Master Thesis

in fulfillment of the requirements for the degree of
Master of Science in
Artificial Intelligence and Data Analytics
at Hochschule Neu-Ulm

Design and Implementation of a Neural Network for Obstacle Avoidance in Robotic Lawn Mowers

submitted by

Kishan Ajudiya

Matriculation Nr. 328658

Submitted on October 28, 2024

Academic Supervisors

Prof. Dr. Stefan Faußer

Prof. Dr. Philipp Brune

Information Management

Hochschule Neu-Ulm

Company Supervisors

Jannik Mehrke

Peter Müller

TCC-E Department

AL-KO Gardentech

Declaration of Authorship

I, Kishan Ajudiya, declare that this thesis titled, "Design and Implementation of a Neural Network for Obstacle Avoidance in Robotic Lawn Mowers" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date: 28.10.2024

Kishan Ajudiya

Abstract

Many garden owners consider lawn mowing as a necessity and an additional burden, with growing interest in automating this task through robotic lawn mowers. However, current robotic lawn mowers face several limitations. There are several operational challenges, such as collision with the walls and damage due to stones. Moreover, they pose a potential risk to surrounding environment. To address these challenges, this thesis explores image segmentation as a method to enhance the ability of robotic lawn mowers to perceive and interpret their surroundings more accurately. Thus it could enable safe autonomous navigation in robotic lawn mowers.

The research begins with an exploration of Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL) and Computer Vision (CV), followed by an in-depth study of image segmentation models. A detailed quantitative comparison of various segmentation models highlights their strengths and weaknesses in terms of accuracy and computational efficiency. After careful consideration, the YOLOv8s segmentation model was selected for this research. A custom image dataset consisting of 2176 images was prepared and labeled specifically for this task. Then YOLOv8s segmentation model was trained on this dataset. Then hyperparameter of the model were tuned to increase its accuracy. Following this, the model was optimized to NCNN format to provide faster inference when deploying it on Raspberry Pi 5.

The optimized model was deployed on Raspberry Pi 5 and tested in real-world garden scenario. The experimental results demonstrated the model's capacity to accurately segment lawn area and critical objects (e.g., humans, dogs). After conversion to NCNN, the model saw a significant improvement in Frames Per Second (FPS) and inference time, achieving a fourfold increase in real-time performance. Despite of this, there was a minor accuracy trade-off. The thesis concludes with a discussion of the model's limitations and potential future improvements that could be made to further enhance robotic lawn mower operations.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

ML Machine Learning

DL Deep Learning

CV Computer Vision

ANN Artificial Neural Network

FNN Feedforward Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

LSTM Long Short-Term Memory

RCNN Region-based Convolutional Neural Network

RPN Region Proposal Network

RoI Region of Interest

IoU Intersection over Union

mIoU Mean Intersection over Union

SSD Single Shot Detector

YOLO You Only Look Once

NMS Non-Maximum Suppression

RL Reinforcement Learning

COCO Common Objects in Context

GPU Graphics Processing Unit

TPU Tensile Processing Unit

tanh Hyperbolic Tangent

ReLU Rectified Linear Unit

FPS Frames per Second

TPR True Positive Rate

TP True Positives

FP False Positives

TN True Negatives

FN False Negatives

mAP Mean Average Precision

AWS Amazon Web Services

JSON Javascript Object Notation

1 Introduction

1.1 Background

Lawn maintenance is a task that many garden enthusiasts and home owners face regularly. For some people, it is a daily routine to engage with nature and take pride in a well-manicured lawn. In contrast, mowing the lawn is a time-consuming activity that consumes several hours of work for many others. Thus, this repetitive task often leads to a desire for more convenient and less labor-intensive solutions.

As technology continues to evolve, the concept of automating routine activities has also increased. The emergence of lawn mowers is one of the most popular technological developments in this area . Nowadays lawn mowers are widely used for trimming the grasses in the fields which acts like an interior decorator in companies, colleges, etc [1]. First introduced in the 1990s, these devices represented a significant leap forward in lawn maintenance technology. This new innovation is now a feasible option for people wanting to keep their lawns looking good with little effort.

Over time, robotic lawn mowers have greatly developed to improve user experience and operational efficiency. The news report by Allied Market Research stated that the global autonomous lawnmower market reached $538 million in 2017 and is projected to reach $1437 million by 2025, with a 12.9% compound annual growth rate from 2018 to 2025 [2]. This increment in market expansion shows the rapid growth of the robotic lawn mower market in recent times.

Despite making some advancements, many traditional lawn mowers still rely on simplistic technologies for navigation and operation. It potentially limited their effectiveness in various dynamic environments. There are significant problems about the safety of individuals with conventional lawn mowers. There is still a high possibility for improvement. Further investigation in the research shows that integrating cutting-edge technologies such as artificial intelligence and computer vision offers potential benefits in overcoming the limitations of conventional mowers. By improving the mower's capacity to recognize and react to its environment, we move closer to the objective of achieving fully autonomous, secure, and efficient lawn maintenance.

1.2 Motivation

The increasing demand for efficiency and convenience in lawn maintenance the reason for growing interest in the development of robotic lawn mowers. Although, while these lawn mowers present an innovative solution to a traditionally labor-intensive lawn mowers, there are still certain limitations.

One of the primary motivations for this research is the necessity to enhance safety of surrounding of robotic lawn mowers. Existing models typically rely on basic bump sensors

Figure 1: Lawn Mower with Camera and AI Integration [5]

to detect obstacles. These sensors only react after a physical collision with obstacles. The mower must first collide with obstacle to activate the sensor and change direction. This design presents high safety risks, particularly for small animals such as hedgehogs or children. Lawn mower injuries to children are often devastating, limb-threatening, and preventable [3]. Moreover, inability of these devices to recognize and react to smaller obstacles poses safety risks to it's surrounding environment. For instance, small stones left on the lawn may be run over by lawn mower. This may potentially damage mowing blade or even injure someone.

