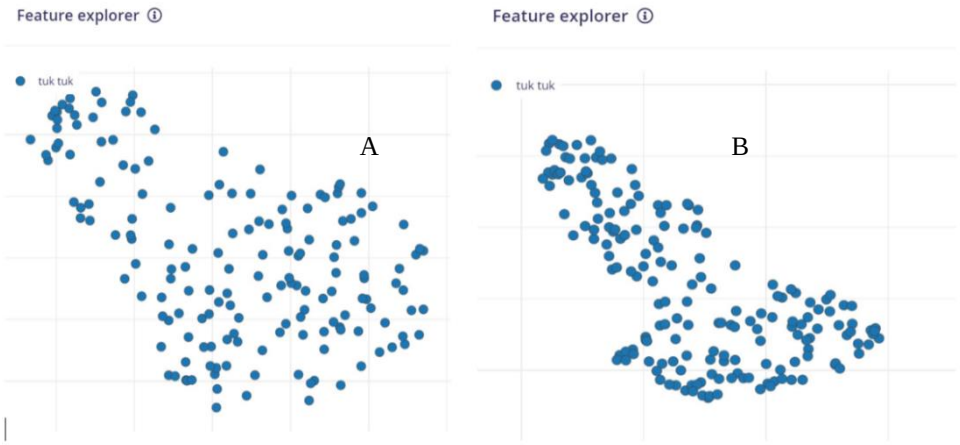

Figure 3.7 Feature explorer (A) Kaggle Dataset (B) Collected Dataset

3.3.3 Improvement model and Training

Train a machine learning model using pre-processed data. Edge Impulse provides tools for building and training different types of models. Three algorithms were used in this work and compared between them to find out the best algorithm that matches the capabilities of the microcontroller. Different TinyML algorithms such as YOLOv5,

MobileNetV2 SSD FPN-Lite, and FOMO for object detection tasks, taking into account the complexity and accuracy of the model.

1- FOMO: This model utilises MobileNetV2 (alpha 0.1) to detect objects and segment a picture into a grid, distinguishing between the background and objects of interest. The purpose of these models is to have a size of less than 100KB and be capable of accepting grayscale or RGB input at any resolution. The FOMO object detection model developed by Edge Impulse is highly efficient in accurately categorising things and has a little computational footprint, making it suitable for running on microcontrollers (MCUs).

Input layer (9,216 features) for the capabilities of the Arduino Nano33 BLE include image size of 96*96 and grayscale. In contrast, the Input layer (307,200 features) has the capabilities of the Arduino Portenta H7, represented by an image size of 320 * 320 and RGB.

- **FOMO Architecture**

Table3.4 outlining the FOMO model architecture tailored for an input image size of 96x96, detailing the key layers and their functions:

Table 3.4 FOMO architecture tailored for an input image size of 96x96

Layer Type	Description	Parameters	Output Shape
Input Layer	Accepts the input image	Image size: 96x96x3	(96, 96, 3)
Convolutional Layer	Initial feature extraction	Filters: 32, Kernel: 3x3, Stride: 2, Padding: Same	(48, 48, 32)
Batch Normalization	Normalizes the output of the previous layer	-	(48, 48, 32)
ReLU Activation	Applies ReLU activation function	-	(48, 48, 32)
Depthwise Separable Convolution	Reduces computational complexity	Filters: 64, Kernel: 3x3, Stride: 2, Padding: Same	(24, 24, 64)

Layer Type	Description	Parameters	Output Shape
Batch Normalization	Normalizes the output of the previous layer	-	(24, 24, 64)
ReLU Activation	Applies ReLU activation function	-	(24, 24, 64)
Convolutional Layer	Further feature extraction	Filters: 128, Kernel: 3x3, Stride: 2, Padding: Same	(12, 12, 128)
Batch Normalization	Normalizes the output of the previous layer	-	(12, 12, 128)
ReLU Activation	Applies ReLU activation function	-	(12, 12, 128)
Depthwise Separable Convolution	Reduces computational complexity	Filters: 256, Kernel: 3x3, Stride: 2, Padding: Same	(6, 6, 256)
Batch Normalization	Normalizes the output of the previous layer	-	(6, 6, 256)
ReLU Activation	Applies ReLU activation function	-	(6, 6, 256)
Convolutional Layer	Produces the final feature map	Filters: 512, Kernel: 3x3, Stride: 1, Padding: Same	(6, 6, 512)
Batch Normalization	Normalizes the output of the previous layer	-	(6, 6, 512)
ReLU Activation	Applies ReLU activation function	-	(6, 6, 512)
Convolutional Layer (Heatmap)	Generates heatmaps for object presence detection	Filters: Number of classes, Kernel: 1x1, Stride: 1, Padding: Same	(6, 6, Number of Classes)
Softmax Layer	Converts logits into probabilities for classification.	-	(6, 6, Number of Classes)

i. **Input Layer**: Receives the input image, resized to 96x96 pixels with 3 color channels.

ii. **Convolutional Layers**: The initial convolutional layer extracts basic features, followed by depthwise separable convolutions to

reduce computational complexity while maintaining feature extraction.

iii. **Batch Normalization**: normalizes outputs of convolutional layers to stabilize and accelerate training.

iv. **ReLU Activation**: Introduces non-linearity, enabling the network to learn complex patterns.

v. **Depthwise Separable Convolution**: reduces the number of parameters and computations, maintaining efficiency while still extracting meaningful features.

vi. **Final Convolutional Layer**: generates the final feature map for heatmap creation.

vii. **Heatmap Layer**: A convolutional layer with a number of filters equal to the number of classes, predicting heatmaps indicating the presence of objects.

viii. **Softmax layer**: takes the final output from the previous layer (the heatmap generated by the last convolutional layer) and normalizes the values into a probability distribution across the number of classes. This is typically the last layer in a classification network, where each value represents the probability of each class.

2. MobileNetV2 SSD FPN-Lite: The provided model is a pre-trained object recognition model that is specifically designed to identify and locate a maximum of 10 things within an image. It produces a bounding box for each discovered object. The system is capable of accepting an RGB input with a resolution of 320x320 pixels. The input layer consists of 307,200 features. Table 3.5 it is a summary of the layers used to build this algorithm.

Table 3.5 MobileNetV2 SSD FPN-Lite architecture tailored for an input image size 320*320

Layer Type	Description	Parameters	Output Shape
Input Layer	Accepts input image for processing.	Shape: (Height, Width, Channels)	(320, 320, 3)
Conv2D Layer	Initial convolutional layer for feature extraction.	Filters: 32, Kernel: 3x3, Stride: 2, Padding: Same	(160, 160, 32)
Batch Normalization	Normalizes the output of the previous layer.	-	(160, 160, 32)
ReLU6 Activation	Applies ReLU6 activation function.	-	(160, 160, 32)
Inverted Residual Block (Bottleneck)	Efficient block for feature extraction and reducing dimensions.	Varies (e.g., Filters: 16, Expansion: 6x, Kernel: 3x3)	Varies (e.g., (80, 80, 16))
Depthwise Separable Convolution	Further feature extraction with reduced computation.	Filters: Varies, Kernel: 3x3, Stride: 1, Padding: Same	Varies (e.g., (80, 80, 96))
Feature Pyramid Network (FPN-Lite)	Combines multi-scale feature maps for better detection of objects of various sizes.	Uses 1x1 and 3x3 convolutions to merge feature maps.	Varies (e.g., (40, 40, 256))
Convolutional Layer (SSD Head)	Predicts object locations and class probabilities from feature maps.	Filters: Varies, Kernel: 3x3, Stride: 1, Padding: Same	Varies (e.g., (40, 40, 24))
Batch Normalization	Normalizes the output of the previous SSD head layer.	-	Varies (e.g., (40, 40, 24))
ReLU6 Activation	Applies ReLU6 activation function.	-	Varies (e.g., (40, 40, 24))
Anchor Box Layer	Generates anchor boxes for different scales and aspect ratios.	Configured per feature map level.	Varies
Classification Convolutional Layer	Classifies each anchor box as a specific class or background.	Filters: Number of Classes x Number of Anchors	Varies (e.g., (40, 40, 8732))
Bounding Box Regression Layer	Regresses the offsets for each anchor box to fit the bounding boxes of detected objects.	Filters: 4 x Number of Anchors	Varies (e.g., (40, 40, 8732))
Softmax Layer	Converts classification logits into probabilities for object classes.	-	(Number of Anchors, Number of Classes)

3- **YOLOv5:** The transfer learning model utilises the yolov5n.pt weights and is capable of handling RGB input at any resolution, albeit it only accepts square images. The input layer consists of 76,800 features.

The YOLOv5 model utilizes Focus, CSP (Cross Stage Partial), SPP (Spatial Pyramid Pooling), and PANet (Path Aggregation Network) layers, which enhance feature extraction, reduce model size, and improve detection accuracy, especially for small and large objects. The detection heads in YOLOv5 are responsible for generating object detection outputs at different scales, allowing the model to detect objects of varying sizes. Table3.6 provides a comprehensive overview of the YOLOv5 architecture, capturing the essential layers, their roles, parameters, and output shapes in the network.