Another motivating factor is the growing expectation for smarter, more adaptable vision based technologies that can easily integrate into automatic lawn mowers. The advancement of deep learning and computer vision technologies presents an opportunity to revolutionize the capabilities of robotic lawn mowers [4]. By enabling these machines to "see" and interpret their surroundings through integration of camera and AI, the limitations of current models can be solved. As shown in the Figure 1, lawn mower could see the hedgehog with camera and AI for it could change the direction to save hedgehog. This would not only improve the efficiency and coverage of lawn mowing but also provide more safety.

Additionally, the market for robotic lawn mowers is majorly driven by the rise in demand for automation [6]. As consumers become more aware of the benefits of automation and the convenience it offers, the demand for more intelligent and reliable lawn care solutions is expected to grow [6]. Addressing the limitations of current lawn mowers by implementing cutting-edge technologies could position these robotic lawn mowers as essential tools in modern households. Thus it provides further adoption and market expansion.

The objective of this research is to investigate and develop more sophisticated methodology

for obstacle avoidance in robotic lawn mowers. Thus, this research aims to develop a system in lawn mower that is capable of real-time environmental perception through the utilization of deep learning technology. Ultimately, this research contributes to the advancement of autonomous lawn maintenance technology for safer and more efficient lawn mowing in diverse and dynamic environments.

1.3 AL-KO Company Description

AL-KO was established in 1931 by Alois Kober in the Bavarian-Swabian town of Großkötz, situated in close proximity to Günzburg. It began as a small metalworking shop. Over the decades, the company has grown into global leader, operating in 19 countries worldwide. Today, AL-KO is recognized as a leading provider in three major areas: vehicle technology, garden technology, and air technology [7].

Figure 2: AL-KO Gardentech headquarter in Großkötz [7]

The Gardentech division plays a crucial role in AL-KO's product portfolio. This project is situated in this division as seen in Figure 2. This division is engaged for design and development of an extensive garden tools and equipment, including lawn mowers, water pumps, blowers, leaf vacuums, hedge cutters, and chainsaws. The products in this division

are developed with a focus on innovation, reliability, and user-friendly operation to cater both hobbyists and professional gardeners.

Figure 3: Robolinho - Robotic Lawn Mower [8]

One of the flagship products in the Gardentech division is AL-KO's robotic lawn mower, known as Robolinho as shown in Figure 3 [8]. The Robolinho series of robotic lawn mowers exemplifies AL-KO's commitment to autonomous lawn mowing solution in garden care. The current version of Robolinho operates with pre-defined boundaries. It relies on bump sensors to change direction upon encountering obstacles. The collision of the mower with obstacle could pose a high risk to it's surrounding. Thus there is a lack of technology in existing lawn mowers that can visually sense the dynamic surrounding and avoid collision. This challenge can be solved by integrating camera and advanced artificial intelligence (AI) technology.
The objective of this project is to address the limitations of the existing Robolinho model by integrating machine learning and computer vision technologies. This research is carried out to improve mower's ability to perceive and navigate its surroundings, with the aim of improving safety and operational efficiency.

1.4 Structure of This Work

In this thesis, the work is structured into the following key sections:

- Literature Review
  This section presents the literature review for this thesis. Afterwards there has been also discussion of the identified research gap and formulation of the research question.

- Research Design and Methodology
  This section outlines methodological approach of the project. It describes overall architecture to design and implement neural network on lawn mower. It starts with the dataset preparation and set up of model training. It also provides details about the hardware setup and experimental configurations performed in this thesis. Moreover, the evaluation metrics and requirements of them are also defined for this study.

- Results
  Here, the findings from training experiments are presented. Performance metrics from training experiments, hyperparameter tuning and final optimized model have been discussed. It also analyses the performance of final developed model in real-world testing.

- Discussion
  This section explains the critical discussion for the topic. This provides better understanding of the practical implications of study's findings.

- Conclusion and Future Outlook
  The final section summarizes the key takeaways from research and proposes directions for future work.

2 Literature Review

In this section, the fundamental concepts of artificial intelligence, machine learning, deep learning and computer vision have been discussed. The theoretical foundations of artificial neural networks (ANNs) are then discussed. Convolutional neural networks (CNNs) and advances in computer vision are also emphasized for their contributions to the image processing problems. Furthermore, the architecture of image segmentation models is examined, followed by a quantitative comparison of performance of existing image segmentation models. It concludes with identifying research gaps and formulating research questions that guide this study.

2.1 Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence (AI) is typically defined as an interactive, autonomous, self-learning agency with the ability to perform cognitive functions in contrast to the natural intelligence displayed by humans, such as sensing and moving, reasoning, learning, communicating, problem solving [9]. The term Artificial Intelligence, first used by computer scientist John McCarthy in 1956, describes the study and engineering of intelligent machines and software that are able "to think, understand, and make decisions like human beings", enabling intelligent machines to learn and evolve from past experience [10] [11]. Artificial intelligence is considered a key driving force of the Industry 4.0 industrial revolution due to its strong capabilities in areas such as prediction, automation or personalization [9].

Figure 4: Human Intelligence can be illustrated as a dynamic integration of abilities, such as learning and decision-making, sensing and communication. each of them are essential for adaptive intelligence [9].

Considered a subset of AI, machine learning (ML) exhibits the experiential “learning” associated with human intelligence, while also having the capacity to learn and improve its analyses through the use of computational algorithms [12]. Large data sets are used

by these algorithms as inputs and outputs to identify patterns and efficiently "learn," enabling the machine to be trained to make suggestions or judgments on its own. The machine can take an input and predict an output when the method has been modified and repeated enough times. Outputs are then compared with a set of known outcomes in order to judge the accuracy of the algorithm, which is then iteratively adjusted to perfect the ability to predict further outcomes [12]. These algorithms are used in machine learning, which enables robots to learn from data and improve over time. It's possible to program robots to carry out certain jobs in robotics, such as grasping, object identification, and path planning [13].

Categories of Machine Learning

Four main categories can be used to categorize machine learning: supervised learning, semi-supervised learning, unsupervised learning and reinforcement learning.

1. Supervised Learning:
   Supervised learning is machine learning task of learning a function that maps an input to output based on example input-output pairs [14]. The model makes predictions on new, unknown data by learning the relation between input and output. Applications like classification (e.g., spam detection, facial recognition) and regression (e.g., home price prediction) frequently make use of supervised learning. The accuracy with which the model can forecast results on a test set is used to measure its performance.

2. Semi-supervised Learning:
   Combining supervised and unsupervised machine learning techniques is known as semi-supervised machine learning. It can be fruit-full in those areas of machine learning and data mining where the unlabeled data is already present and getting the labeled data is a tedious process [14]. Semi-supervised learning achieves a balance between the scalability of unsupervised learning and the accuracy of supervised learning by utilizing both kinds of data.

3. Unsupervised Learning:
   Unsupervised learning operates on datasets without output labels. With this kind of learning, the model looks for structures or patterns on its own in the incoming data. Clustering (e.g., grouping related things, such as customer segmentation) and dimensionality reduction (e.g., reducing complex data while maintaining crucial information) are common unsupervised learning problems. When new data is introduced, it uses the previously learned features to recognize the class of the data [14].

4. Reinforcement Learning:
   Reinforcement Learning (RL) is a distinct paradigm of machine learning where

an agent learns to make decisions by interacting with an environment [14]. In reinforcement learning, an agent takes actions and gets feedback in the form of rewards or penalties depending on how those actions turn out. This process is known as trial-and-error. The agent wants to learn a policy—a method that maximizes the cumulative reward over time—in order to achieve this goal. Because of this method, RL can handle difficult sequential decision-making situations, which makes it especially helpful in domains like robotics, gaming, and autonomous systems. Through continuous interaction and adaptation, RL enables systems to improve their performance in dynamic and uncertain environments.

Deep learning (DL) is a sub-field of machine learning which attempts to learn high-level abstractions in data by utilizing hierarchical architectures [15]. This is a relatively new method that has been extensively used in many classical AI fields, such as computer vision, natural language processing, semantic parsing, transfer learning, and many more. With its layered structure of artificial neural networks, namely Convolutional Neural Networks (CNNs) for visual data, DL allows automatic feature extraction. There are mainly three important reasons for the booming of deep learning today: the dramatically increased chip processing abilities (e.g. GPU units), significantly lowered cost of computing hardware, and the considerable advances in the machine learning algorithms [16].

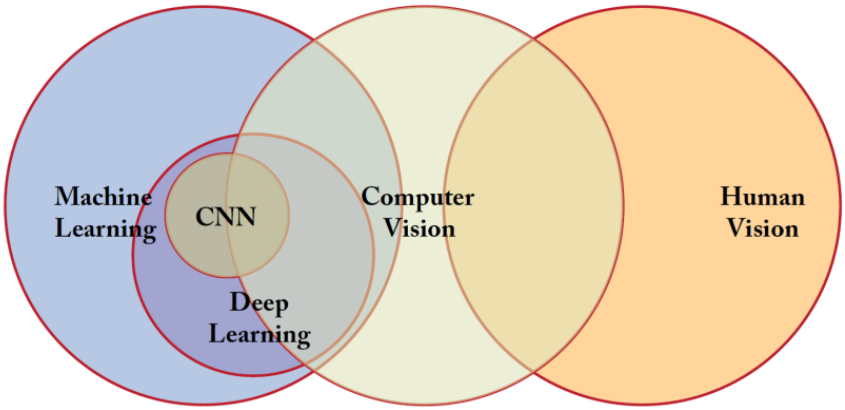

Figure 5: Computer Vision uses Machine Learning algorithms to mimic the way Human Vision works. Deep Learning is a subset of ML that employs convolutional neural networks (CNN) [17].

As depicted in the Figure 5, DL connects deeply with both Machine Learning and Computer Vision. Deep learning techniques, like CNNs, have transformed computer vision tasks by making robots much more capable of interpreting visual data with little assistance from humans. Human Vision, on the other hand, serves as an inspiration for both Computer Vision and DL models. While the human brain processes visual data through highly

complex neural mechanisms, DL attempts to replicate this ability using artificial neural networks, making connections between visual inputs and meaningful outcomes. This interrelation highlights the progress toward bridging the gap between artificial perception and natural human vision, making DL a key factor in advancing autonomous systems, medical imaging, and various AI-driven vision applications.

2.2 Artificial Neural Networks (ANN) Theory

ANNs are artificial adaptive systems that are inspired by the functioning processes of the human brain [18]. Artificial Neural Networks (ANNs) have a long and complex history, with their roots dating back to the 1940s. The idea originated from an effort to replicate the way the human brain processes information by simulating biological neural networks' structure and operation through a network of connected nodes, or "neurons." The network that resembles or mimics the biological human brain functions to accomplish a given task is an artificial neural network [19]. The journey of ANNs has been marked by alternating periods of significant progress and periods of stagnation, driven by both technological advances and theoretical breakthroughs. These are systems with the capacity to alter their internal organization in response to a functional goal. They are particularly suited for solving problems of the nonlinear type, being able to reconstruct the fuzzy rules that govern the optimal solution for these problems [18].