Table 3.6 YOLOv5 architecture tailored for an input image size 320*320

Layer Type	Description	Parameters	Output Shape
Input Layer	Accepts the input image for processing.	Shape: (Height, Width, Channels)	(320, 320, 3)
Focus Layer	Slices the input image into 4 and concatenates them depth-wise to enhance small object detection.	Kernel: 3x3, Stride: 1, Padding: Same	(160, 160, 32)
Conv2D Layer	Convolutional layer for initial feature extraction.	Filters: 64, Kernel: 3x3, Stride: 2, Padding: Same	(80, 80, 64)
Batch Normalization	Normalizes the output of the convolutional layer.	-	(80, 80, 64)
Leaky ReLU Activation	Applies Leaky ReLU activation function.	-	(80, 80, 64)

Layer Type	Description	Parameters	Output Shape
CSP (Cross Stage Partial) Block	Enhances gradient flow and reduces model size for deeper layers with improved performance.	Multiple Bottleneck Blocks, varies	Varies (e.g., (80, 80, 128))
SPP (Spatial Pyramid Pooling) Layer	Aggregates global context with max-pooling to improve receptive field and object localization.	Kernel: 5x5, 9x9, 13x13	(40, 40, 256)
PANet (Path Aggregation Network)	Improves information flow for small and large objects by fusing low-level and high-level features.	Concat and Upsample Layers	Varies (e.g., (20, 20, 512))
Convolutional Layer	Generates feature maps for object detection with various anchor boxes.	Filters: 256, Kernel: 3x3, Stride: 1, Padding: Same	Varies (e.g., (20, 20, 256))
Batch Normalization	Normalizes the output of the previous convolutional layer.	-	Varies (e.g., (20, 20, 256))
Leaky ReLU Activation	Applies Leaky ReLU activation function.	-	Varies (e.g., (20, 20, 256))
Detection Head	Generates final bounding boxes and class probabilities.	Uses anchors specific to different scales	Varies (e.g., (10, 10, 3, (5 + Number of Classes)))
Anchor Box Layer	Generates anchor boxes at different scales for object localization and bounding box regression.	Configured per feature map level	Varies
Convolutional Layer (Final)	Refines predictions for object locations and classes for each anchor box.	Filters: Number of Classes x Number of Anchors	Varies (e.g., (10, 10, Number of Classes x 3))

Table 3.7 is a comparison of YOLOv5, MobileNetV2 SSD FPN-Lite, and FOMO algorithms, focusing on their architecture, efficiency, suitability for real-time applications, and other characteristics.

Table 3.7Comparison of YOLOv5, MobileNetV2 SSD FPN-Lite, and FOMO algorithms

Feature	YOLOv5	MobileNetV2 SSD FPN-Lite	FOMO
Architecture	Single-stage, CNN backbone with neck and head for prediction	MobileNetV2 CNN, SSD for multi-scale detection, FPN for feature enhancement	Simplified detection using heatmaps
Speed and Accuracy	High speed and good accuracy for real-time applications	Good balance of speed and accuracy for mobile and edge devices	Extremely low latency and power consumption, suitable for highly constrained devices
Model Size	Multiple versions (s, m, l, x) to suit different hardware capabilities	Lightweight, ideal for mobile and edge deployment	Ultra-compact, designed for microcontrollers and embedded systems
Target Applications	Real-time object detection in moderate to high resource environments	Mobile and edge devices with limited computational resources	TinyML applications where presence and approximate location detection is sufficient
Optimization	Process image in one forward pass for efficiency	Uses depthwise separable convolutions for efficiency, combines multi-scale feature maps	Simplifies object detection problem for extreme resource efficiency
Deployment	Suitable for GPUs, powerful CPUs, and edge devices	Optimized for mobile phones, tablets, and edge devices	Designed for microcontrollers, IoT devices, and embedded systems

3.3.4 Model Optimization

In TinyML, model optimization is paramount due to the stringent constraints of resource-constrained edge devices like microcontrollers and IoT devices. These models need to be optimized for size, speed, and energy efficiency to fit within limited memory and processing power while ensuring real-time inference and preserving battery life. Optimized models are more robust to noisy environments, scalable for deployment across edge devices, and offer enhanced security and privacy protections.

The Edge Impulse platform offers two optimization models to adapt the model for execution on devices with limited resources, such as Arduiszno. The "int8" quantum models trade off accuracy for lower memory usage and enhanced efficiency, making them ideal for deployment on devices with limited resources: "int8" and "float32", shown Table3.8. However, unoptimized "float32" models maintain a high level of accuracy but necessitate greater memory and processing resources. As a result, they 3are better suited for the development and training stages where precision is of utmost importance.

Table 3.8 Model Optimization

Quantized (int8)	Unoptimized (float32)
Int8 quantization significantly reduces model size and memory footprint, making it more efficient for deployment on resource-constrained devices such as microcontrollers.	Float32 models offer higher accuracy and precision compared to quantized models but may be less efficient for deployment on devices with limited computational capabilities.
Int8 quantization also improves inference speed and energy efficiency by reducing the computational cost of numerical operations.	Float32 models are commonly used during model development and training to achieve the highest possible accuracy before optimization for deployment.

3.3.5 Deployment to embedded devices

It is time to convert the optimized model to the appropriate format compatible with TinyML frameworks after optimizing it and making the memory suitable for the capabilities of the device. At this stage, the modified model is used on microcontroller or embedded device platforms. Arduino Portenta H7 and Arduino nano33 BLE were used in this work.

We export the trained Edge Impulse model in a format compatible with Arduino hardware. The exported model was built into both devices using the Arduino IDE. We next combine the TensorFlow Lite and Edge Impulse Arduino libraries. After then, the device's flash memory held the exported model file. The model's performance on the device should lastly be monitored and, if needed, optimized.

We further alter model structure, adjust hyperparameters, or integrate new data to improve model accuracy and efficiency in real-world scenarios. These steps let Arduino Nano 33 BLE devices and Arduino Portenta H7 devices to deploy machine learning models trained with Edge Impulse for a range of embedded and Internet of Things applications.

3.3.6 Evaluation

- **Inference Time Calculation:** Edge Impulse enables real-time inference on embedded devices, therefore deployment of models in resource-constrained settings depends on an understanding of inference time. Complexity of the model, hardware platform, and any deployment-related optimizations all affect inference time. Regarding blind spot detection, the success of the system is mostly dependent on its reasoning time.
- **Validation Set Performance:** To evaluate the model's post-training performance, use a validation dataset. Accuracy,

precision, recall, F1 score, and other metrics are computed using the model's predictions on the validation set in relation to the ground truth labels.

3.3.7 Device Inference

Testing an object detection model on peripheral devices such as the Arduino Nano 33 BLE or Arduino Portenta H7 (as shown in Figure 3.8) involves deploying the trained model on the device and evaluating its performance in real-world scenarios.

Figure 3.8 (A.) Arduino Portenta H7 test (B.) Arduino Nano 33 BLE test

Flowchart in Figure 3.9 a continuous, real-time loop where a microcontroller, integrated with a camera, captures images, processes them, and performs object detection to identify three-wheeled vehicles. The detection results are marked visually, and the system is designed to operate repeatedly, making it suitable for applications like blind spot detection.

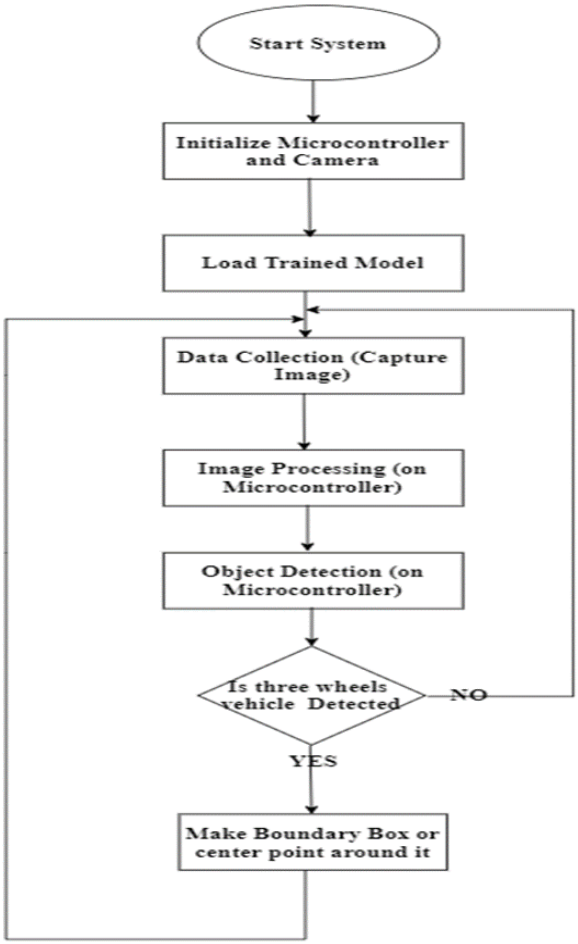

Figure 3.9 flowchart describes the process for a real-time object detection system using a microcontroller and camera

4

Chapter Four

Results & Discussion

Chapter Four

Results and Discussions

4.1 Introduction

This chapter provides a thorough examination and assessment of the effectiveness of the proposed approach in detecting blind spots in embedded vehicles. The process entails a sequence of meticulously planned tests, each aimed at assessing various facets of the model's capabilities. The primary objective of this chapter is to showcase and evaluate the numerical outcomes derived from the model's effectiveness in different benchmarks and experimental scenarios. The chapter begins by presenting a thorough examination of methodology, a validation dataset, and specific evaluation criteria. The chapter subsequently showcases the numerical findings using tables and graphs, facilitating a comparative evaluation of the model's efficacy in relation to reference models.

4.2 Performance of lite object detection algorithms

In order to explore the trade-off between accuracy, inference time, and memory, we trained our model on two devices (Arduino Portenta H7 and Arduino Nano 33 BLE) each device was trained on different input sizes of dataset (shown 3.3.1 in chapter 3), 70 * 70, 96 * 96, 120 * 120, and 320 * 320. These sizes were chosen so that the model fit the target device. While maintaining a reasonable portion of image resolution.

In this work, YOLOv5, FOMO, and MobileNetV2 SSD FPN-Lite algorithms are compared and applied to the two devices used during parameter tuning: learning rate = 0.004, validation set size = 20, and batch size = 20, as shown in table 4.1 summary of the behavior of the algorithms used. The accuracy of the work is important, but the most crucial aspect is

the efficient use of RAM and flash memory, ensuring they are compatible with the capabilities of the Arduino.

Table 4.1 General comparison of algorithm behavior on microcontrollers used

Algorithm	**Arduino Portinta H7**	**Arduino Nano 33 BLE**
SSD FPN-Lite	✗	✗
YOLOv5	✓	✗
FOMO	✓	✓

4.2.1 Single Shot Detector- Feature Pyramid Network (SSD FPN-Lite)

After training the data on the SSD algorithm, it showed unacceptable results for the two devices, as this algorithm restricts us to an image size of 320*320 and RGB colors, as it requires a device with higher resources than the one that was used. As we note in Table4.2, this algorithm is not compatible with the capabilities of the Arduino, whether in terms of memory or inference time. The choice of a particular input size to train a model such as SSD-FPN (Single-shot Multi-Box Detector with Feature Hierarchical Network) with 320 x 320 RGB resolution can be due to the architecture design. This size could have been chosen to balance model complexity with accuracy and computational efficiency.

Table 4.2 Behavior of the SSD algorithm on the dataset and devices used

Algorithm	Image Size	Image color	Inferencing Time(Arduino Nano 33ble)	Inferencing Time(Portenta H7)	Flash Usage	Validation Set Accuracy	Test Accuracy
Kaggle Dataset							
SSD FPN-Lite	320x320	RGB	960945 ms	18225ms	11.0MB	81.9%	77.5%
Collected spot dataset							
SSD FPN-Lite	320x320	RGB	953015 ms	17198 ms	11.0MB	71.5%	77.5%

4.2.2 You only look once (YOLOv5)

Using YOLOv5 with an Arduino Nano 33 BLE and Portenta H7, flash usage is about 1.8MB, as shown in Table 4.3. This flash usage refers to the amount of program memory required to store the model and its associated code for inference.

Given this flash usage and inference time results:

i. **Arduino Nano 33 BLE:** Due to limited flash memory of 1MB, using YOLOv5 on an Arduino Nano 33 BLE it cannot be implemented. Flash usage of 1.8MB is very high for this microcontroller, and it can be difficult to accommodate both the model and the necessary inference code within the available memory. In addition, at the time of inference, numbers appeared that were impractical and were not useful for detecting blind spots in vehicles.

ii. **Portenta H7:** Using 1.8MB of flash is within the capabilities of the Portenta H7, which is a more powerful microcontroller compared to the Nano 33 BLE. The Portenta H7 has 16MB available flash memory, making it more suitable for running complex models such as YOLOv5. This algorithm has the ability to detect during the day and night, as shown in the Figure 4.1 and Figure 4.2.

The size of images directly impacts RAM usage because larger images require more memory for storage and processing. When an image is captured or loaded, it is temporarily stored in RAM for quick access during processing. The amount of memory needed depends on the image's resolution (width x height) and the number of channels (e.g., RGB channels require 3 bytes per pixel while Grayscale require only 1 bytes per pixel).

In other words, smaller images have fewer pixels, meaning less data needs to be stored in memory. As shown in Table4.3, an image of 320x320 size will take up more RAM compared to an image of 96x96 size.

Table 4.3 Behavior of the YOlOv5 algorithm on the dataset and devices used

Algorithm	**Image Size**	**Image color**	**Inferencing Time(Arduino Nano 33ble)**	**Inferencing Time (Portenta H7)**	**RAM Usage**	**Flash Usage**	**Validation Set Accuracy**	**Test Accuracy**
Kaggle Dataset								
YOLOv5	320x320	RGB	61498 ms	4115 ms	1.3MB	1.8MB	88.0%	58.82%
YOLOv5	96 x 96	RGB	4028ms	531 ms	222.1KB	1.8MB	63.9%	37.93%
Collected dataset								
YOLOv5	**320x320**	**RGB**	**9297 ms**	**3668ms**	**1.3MB**	**1.8MB**	**89.0%**	**75.00%**
YOLOv5	96x96	RGB	8830 ms	433 ms	220.8KB	1.8MB	74.4%	56.25%

Figure 4.1 Implementing the YOLOv5 algorithm on an Arduino PortentaH7 during Morning-time

Figure 4.2 Real-time results of work using portent H7 with YOLOv5 algorithm during night-time

iii. Mean Average Precision (mAP)

As shown in Figure 4.3, the mean Average Precision (mAP) of the model showed remarkable improvement throughout the training process. Initially, the mAP was very low at 0.00969, indicating poor overall performance in detecting and localizing objects across different categories. However, as training progressed and the model underwent iterative adjustments such as enhanced feature extraction, hyperparameter tuning, and data augmentation, there was a significant enhancement in its ability to accurately detect and localize objects. By the end of the training period, the mAP had increased dramatically to 0.796. This substantial improvement demonstrated the model's enhanced reliability and suitability for practical applications, reflecting a high overall detection quality with relatively low rates of false positives and false negatives.

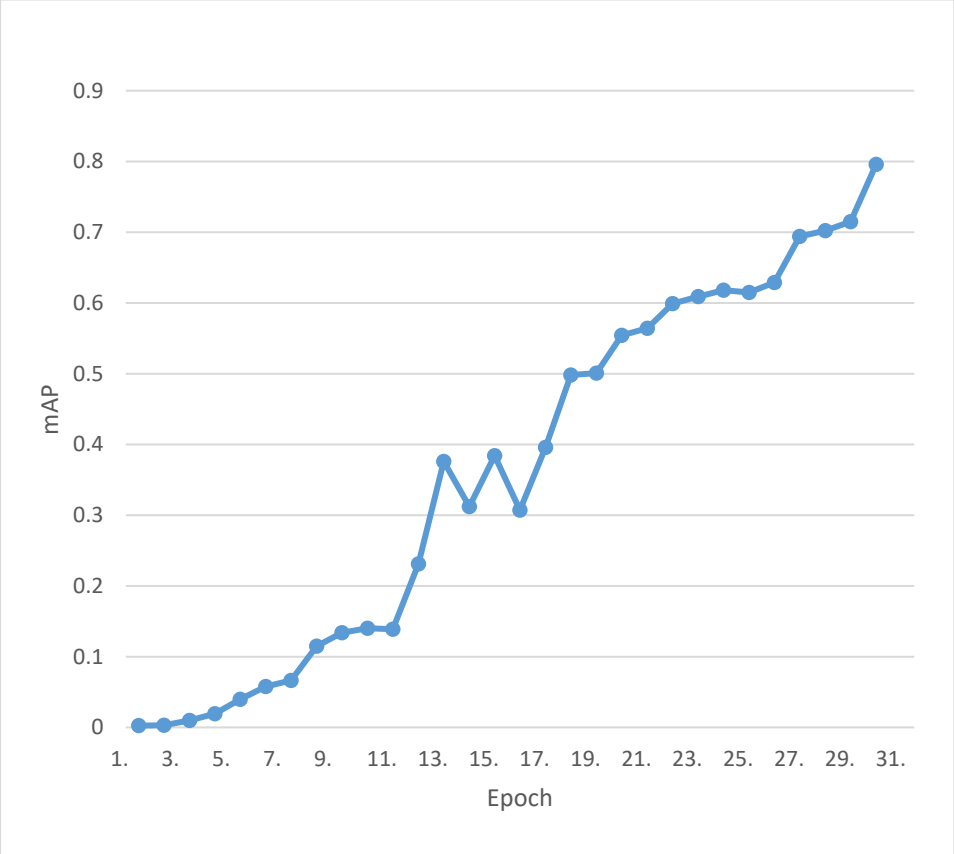

Figure4.3. mAP during Training in YOLOv5

iv. Precision and Recall

Over the course of training, the precision and recall of the model showed significant improvement. As shown in Figure 4.4, the model's performance was extremely poor, with a precision of 0.00104 and a recall of 0.109. This indicated that almost all positive predictions were incorrect, resulting in a very high number of false positives, and the model detected only about 10.9% of the actual objects, resulting in a high number of false negatives. As training progressed, iterative adjustments and optimizations were applied to the model, such as enhanced feature extraction, hyperparameter tuning, and data augmentation. These efforts gradually improved the model's accuracy in predicting positive instances and detecting actual objects. By the end of the training period, the precision had increased dramatically to 0.88, meaning 88% of the model's positive predictions were correct, while the recall had improved to 0.89, indicating that the model was now detecting 89% of the actual objects. This significant enhancement in both metrics demonstrated the model's improved reliability and suitability for practical applications, reflecting a well-balanced performance with relatively low rates of false positives and false negatives.