Artificial Neurons

The concept of ANN starts its inspiration from the human brain, particularly its building blocks, the neurons. The human brain is a powerful tool that can do tasks such as thinking, recognizing, and solving hard and complex problems, and to do all of that, the neuron - an electrically excitable cell - plays a big part. Human brain consists of an estimated 90 to 100 billion neurons, where each neuron is connected with 1 to 10 thousand others, which makes up to 10^{15} interconnections [20]. This massive web of billions of interconnected neurons working together allows the brain to achieve its amazing abilities [21].

ANNs also have artificial neurons, as biological neural networks have neurons [22]. The neurons utilized in Artificial Neural Networks (ANN) often have several inputs, through which it is typically connected to other neurons as depicted in Figure 6. Information that a neuron receives from the outside environment is known as an input. Weights indicate the significance of information arriving at neurons and how it affects them. Every input has a different weight. In Figure 6, W_1 weight illustrates how the neuron responds to x_1 input. Weights may not necessarily indicate significance or importance based on their size. The net input that reaches a neuron is determined by the transfer function. The weighted sum is the most often used transfer function for this, despite the fact that there are many others. Every piece of incoming data is multiplied by its own weight to get its

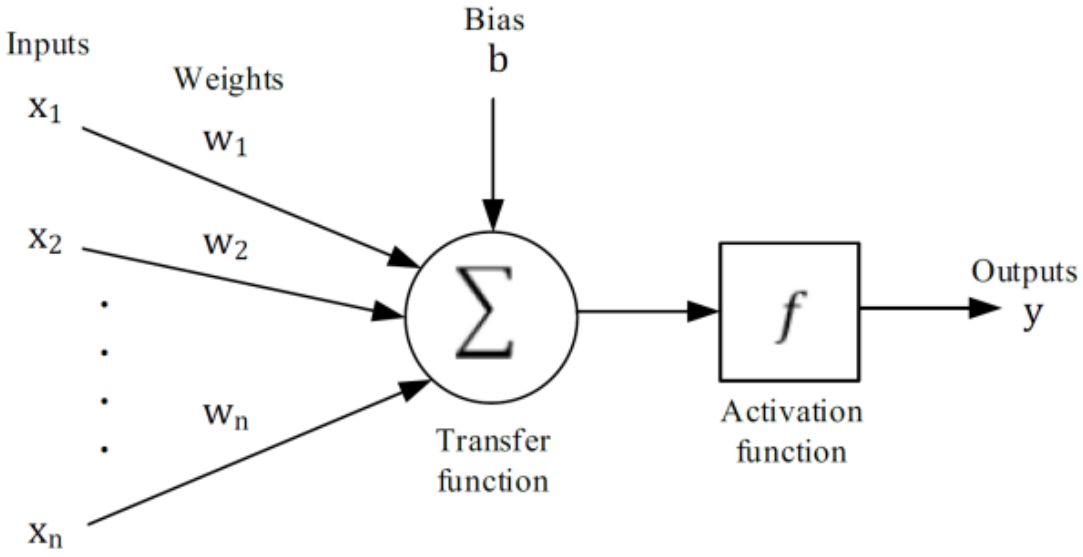

Figure 6: The artificial neuron is a function f that receives one or more inputs x, applies weights w to these inputs, and sums them to produce an output y. Bias b is the constant which is added to the product of inputs x and weights w [22].

total. By analyzing the net input that enters the neuron, the activation function calculates the output that the neuron will produce in response to this input. The produced output is transmitted to either another neuron or the outside world. [22]

$$y = f\left(\sum_{i=1}^{n} W_i x_i + b\right) \tag{2.1}$$

Typically, a nonlinear function is selected as the activation function. Transferring the nonlinearity to the neuron's output is the aim of the activation function. Because of the nonlinearity of activation functions, ANNs have a property known as nonlinearity. It's crucial to keep in mind that the function's derivative is simple to compute while selecting the activation function. This guarantees that the computations happen fast.

In the literature, there are many activation functions such as linear, step, sigmoid, hyperbolic tangent (tanh), Rectified Linear Unit (ReLU) and threshold functions [23]. On the other hand, ANN applications typically use the sigmoid, tanh, and ReLU activation functions.

A continuous and derivable function is the sigmoid activation function. It is among the most often utilized ANN activation functions. For every input value, this function produces a value between 0 and 1. This can be computed using the equation that follows [22]:

$$\sigma(x) = \frac{e^x}{1 + e^x} \tag{2.2}$$

The sigmoid activation function and the tanh activation function are comparable. Nonetheless, the output values are between -1 and 1. For this function, the formula is [22]:

$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.3}$$

The ReLU activation function generates an output with a threshold value of 0 for each of the input values. The use of this function in ANNs has grown popularity recently. The formula of ReLU activation function is [23]:

$$f(x) = \max(0, x) \tag{2.4}$$

Architecture of artificial neural network

An artificial neuron network (ANN) is a computational model that is based on the architecture and operations of biological neural networks. The information that passes through a neural network changes the structure of the artificial neural network (ANN) because the network adapts, or learns, based on the input and output. The Artificial Neural Network Architecture typically consists of three layers: Layer 1: Input; Layer 2: Hidden; Layer 3: Output. Neurons in the input layer comprise the first layer. When data enters the network, it passes through each layer until it reaches the output layer.

Types of neural network

In this section, a short overview of three commonly used types of neural network architectures will be discussed: Feedforward Neural Networks (FNN), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN)

1. Feedforward Neural Networks(FNN) The most basic kind of neural network is a feedforward neural network, often known as a multilayer perceptron. The information goes in a single route from the input layer to the output layer, via one or more hidden levels. FNN shows a good performance in regression and classification in various types of fields. Because it understands complex relationships between input and output data [24].

2. Convolutional Neural Networks(CNN)

   For processing images, audio, and video, convolutional neural networks are an excellent choice. Convolutional layers in CNN are used to apply filters to the input data in order to learn various patterns, including edges, textures, and more intricate models. CNN revolutionized computer vision tasks such as object identification, image segmentation, and image classification because of its capacity to identify patterns. CNN is widely used in healthcare, robotics, and recently in self-driving cars [25].

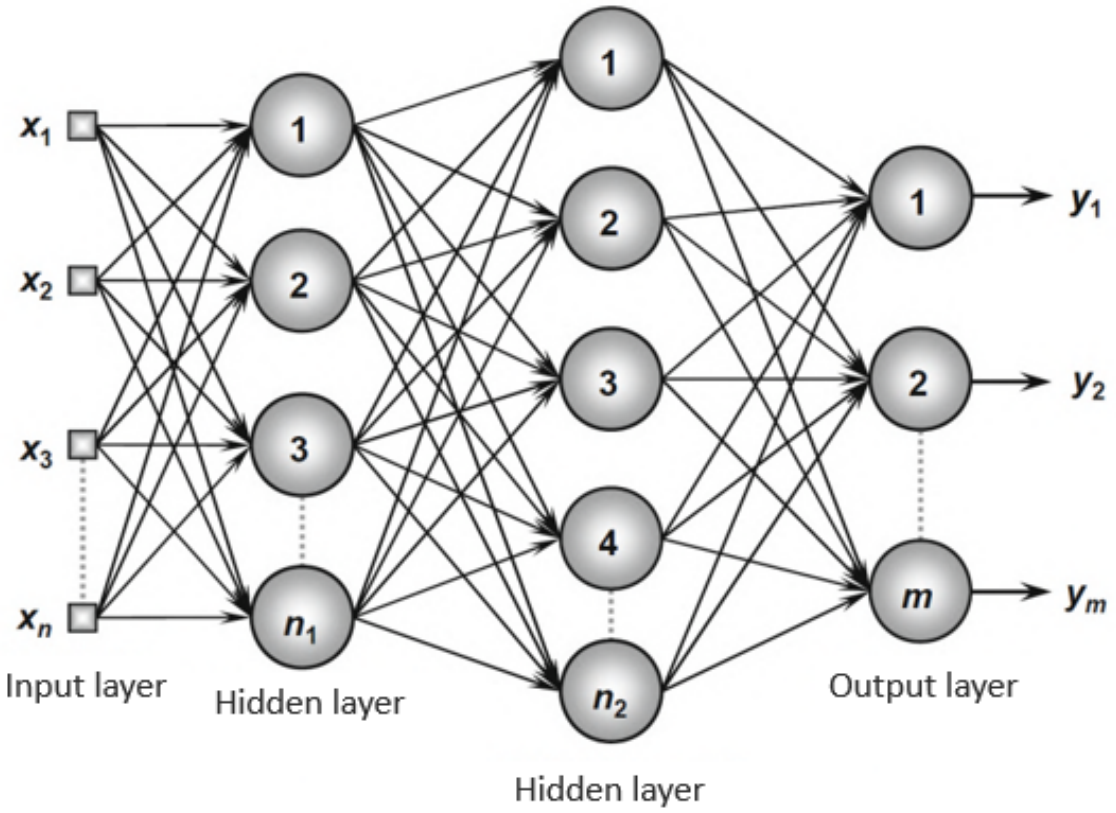

Figure 7: A feedforward multilayer perceptron architecture [26]

3. Recurrent Neural Networks(RNN)

   Recurrent Neural Networks (RNN) are a subclass of FNNs that are ideal for processing sequential input where information order matters. RNNs have circular links that allow them to store a complete state in an internal module, hence they can deal with temporal dependencies and context. The variant of RNNs, such as LSTM (Long Short-Term Memory), have addressed challenges like improved speech recognition and language translation [27].

2.3 Advances in Computer Vision

Artificial intelligence (AI)'s fast developing discipline of computer vision enables machines to comprehend and evaluate visual input from their surroundings. Using images from cameras, machines can extract meaningful information with the help of deep learning models. These developments have led to breakthroughs in various computer vision applications. The most recent developments in computer vision are examined in this section.

A. Image Classification

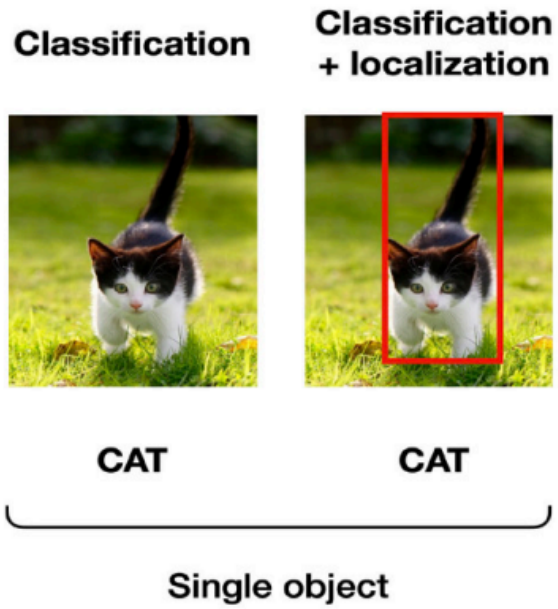

Figure 8: In Classification, the whole image is assigned a category "CAT". In Classification with Localization, the category of main object "CAT" and the bounding box around it is also provided [28].