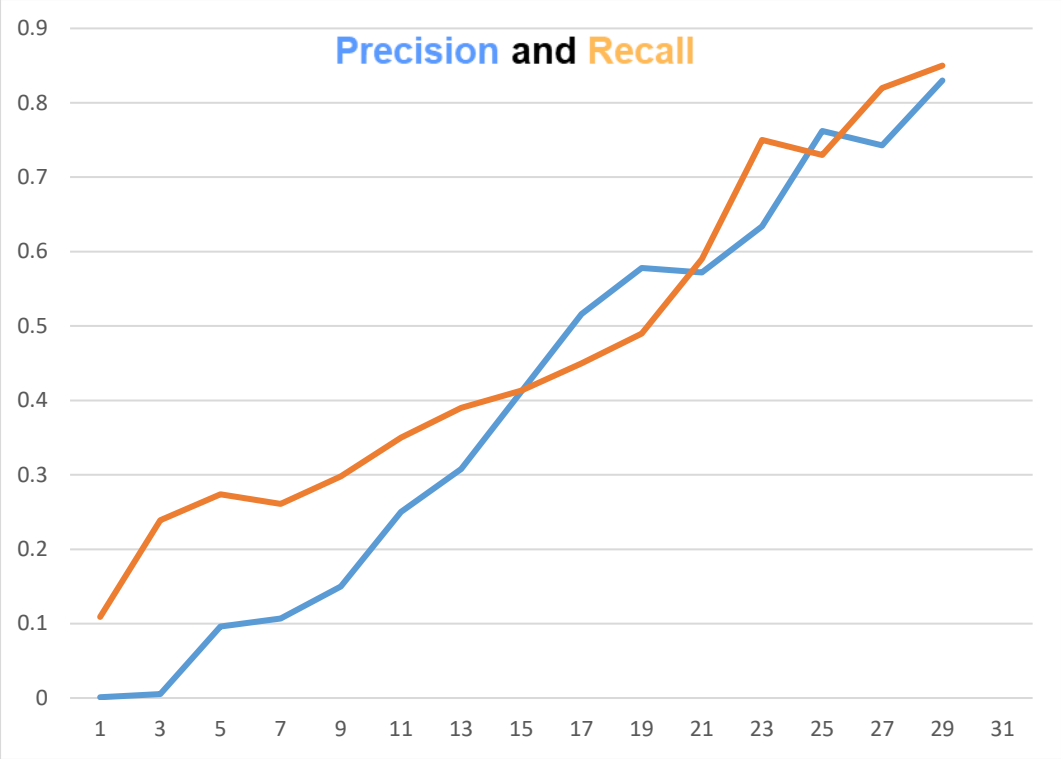

Figure 4.4 Precision and Recall during Training in YOLOv5

v. Confusion Matrix

Table4.4 Confusion Matrix of YOLOv5 algorithm

	BACKGROUND	**TUK TUK**
BACKGROUND	100%	0%
TUK TUK	11.0%	89.0%

1. Actual background, predicted background (100%): when the actual class was "background", the model correctly predicted it as "background" 100% of the time. this means the model perfectly identifies instances of "background".
2. Actual background, predicted Tuktuk (0%): when the actual class was "background", the model never incorrectly predicted it as "Tuktuk".

3. Actual Tuktuk, predicted background (11%): when the actual class was "Tuktuk", the model incorrectly predicted it as "background" 11% of the time. this indicates that the model sometimes struggles to distinguish "Tuktuk" from "background".
4. Actual Tuktuk, predicted Tuktuk (89%): when the actual class was "Tuktuk", the model correctly predicted it as "Tuktuk" 98% of the time.
5. F1 Score:

   -background (1.00): the f1 score for the "background" class is 1.00, indicating perfect precision and recall for this class.

   -Tuktuk (0.87): the f1 score for the "Tuktuk" class is 0.87, which shows good performance but with more errors compared to the "background" class.

4.2.3 Fast Objects More Objects (FOMO)

While both Arduino Nano 33 BLE and Arduino Portenta H7 can support the implementation of the FOMO algorithm, their performance differs significantly due to differences in processing power and memory. Arduino Nano 33 BLE is suitable for lightweight object detection tasks with modest performance requirements, while Arduino PortentaH7 offers higher performance and is capable of handling more complex object detection tasks with greater accuracy.

Table 4.5 Behavior of the FOMO algorithm on the dataset and devices used with Quantized (int8) models

Algorithm	Image Size	Image color	Inferencing Time(Arduino Nano 33ble)	Inferencing Time (Portenta H7)	RAM Usage	Flash Usage	Validation Set Accuracy	Test Accuracy
Kaggle Dataset								
FOMO	96x96	RGB	765 ms	40ms	235.5KB	64.2KB	84.1%	82.35%
FOMO	96x96	Grayscale	537 ms	38ms	235.3KB	63.9KB	82.1%	78.57%
FOMO	70x70	Grayscale	386 ms	30ms	136.5KB	63.7KB	71.7%	64.29%
Collected dataset								
FOMO	96x96	RGB	644 ms	41 ms	123.9KB	76.6KB	92.5%	75.86%
FOMO	96x96	Grayscale	469 ms	36 ms.	123.8KB	76.4KB	91.1%	75.86%
FOMO	**320X320**	**RGB**	**13048 ms**	**484 ms**	**280.6KB**	**91.5KB**	**94.6%**	**96.55%**
FOMO	120X120	RGB	1529 ms	75 ms	182.0KB	76.6KB	91.7%	82.76%

As shown in table4.6, the results indicate that the use of Unoptimized (float32) models as opposed to Quantized (int8) models negatively impacts RAM usage, flash memory, and inference time. Float32 models use 32-bit floating-point representations for weights and activations, which require significantly more memory compared to the 8-bit integer (int8) representations used in quantized models. This increased memory requirement results in higher RAM usage during model inference, as more space is needed to store and process the larger floating-point numbers.

Moreover, the inference time is significantly increased when using Unoptimized (float32) models. Float32 computations are more

computationally intensive and require more processing power than int8 computations. This increased computational demand results in slower inference times, as the processor takes longer to perform the necessary calculations. The combination of higher RAM usage, increased flash memory consumption, and slower inference times can be particularly challenging for embedded systems and devices with limited resources, highlighting the efficiency of quantization in reducing memory demands and speeding up inference while maintaining acceptable levels of model performance.

Table 4.6 Behavior of the FOMO algorithm on the dataset and devices used with Unoptimized (float32) model

Algorithm	Image Size	Image color	Inferencing Time(Arduino Nano 33ble)	Inferencing Time (Portenta H7)	RAM Usage	Flash Usage	Validation Set Accuracy	Test Accuracy
Kaggle Dataset								
FOMO	96x96	Grayscale	6639 ms	120ms	887.2KB	173.0KB	83.3%	68.75%
FOMO	120X120	RGB	11208 ms	203 ms	1.3MB	188.7KB	80.9%	75.00%
FOMO	320X320	RGB	90254 ms	1629 ms	2MB	260.4KB	82%	62.89%
Collected dataset								
FOMO	70x70	Grayscale	2095ms.	238ms	240.9KB	166.9KB	92.0%	79.31%
FOMO	96x96	Grayscale	3904ms	356ms	399.7KB	176.8KB	94.8%	75.86%
FOMO	96x96	RGB	9733ms	476ms	399.7KB	178.0KB	93.9%	79.31%
FOMO	320x320	RGB	67594ms	2120ms	1.9MB	235.5KB	97.6%	93.31%
FOMO	120X120	RGB	11302ms	743ms	1.3MB	182.3KB	93.8%	86.21%

i. Arduino Nano 33 BLE

The performance of the FOMO algorithm on the Arduino Nano 33 BLE will be limited by processing capabilities and memory limitations. These models sacrifice some accuracy to improve speed and are suitable for detecting a limited number of objects in real-time applications. According to Table4.5, by training several models with different image sizes and colors, we were able to arrive at a model suitable for the resources of this device, which is the case in which the image size is 70*70 with Grayscale color, as this case showed better results, as shown in the Figure 4.5.