In computer vision, one of the most basic jobs is image classification. Assigning a title or category to a whole image based on its visual content is the aim of image classification. As shown in the left image of Figure 8, classification identifies the main object within the image, such as a "CAT," without determining its precise location or size. The classification model processes the image as a whole and provides a single label for the dominant object. On the other hand, classification with localization not only labels the main object but also indicates where it is in the image. As shown in right image of Figure 8, it also draws a bounding box around the object. This advancement over basic classification allows for a more detailed understanding of image, combining recognition with spatial information about object's location.

B. Object Detection

Detecting instances of objects of a specific class inside an image is known as object detection [29]. Compared to image classification, object detection is more complicated task since it involves not only classifying all the objects inside an image but also localizing them with bounding boxes. In order to detect the all objects in single image, the model

must recognize the existence of many items in an image and forecast the categories and spatial locations of those objects.

Figure 9: In Object Detection, bounding boxes are drawn around identified objects and assigned a label of "CAT", "DOG" or "DUCK" [28].

As shown in the Figure 9, object detection draws bounding boxes around the identified objects. Each box encloses a specific object and assigns a label, such as "CAT," "DOG," or "DUCK." For applications like autonomous driving or robotic navigation, object detection is necessary. Because in these applications, it's crucial to know the position of several objects.

C. Image Segmentation

The process of dividing an image into several segments or regions in order to facilitate analysis is known as image segmentation. Assigning a label to each pixel in image so that pixels with the same label have similar properties is the aim of segmentation. Semantic and instance segmentation are two main categories of image segmentation.

1. Semantic Segmentation:

   In the semantic segmentation process, a label is assigned to every pixel in image [30]. The segmentation mask is created across pixels with same labels. Semantic segmentation treats multiple objects of same class as single entity and involves detecting objects within image and grouping them based on defined categories [30].

Figure 10: Every pixels of the image are assigned a labels of "CAT", "GRASS" or "TREE" in semantic segmentation. Segmentation mask with red color indicates instance of "CAT" class [28].

For example, as shown in the Figure 10, semantic segmentation labels all pixels belonging to "CAT," "GRASS," and "TREE" accordingly. All objects belonging to the same class are handled as a single entity in this task. This method is beneficial in the tasks where the individual identity of objects is not as important as identifying the object types within a scene.

2. Instance Segmentation:

   In the instance segmentation process, multiple objects of same class are treated as distinct individual objects or instances [30]. As depicted in the Figure 11, instance segmentation identifies each object separately, labeling multiple "CAT" or "DOG" instances individually. This allows for distinguishing between objects of same category within an image. Instance segmentation is critical in applications where object-level distinction is necessary, where system must differentiate between multiple instances of same object to interact appropriately with them.

Figure 11: In instance segmentation, all instances of the classes are assigned the labels of "CAT", "DOG" or "DUCK". Moreover, both the instance of same class "CAT" are assigned distinct segmentation mask [28].

2.4 Convolutional Neural Networks

One core class of deep learning models called Convolutional Neural Networks (CNNs) is mostly used for analysis of visual data. The advancements in Computer Vision tasks discussed earlier utilizes CNNs as a backbone to perform these tasks. CNNs have revolutionized various computer vision tasks such as image classification, object detection, and image segmentation [31]. Convolutional, pooling, and fully connected layers enable CNNs to automatically learn the spatial hierarchies of data, which is their main strength. CNNs are particularly good at tasks involving the recognition of edges, textures, shapes and other complicated features because of their architecture, which enables them to efficiently record local patterns and structures in images.

Convolutional Neural Network Layer and Architecture

As depicted in Figure 12, CNNs are comprised of three types of layers. These are convolutional layers, pooling layers and fully-connected layers [33].A CNN architecture is created when these layers are stacked. Together, these layers extract features from the input data, gradually decreasing its spatial dimensions without losing any of the most important information.

1. Convolutional Layer:
   The convolution layer is composed of a collection of **kernels** or **filters**, which extract

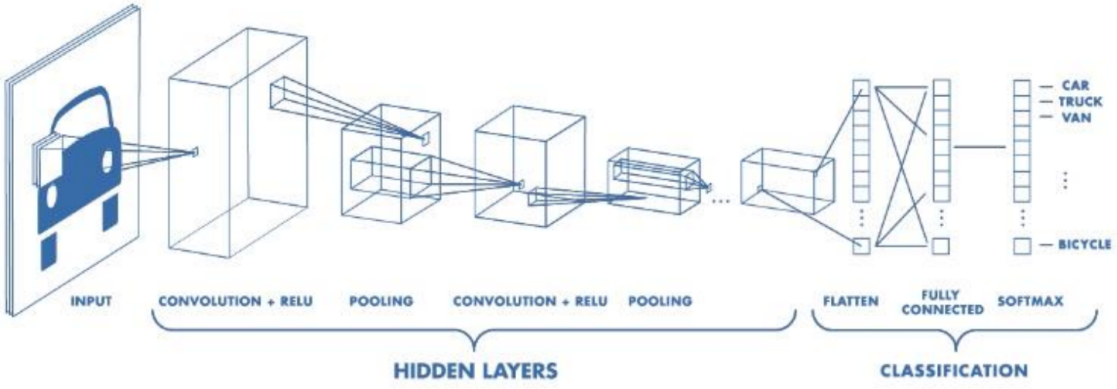

Figure 12: To identify the "CAR" class, CNN architecture performs several Convolution operation to extract relevant feature maps. These feature maps are passed to Pooling operation, so that the size of feature maps is reduced. Then feature maps are flattened and passed to FCN. The softmax function at the end calculates the possibility of "CAR" class. [32]

various features from different local regions of the input data [34]. Three dimensions make up each kernel: depth (D), width (W), and length (L). The length and breadth of the kernel in the convolution layer of a CNN are intentionally created; the kernel size is denoted by L × W. Typical dimensions are 3 by 3, 5 by 5, and so on. The CNN's capacity for feature extraction and computational complexity are directly impacted by the number of channels (D). CNN's capacity for feature extraction can be improved by adding more channels, but doing so also adds to its computational complexity [34].