Figure 4.5 Real-time results of work using the OV7675 camera with FOMO algorithm and image size 70*70

ii. Arduino Portinta H7

It offers much higher processing power and memory compared to the Arduino Nano 33 BLE, making it suitable for more computationally intensive tasks. The FOMO algorithm will benefit from the increased processing power and memory available in the

Arduino Portenta H7. We applied this algorithm with an image size of 320x320 and RGB, this model gave the highest test and validation accuracy 94.6%,96.55%. This model can handle a larger number of objects and more complex scenes compared to those suitable for the Arduino Nano 33 BLE (shown Figure 4.6 and 4.7).

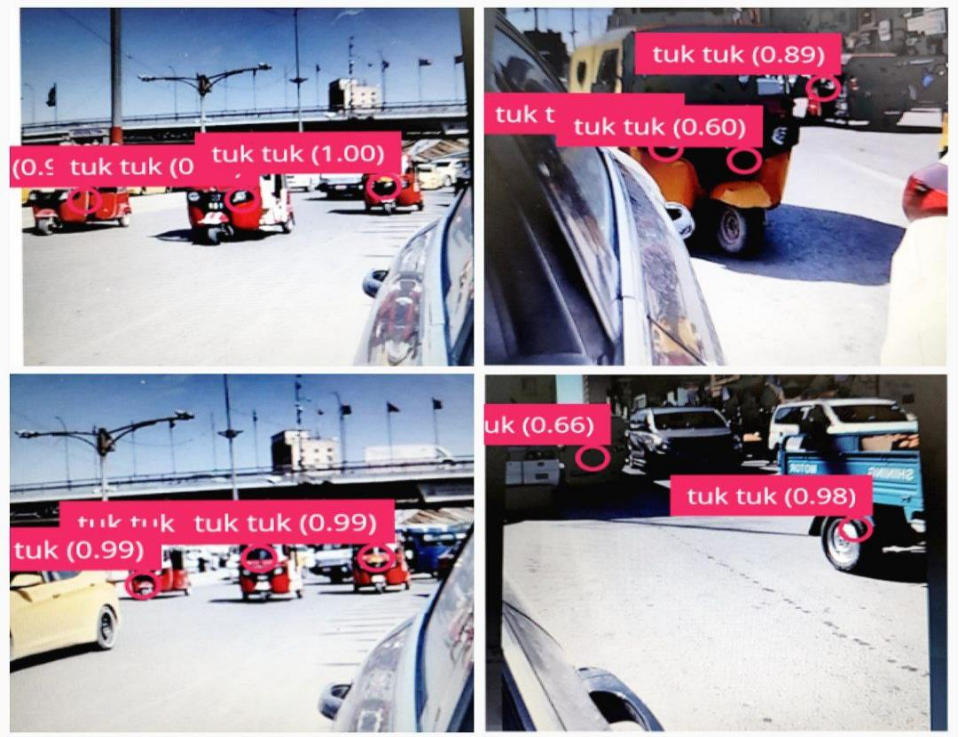

Figure 4.6 Real-time results of work using Portinta H7 with FOMO algorithm and image size 320*320

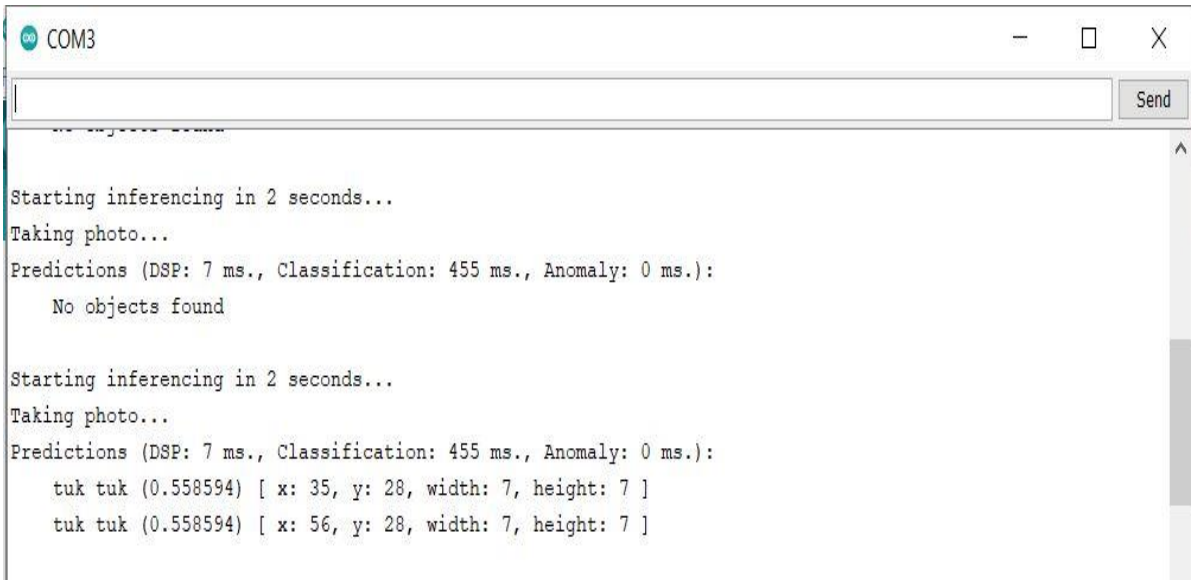

Figure 4.7 Real time result using Arduino IDE

Through real-time testing, it was found that the OV7675 camera was not able to detect at night, while it gave good results during the day. As for the Arduino portent H7 it gave excellent results during the day and night as shown in Figure 4.8.

Figure 4.8 Real-time results of work using portent H7 with FOMO algorithm during night-time

iii. Confusion matrix

Table 4.7 Confusion matrix of FOMO algorithm

	BACKGROUND	TUK TUK
BACKGROUND	100.0%	0.0%
TUK TUK	5.4%	94.6%

1. Actual background, predicted background (100.0%): this means that when the actual class was "background", the model correctly predicted it as "background" 100% of the time.
2. Actual background, predicted Tuktuk (0.0%): this means that when the actual class was "background", the model incorrectly predicted it as "Tuktuk" 0% of the time (i.e., it never made this mistake).
3. Actual Tuktuk, predicted background (5.4%): this indicates that when the actual class was "tuktuk", the model incorrectly predicted it as "background" 8.3% of the time.
4. actual Tuktuk, predicted Tuktuk (94.6%): this means that when the actual class was "Tuktuk", the model correctly predicted it as "Tuktuk" 94.6% of the time.
5. F1 score:

-Background (1.00): the f1 score for the "background" class is 1.00, indicating perfect precision and recall for this class.

-Tuktuk (0.95): the f1 score for the "Tuktuk" class is 0.95, which is still very high, indicating strong performance but with some room for improvement.

In comparison to the Confusion matrix of the YOLOv5 algorithm (shown Table 4.4), it was observed that the F1 score of the FOMO algorithm was higher. Additionally, the FOMO model demonstrated the ability to predict and detect objects more accurately.

iv. Train and Validation Loss

In the FOMO algorithm, we need this measure because it is the most appropriate due to the lack of a square bound in the FOMO output. The training and validation loss progression starts from initial values of 0.37719 and 0.24065, respectively, and reaches final values of 0.01683 and 0.0183

(as shown in Figure 4.9). In the initial stage (Epoch 1-5), both training and validation losses decrease as the model begins learning from the data, with the training loss starting at 0.37719 and the validation loss at 0.24065. During early training (Epoch 6-15), the training loss drops to around 0.1 to 0.15, indicating effective learning, while the validation loss also decreases to a similar range, reflecting improved generalization.

In the mid-training phase (Epoch 16-30), the training loss continues to decrease, approaching 0.05 to 0.1, with the validation loss following suit, suggesting good model performance on unseen data. During late training (Epoch 31-50), the training loss further drops towards the final value of 0.01683, demonstrating high accuracy, and the validation loss decreases to 0.0183, indicating sustained good generalization.

In the final stage (Epoch 51-100), both losses stabilize around 0.01683 and 0.0183, respectively, showing that the model has minimized error on the training data while maintaining excellent generalization to new data. The close final values of training and validation losses suggest the model is well-regularized and avoids overfitting.

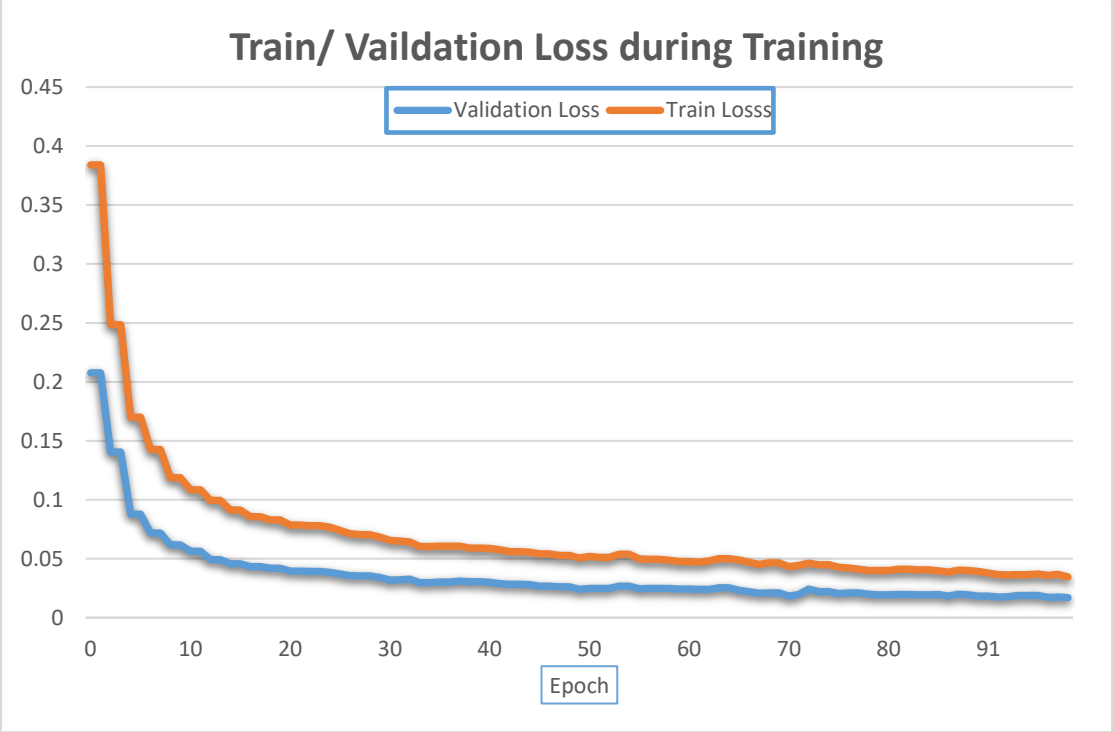

Figure 4.9 Train and Validation Loss during Training in FOMO

v. F1 Score / Recall

The F1 Score and Recall during model training start from an initial value of 0 and reach 0.95 (as shown in Figure 4.10). At the beginning of training (Epoch 1-5), both metrics start at 0.0, indicating that the model has not yet learned to identify positive instances effectively. As training progresses through the initial epochs, there is a slight improvement as the model begins to learn basic features from the data. During early training (Epoch 6-15), the F1 Score and Recall show more noticeable improvements, potentially reaching values between 0.2 and 0.4, reflecting the model's growing ability to correctly identify true positives and balance precision and recall.

In the mid-training phase (Epoch 16-30), both metrics continue to increase steadily, moving towards 0.5 to 0.7, as the model refines its ability to recognize positive instances and reduce false positives and false negatives.

The model is getting close to ideal performance with high recall and a good balance between precision and recall as seen by the F1 Score and Recall rising further during late training (Epoch 31–50) to values between 0.7 and 0.85. Both measures peak at 0.95 by the end of training (Epoch 51-100), showing good model performance with high recall and a good balance between accuracy and recall, indicating few false positives and false negatives. This increase from 0 to 0.95 demonstrates the model's efficient learning and great precision and recall achieved by well generalising to new data.

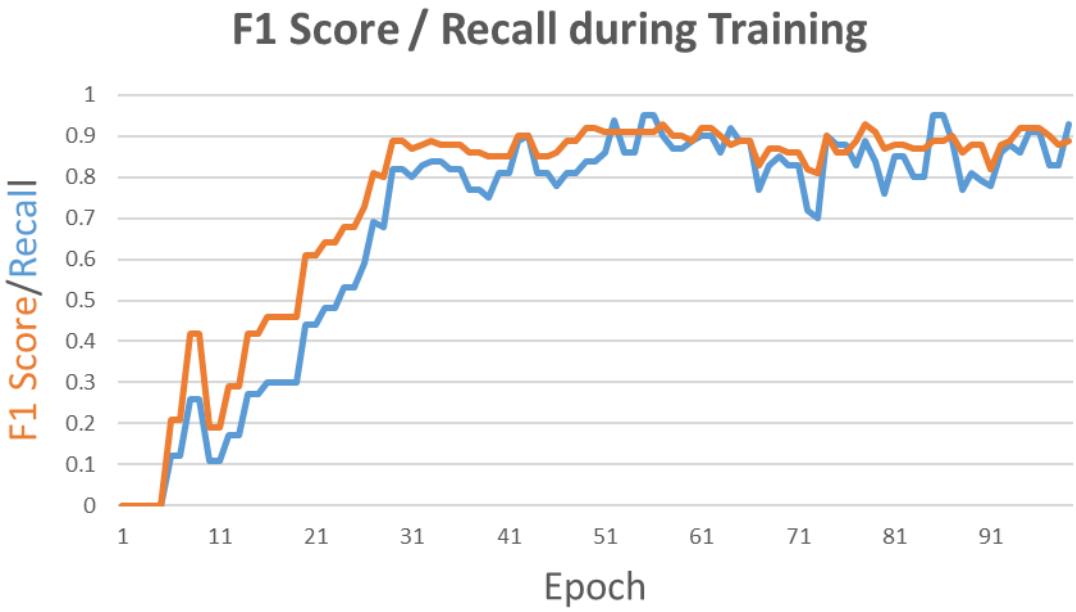

Figure4.10 Describe F1 Score and Recall during Training

4.3 Comparative analysis

The proposed strategy reveals several properties that can be effectively recorded and used to achieve a system that accurately detects blind spots with exceptional accuracy and minimal cost through the use of microcontrollers. The benefits compared to alternative approaches are illustrated in Table 4.8 provided. This thesis presents a new model that uses control units to alert the driver about cars on the sides. In addition, the model

has the ability to identify the type of vehicle as part of its object detection function.

This model also outperforms its peers in scientific research that includes detecting objects using the tinyML model, both in terms of accuracy and the resources used.

Table 4.8 Comparative analysis of the proposed method

Paper ID	Method Techniques	Hardware	Test accuracy	Resources and cost	Real time implementation
[9] 2012	partial features	Side-mounted camera	90%	---	×
[10] 2014	motion features and static features	Two cameras are mounted under side mirrors	96%	---	×
[12] 2019	DL model and Fully Connected Network	Camera-Based Blind Spot Detection and Nvidia Quadro p6000	97.58 %	High resources and High cost	✓
[13] 2020	Ultrasonic Sensor	Arduino and an ultrasonic sensor	---	Low resources and cost	×
[14] 2020	Radar Sensors	24 GHz Radar Sensors with OpenMV M7	---	High resources and low cost	✓
[15] 2023	multi-LIDAR network.	Lidar sensor	93.75% (Warning accuracy)	High resources and medium cost	✓
[16] 2022	DL modal (YOLOv3)	Depth Camera D455	---	High resources	×

Paper ID	Method Techniques	Hardware	Test accuracy	Resources and cost	Real time implementation
				and High cost	
[17] 2023	Frequency Modulated Continuous Wave (FMCW) radar technology	Blind Spot Detection BSD radar system	---	High resources and High cost	✓
[18] 2023	DL modal	High resolution single camera sensor on each side	97%	High resources and High cost	×
Proposed method	**DL modal (FOMO)**	**1-Arduino Nano33 BLE and OV7675 Camera 2-portentaH7 with vision shield**	**96.55%**	**Low resources and cost**	✓

The cost-effectiveness of our approach is a key strength, as we have provided a low-cost solution compared to traditional blind spot detection systems that rely on more expensive sensors or GPUs. This affordability makes it accessible to a wider range of users, including individuals and organizations with financial constraints, without compromising on functionality.

4.4 Discussion

In this study, we aimed to develop a blind spot detection system using cost-effective hardware, such as microcontrollers. The rationale behind this endeavor was to popularize this feature even in budget-conscious markets and to create a detection system for three-wheelers due to the numerous accidents they cause due to their small size and inability to see into blind spots.

Arduino Nano 33 BLE and Portenta H7 were selected based on their small size, low power consumption, and wireless communication capabilities, which make them suitable for embedded applications and integration into vehicles for blind spot detection purposes.

To ensure real-time performance on microcontrollers, we chose a lightweight object detection algorithm. This algorithm strikes a balance between accuracy and computational efficiency, and is optimized to run efficiently on the limited computational resources of Arduino microcontrollers. We compared three types of algorithms MobileNetV2 SSD FPN-Lite, YOLOv5, and FOMO, with the latter being the best algorithm suitable for microcontrollers. It gave excellent results on the H7 vehicle, where we obtained a test accuracy of 96.55% and validation accuracy 94.6% with inference time up to 30 milliseconds. During testing, the blind spot detection system showed promising performance, accurately detecting objects in the vehicle's blind spots in real time with a reasoning time of more than 30ms. In conclusion, the blind spot detection system developed in this study is shown to be an effective and low-cost solution for enhancing vehicle safety. By leveraging the capabilities of Arduino microcontrollers and a lightweight object detection algorithm, we demonstrated the feasibility of implementing such a system using off-the-shelf components. This work

contributes to the field of vehicle safety technology and opens up possibilities for future research and development in this area.

5

Chapter Five

Conclusions & Future Work

Chapter Five
Conclusions and Future Work

5.1 Conclusions.

Throughout the creation and completion of the planned project, we have gained significant expertise in various domains like programming, electronics, measurement, and more. Moreover, the subsequent points are deduced:

1. The system has shown efficacy and efficiency throughout hardware setup, data collecting, model creation, optimization, integration with microcontrollers, firmware development, and testing.
2. The system is competitive for blind spot detection because of its efficiency and affordability in comparison to other systems.
3. Situational awareness and general safety are increased when the system can determine the kind of car on the driver's side.
4. More optimization of the system for practical application and extension of its capacity to identify other possible road dangers could be the main goals of future development.
5. We concluded that the best object detection algorithm for working with microcontrollers is FOMO, and the most effective way to reduce the model size to fit within the TinyML framework is Quantization (int8).

5.2 Future Work

- Search for different types of vehicles so that every vehicle on the road is covered.
- Using embedded devices with greater capabilities and resources than those previously used.
- Improving object recognition in blind spots, including identifying pedestrians, cars and other potential hazards.
- Create an easy-to-use interface that provides blind spot detection system drivers with clear and understandable warnings and feedback.
- To improve situational awareness and safety in self-driving cars, blind spot detection can be integrated with self-driving vehicle technologies.
- Utilize huge datasets to train the model to assess whether its accuracy decreases or remains unaffected.

References

[1] Strandroth, Johan, et al. Lives saved by accelerating the implementation of vehicle safety technology in New South Wales. No. 2021-22-0001. SAE Technical Paper, 2022.

[2] Bhalla, Kavi, and Kevin Gleason. "Effects of vehicle safety design on road traffic deaths, injuries, and public health burden in the Latin American region: a modelling study." The Lancet Global Health 8.6 (2020): e819-e828.

[3] Widen, William H., and Philip Koopman. "Autonomous Vehicle Regulation & Trust: The Impact of Failures to Comply with Standards." UCLA JL & Tech. 27 (2022): 169.

[4] World Health Organization. Pedestrian safety: a road safety manual for decision-makers and practitioners. World Health Organization, 2023.

[5] Sharma, Aditya Kumar. "A Comprehensive Analysis of the School Bus Driver Shortage in the United States." Journal of Marketing & Supply Chain Management. SRC/JMSCM-146. DOI: doi. org/10.47363/JMSCM/2022 (1) 129 (2022): 2-9.

[6] Jansen, Reinier J., and Silvia F. Varotto. "Caught in the blind spot of a truck: A choice model on driver glance behavior towards cyclists at intersections." Accident Analysis & Prevention 174 (2022): 106759.

[7] Barat, Anik, and Sanhita Das. "A systematic literature review of driver behaviour in blind spot: Assessing risk and influencing factors for enhanced ergonomics in vehicle design." Ergonomics (2024): 1-17.

[8] Adeleke, Richard, and Ayodeji Emmanuel Iyanda. "Transport fare and road traffic crashes in Nigeria: insights from a geographical analysis." International journal of injury control and safety promotion (2024): 1-9.

[9] Kim, Wantae, Heejin Yang, and Jinhong Kim. "Blind Spot Detection Radar System Design for Safe Driving of Smart Vehicles." Applied Sciences 13.10 (2023): 6147.

[10] Lin, Bin-Feng, et al. "Integrating appearance and edge features for sedan vehicle detection in the blind-spot area." IEEE Transactions on Intelligent Transportation Systems 13.2 (2012): 737-747.

[11] Tseng, Din-Chang, Chang-Tao Hsu, and Wei-Shen Chen. "Blind-spot vehicle detection using motion and static features." International Journal of Machine Learning and Computing 4.6 (2014): 516.

[12] Kiefer, R. J., & Mayer, G. G. (2017). Effectiveness of Blind Spot Mirrors in Reducing Lane Change Crashes. Accident Analysis & Prevention, 99(Pt A), 98-106.
[13] Zhao, Yiming, et al. "Camera-based blind spot detection with a general purpose lightweight neural network." Electronics 8.2 (2019): 233.
[14] Adnan, Z., et al. "Vehicle blind spot monitoring phenomenon using ultrasonic sensor." International Journal of Emerging Trends in Engineering Research 8.8 (2020).
[15] Nor, M., et al. "Development of Smart Vehicle Blind Spot Detection System Based on 24GHz Radar Sensors." International Journal of Engineering and Advanced Technology (IJEAT) 9 (2020): 2726-2730.
[16] Negishi, Jumpei, et al. "Edge system for providing blind-spot information using the multi-LIDAR network." 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC). IEEE, 2023.
[17] Wang, Zijun, Qun Jin, and Bo Wu. "Design of a Vision Blind Spot Detection System Based on Depth Camera." 2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, 2022.
[18] Kim, Wantae, Heejin Yang, and Jinhong Kim. "Blind Spot Detection Radar System Design for Safe Driving of Smart Vehicles." Applied Sciences 13.10 (2023): 6147.
[19] Nezhad, Arash Pourhasan, Mehdi Ghatee, and Hedieh Sajedi. "Blind Spot Warning System based on Vehicle Analysis in Stream Images by a Real-Time Self-Supervised Deep Learning Model." Authorea Preprints (2023).
[20] Mahesh, Batta. "Machine learning algorithms-a review." International Journal of Science and Research (IJSR).[Internet] 9.1 (2020): 381-386.
[21] Ghosh, Swarnendu, et al. "Understanding deep learning techniques for image segmentation." ACM computing surveys (CSUR) 52.4 (2019): 1-35.
[22] Szeliski, Richard. Computer vision: algorithms and applications. Springer Nature, 2022.

[23] Taye, Mohammad Mustafa. "Understanding of machine learning with deep learning: architectures, workflow, applications and future directions." Computers 12.5 (2023): 91.
[24] Wang J, Ma Y, Zhang L, Gao RX (2018) Deep learning for smart manufacturing: Methods and applications. J Manuf Syst 48:144–156. https://doi.org/10.1016/J.JMSY.2018.01.003
[25] Voulodimos A, Doulamis N, Doulamis A, Protopapadakis E (2018) Deep Learning for Computer Vision: A Brief Review. Comput Intell Neurosci 2018:1–13. https://doi.org/10.1155/2018/7068349
[26] Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet Classification with Deep Convolutional Neural Networks. NIPS'12 Proc 25th Int Conf Neural Inf Process Syst 1:1097–1105.
[27] Liang, Ming, and Xiaolin Hu. "Recurrent convolutional neural network for object recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. O' Mahony N, Murphy T, Panduru K, et al (2016) Adaptive process control and sensor.
[28] Liu, Yao, Hongbin Pu, and Da-Wen Sun. "Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices." Trends in Food Science & Technology 113 (2021): 193-204.
[29] He, Juncai, et al. "ReLU deep neural networks and linear finite elements." arXiv preprint arXiv:1807.03973 (2018).
[30] Horiguchi S, Ikami D, Aizawa K (2017) Significance of Softmax-based Features in Comparison to Distance Metric Learning-based Features
[31] He, Fengxiang, Tongliang Liu, and Dacheng Tao. "Control batch size and learning rate to generalize well: Theoretical and empirical evidence." Advances in neural information processing systems 32 (2019).
[32] Jepkoech, Jennifer, et al. "The effect of adaptive learning rate on the accuracy of neural networks." (2021).
[33] Hoefler, Torsten, et al. "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks." Journal of Machine Learning Research 22.241 (2021): 1-124.
[34] Dargan, Shaveta, et al. "A survey of deep learning and its applications: a new paradigm to machine learning." Archives of Computational Methods in Engineering 27 (2020): 1071-1092.

[35] Pathak, Ajeet Ram, Manjusha Pandey, and Siddharth Rautaray. "Application of deep learning for object detection." Procedia computer science 132 (2018): 1706-1717.
[36] Sengupta, Saptarshi, et al. "A review of deep learning with special emphasis on architectures, applications and recent trends." Knowledge-Based Systems 194 (2020): 105596.
[37] Rastogi, Deependra, et al. "Multi-class classification of brain tumour magnetic resonance images using multi-branch network with inception block and five-fold cross validation deep learning framework." Biomedical Signal Processing and Control 88 (2024): 105602.
[38] Alqahtani, Ola Mustafa, and Lakshmish Macheeri Ramaswamy. "A Layer Decomposition Approach to Inference Time Prediction of Deep Learning Architectures." 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2022.
[39] Varoquaux, Gaël, and Olivier Colliot. "Evaluating machine learning models and their diagnostic value." Machine Learning for Brain Disorders (2023): 601-630.
[40] Rodrigo Verschae and Javier Ruiz-del Solar. Object Detection:Current and Future Directions. Frontiers in Robotics and AI, 2, 2015
[41] Real-Time Motion Detection and Surveillance using Approximation of ImagePre-processing Algorithms. IEEE, 2019.
[42] Computer Vision Application Analysis based on Object Detection,2023
[43] Qiang Bai, Shaobo Li, Jing Yang, Qisong Song, Zhiang Li, andXingxing Zhang. Object Detection Recognition and Robot Grasp-ing Based on Machine Learning: A Survey. IEEE Access, 8:181855–181879, 2020
[44] Khan, Asifullah, et al. "A survey of the recent architectures of deep convolutional neural networks." Artificial intelligence review 53 (2020): 5455-5516.
[45] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized Intersec-tion over Union: A Metric and A Loss for Bounding Box Regression,2019.
[46] Henderson, Paul, and Vittorio Ferrari. "End-to-end training of object class detectors for mean average precision." Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei,

Taiwan, November 20-24, 2016, Revised Selected Papers, Part V 13. Springer International Publishing, 2017.
[47] Maxwell, Aaron E., Timothy A. Warner, and Luis Andrés Guillén. "Accuracy assessment in convolutional neural network-based deep learning remote sensing studies—Part 1: Literature review." Remote Sensing 13.13 (2021): 2450.
[48] Kumar, Ashwani, Zuopeng Justin Zhang, and Hongbo Lyu. "Object detection in real time based on improved single shot multi-box detector algorithm." EURASIP Journal on Wireless Communications and Networking 2020.1 (2020): 204.
[49] Zheng, Yu, et al. "Rotation-robust intersection over union for 3d object detection." European Conference on Computer Vision. Cham: Springer International Publishing, 2020.
[50] Wang, Beinan. "A parallel implementation of computing mean average precision." arXiv preprint arXiv:2206.09504 (2022).
[51] Aoueileyine, Mohamed Ould-Elhassen. "Tiny Machine Learning for IoT and eHealth Applications: Epileptic Seizure Prediction Use Case." International Conference on Digital Technologies and Applications. Cham: Springer Nature Switzerland, 2023.
[52] Schizas, Nikolaos, et al. "TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review." Future Internet 14.12 (2022): 363.
[53] Sanchez-Iborra, Ramon, and Antonio F. Skarmeta. "Tinyml-enabled frugal smart objects: Challenges and opportunities." IEEE Circuits and Systems Magazine 20.3 (2020)
[54] Abadade, Youssef, et al. "A comprehensive survey on tinyml." IEEE Access (2023).
[55] Ray, Partha Pratim. "A review on TinyML: State-of-the-art and prospects." Journal of King Saud University-Computer and Information Sciences 34.4 (2022): 1595-1623.
[56] Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., & Hafid, A. S. (2023). A comprehensive survey on tinyml. IEEE Access.
[57] Zim, Md Ziaul Haque. "TinyML: analysis of Xtensa LX6 microprocessor for neural network applications by ESP32 SoC." arXiv preprint arXiv:2106.10652 (2021).
[58] Warden, Pete, and Daniel Situnayake. Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers. O'Reilly Media, 2019.

[59] Bell, Charles. MicroPython for the Internet of Things. Berlin/Heidelberg, Germany: Springer, 2017.
[60] Raza, Wamiq, et al. "Energy-efficient inference on the edge exploiting TinyML capabilities for UAVs." Drones 5.4 (2021): 127.
[61] Catsoulis, John. Designing Embedded Hardware: Create New Computers and Devices. " O'Reilly Media, Inc.", 2005.
[62] Kamath, Vidya, and A. Renuka. "Deep Learning Based Object Detection for Resource Constrained Devices-Systematic Review, Future Trends and Challenges Ahead." Neurocomputing (2023).
[63] Zhou, Xu, et al. "A Survey of the Security Analysis of Embedded Devices." Sensors 23.22 (2023): 9221.
[64] Rakkiannan, Thamilselvan, et al. "An automated network slicing at edge with software defined networking and network function virtualization: a federated learning approach." Wireless Personal Communications 131.1 (2023): 639-658.
[65] Mittal, Sparsh, and Jeffrey S. Vetter. "A survey of software techniques for using non-volatile memories for storage and main memory systems." IEEE Transactions on Parallel and Distributed Systems 27.5 (2015): 1537-1550.
[66] Ali, Mustafa, et al. "Compute-in-memory technologies and architectures for deep learning workloads." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 30.11 (2022): 1615-1630.
[67] Björnsson, Unnar Elías. Designing a Multifunctional Sensor Board for Teaching IoT and Microcontroller Fundamentals. Diss. 2023.
[68] Jolles, Jolle W. "Broad-scale applications of the Raspberry Pi: A review and guide for biologists." Methods in Ecology and Evolution 12.9 (2021): 1562-1579.
[69] Madhekar, Dhanashree Vijayrao, and Mrinal Rahul Bachute. "Real time object detection and tracking using Raspberry Pi." IJESC paper 7.6 (2017).
[70] Soviany, Petru, and Radu Tudor Ionescu. "Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction." 2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). IEEE, 2018.
[71] Aamir, Muhammad, et al. "A Progressive Approach to Generic Object Detection: A Two-Stage Framework for Image Recognition." Computers, Materials & Continua 75.3 (2023).

[72] Wang, Tong, et al. "Attention-guided knowledge distillation for efficient single-stage detector." 2021 IEEE International Conference on Multimedia and Expo (ICME). IEEE, 2021.
[73] Carranza-García, Manuel, et al. "On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data." Remote Sensing 13.1 (2020): 89.
[74] Zhou, Yan, et al. "Review of research on lightweight convolutional neural networks." 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC). IEEE, 2020.
[75] Yan, Rui, et al. "STMS-YOLOv5: A Lightweight Algorithm for Gear Surface Defect Detection." Sensors 23.13 (2023): 5992
[76] Liu, Haiying, et al. "Sf-yolov5: A lightweight small object detection algorithm based on improved feature fusion mode." Sensors 22.15 (2022): 5817.
[77] Srivastava, Harsh, and Kishor Sarawadekar. "A depthwise separable convolution architecture for CNN accelerator." 2020 IEEE Applied Signal Processing Conference (ASPCON). IEEE, 2020.
[78] Lu, Gangzhao, Weizhe Zhang, and Zheng Wang. "Optimizing depthwise separable convolution operations on gpus." IEEE Transactions on Parallel and Distributed Systems 33.1 (2021): 70-87.
[79] Nguyen, Hoanh. "Fast object detection framework based on mobilenetv2 architecture and enhanced feature pyramid." J. Theor. Appl. Inf. Technol 98.05 (2020).
[80] Sanchaari, A. (2024) Convoluted truth of CNN (Convolutional Neural Networks), Medium.Available at: https://blog.stackademic.com/convoluted-truth-of-cnn-convolutional-neural-networks-843a72eac0e0 (Accessed: 31 March 2024).
[81] Dharani, Andhe, S. Anupama Kumar, and Peethi N. Patil. "Object Detection at Edge Using TinyML Models." SN Computer Science 5.1 (2023): 11.
[82] Novak, Milan, et al. "Intelligent inspection probe for monitoring bark beetle activities using embedded IoT real-time object detection." Engineering Science and Technology, an International Journal 51 (2024): 101637.
[83] FOMO: Object Detection for constrained devices (no date) FOMO: Object detection for constrained devices - Edge Impulse Documentation. Available at: https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-

blocks/object-detection/fomo-object-detection-for-constrained-devices(Accessed:12 February 2024).
[84] Dipert, B. (2023) 'FOMO: Real-time object detection on microcontrollers,' a presentation from Edge Impulse, Edge AI and Vision Alliance. Available at: https://www.edge-ai-vision.com/2022/06/fomo-real-time-object-detection-on-microcontrollers-a-presentation-from-edge-impulse/ (Accessed: 12 February 2024).
[85] Budd, Samuel, Emma C. Robinson, and Bernhard Kainz. "A survey on active learning and human-in-the-loop deep learning for medical image analysis." Medical image analysis 71 (2021): 102062.
[86] Priyadharshini, S., and Ansuman Mahapatra. "Spherical video synopsis generation and visualization framework." Journal of Intelligent & Fuzzy Systems Preprint (2023): 1-16.
[87] Preechakul, Konpat, et al. "Improved image classification explainability with high-accuracy heatmaps." Iscience 25.3 (2022).
[88] Preechakul, Konpat, et al. "Improved image classification explainability with high-accuracy heatmaps." Iscience 25.3 (2022).
[89] Zamir, Amir R., et al. "Taskonomy: Disentangling task transfer learning." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
[90] Alankar, Bhavya, et al. "Deep Learning Model for Computer Vision: Sustainable Image Classification Using Machine Learning." International Conference on Sustainable Development through Machine Learning, AI and IoT. Cham: Springer Nature Switzerland, 2023.
[91] Azunre, Paul. Transfer learning for natural language processing. Simon and Schuster, 2021.
[92] Pruksachatkun, Yada, et al. "Intermediate-task transfer learning with pretrained models for natural language understanding: When and why does it work?." arXiv preprint arXiv:2005.00628 (2020).
[93] Han, Xu, et al. "Pre-trained models: Past, present and future." AI Open 2 (2021): 225-250.
[94] Alyafeai, Zaid, Maged Saeed AlShaibani, and Irfan Ahmad. "A survey on transfer learning in natural language processing." arXiv preprint arXiv:2007.04239 (2020).
[95] Kurniawan, Agus. "Iot projects with arduino nano 33 ble sense." Berkeley: Apress 129 (2021).

[96] Pino, Daniel Lopez, Felix Freitag, and Mennan Selimi. "Designing a double LoRa connectivity for the Arduino Portenta H7." 2022 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). IEEE, 2022.