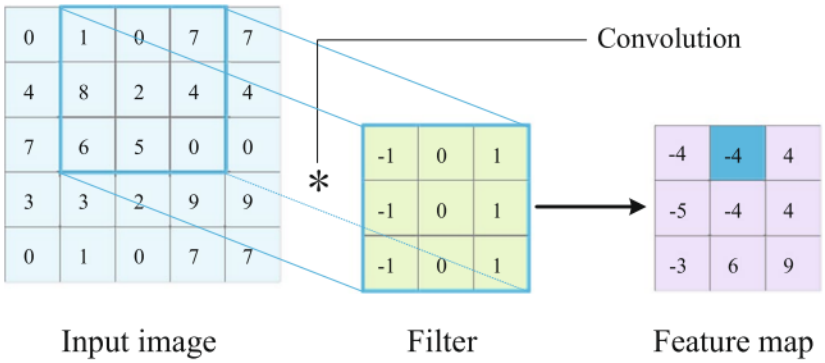

Figure 13: The Convolution Operation is performed on the input image with size of 5x5x1 and stride of 1. The kernel(filter) size is 3x3 matrix. The output feature map contains only important features [34].

A convolution operation is the process of sliding a convolution kernel (filter) over the input image, multiplying the convolution kernel and the pixel values at the corresponding positions of the input image, and summing them to obtain a feature map [34]. Figure 13 illustrates the convolution process with a 5 x 5 single-channel original image and a 3 × 3 convolution kernel. The corresponding pixel values of the original picture covered by the convolution kernel at the corresponding place are multiplied and summed to create each pixel value of the feature map produced by convolution. The -4 in the feature map's blue background in Figure 13 is computed using the following formula [34]:

$$(-4) = [(-1)\times1+0\times0+1\times7+(-1)\times8+0\times2+1\times4+(-1)\times6+0\times5+1\times0] \quad (2.5)$$

Stride

Stride is the number of rows and columns that the convolution kernel slides over the input matrix in order from left to right and top to bottom, starting from the top left of the input matrix [34]. For instance, the stride in both the height and breadth directions is 1 in Figure 13.

Moreover, larger stride can also be used in convolution operation. A convolution operation with a stride of 3 in the vertical direction and 2 in the horizontal direction is shown in Figure 14.

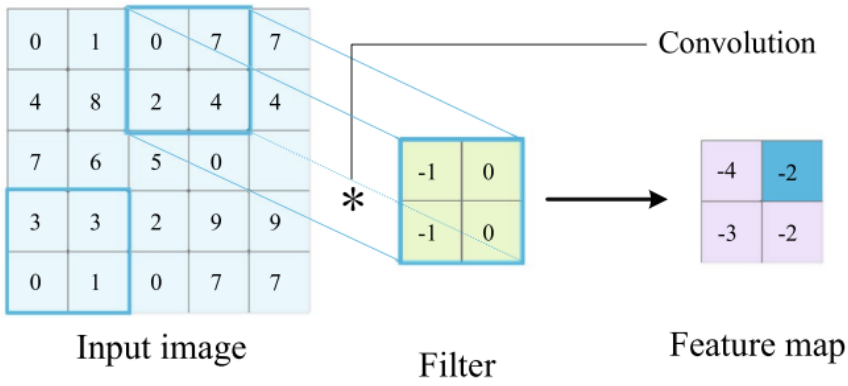

Figure 14: Convolution Operation is performed with the stride=(2,3) on the input image with the size of 5x5x1 and kernel size of 2x2. The large stride may lose the necessary information in feature maps [34].

The convolution window is moved down three rows to generate output of the second element of the first column. The elements that were employed in the calculation are [34]:

$$(-1) \times 3 + 0 \times 3 + (-1) \times 0 + 0 \times 1 = -3 \quad (2.6)$$

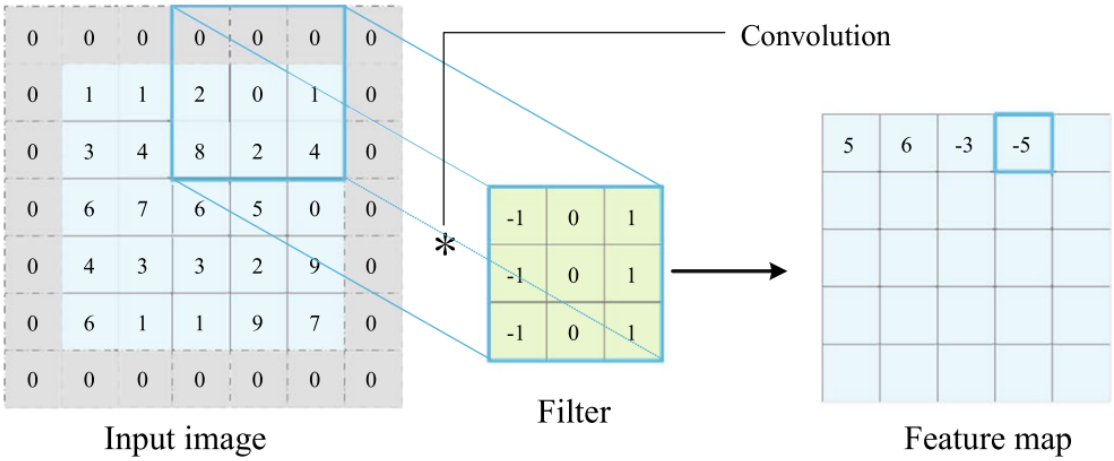

Figure 15: Zero Padding is applied on the Input image of size 5x5x1 during Convolution Operation with filter size of 3x3 and stride of 1. Thus the output feature map has the same input size and does not lose the important information of boundary pixels [34].

Upon output of the second element of the first row, the convolution window moves two columns to the right. The following components were utilized in the computation [34]:

$$(-1) \times 0 + 0 \times 7 + (-1) \times 2 + 0 \times 4 = -2 \tag{2.7}$$

The convolution window moves two more columns to the right on the input, but no result is generated since the input items cannot fit the convolution kernel window. The stride can affect the feature extraction capability, computational difficulty, and size of extracted features. As the stride grows, the computing speed increases but the output data size decreases and the ability to extract features weakens.

Padding

Padding is the process of adding a certain number of pixels to the edges of the input data so that the size of the output data can match the input data [34]. It is also known as padding some values on the matrix boundary to increase the size of the matrix, as illustrated in Figure 15. Typically, 0 is used, or the boundary pixels are copied for padding. CNN often uses padding to stop feature map sizes from getting smaller at each layer. Padding also facilitates the convolution kernel's learning of the details surrounding the input image. For instance, when the $5 \times 5 \times 1$ image is reinforced into a $7 \times 7 \times 1$ and applied to the $3 \times 3 \times 1$ kernel over it, the complex matrix is shown to be of dimensions $5 \times 5 \times 1$ [34]. It proves that the input and output images have the same dimensions. Should the identical process be carried out without any padding, the resultant image may be smaller. As a result, an image that is $5 \times 5 \times 1$ will be changed to a $3 \times 3 \times 1$ image.

2. Pooling Layer:

   This layer is similar to convolution layer but minimizes the size of feature maps and also the parameters of the network [35]. It is necessary to add a pooling (down sampling) layer after obtaining the feature maps. The pooling operation can be conceptualized as a weightless pooling function. This involves segmenting the input feature mapping group into many parts and combining each area to get a value that represents a generalization of that region. The two most used pooling functions are average and maximal pooling.

   Maximum pooling, which is typically employed for low-level feature extraction, chooses the highest activity value of all neurons as the representation of a region and extracts the most important characteristics from the input feature mapping. As seen in Figure 16, when maximum pooling (stride = 2) is used, a 2 × 2 kernel is moved over the matrix, and the maximum value is chosen and placed in the proper location of the output matrix. For example, pooling the four numbers '0, 1, 4, 8' in the blue region yields 8, the maximum of these four numbers [34].

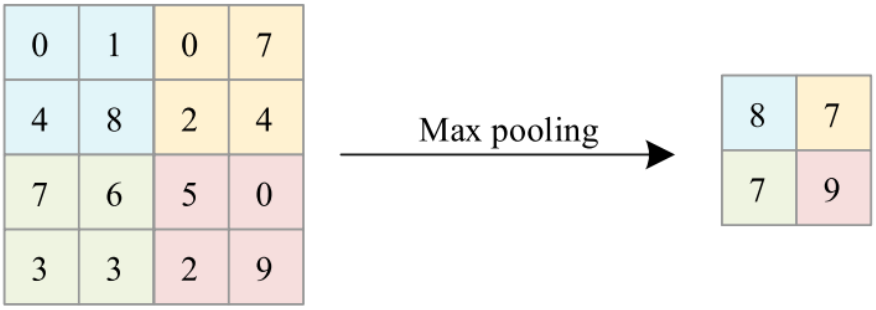

Figure 16: Maximum Pooling is performed on the feature map with size of 4x4 with the stride of 2. It reduces the output size of feature map to 2x2 [34].

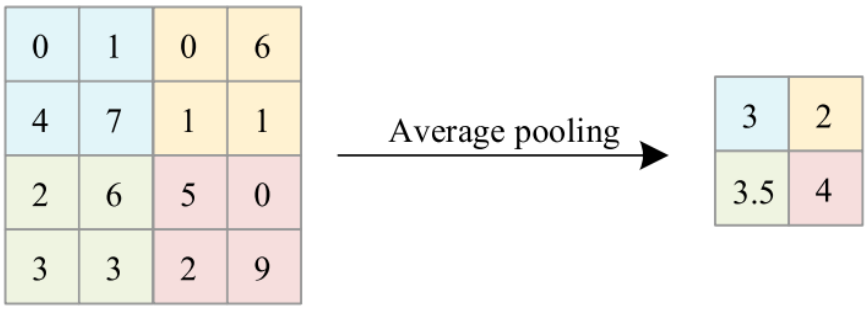

Figure 17: Average Pooling is performed on the feature map with size of 4x4 with the stride of 2. It reduces the output size of feature map to 2x2 [34].

The output result of average pooling is the mean value of the local response of the extracted feature mapping, which is the arithmetic mean of all elements in the region. Figure 17 displays the average pooling results with filter $= 2 \times 2$ and stride 2. It is clear that the pooling result for the green zone is $(2 + 6 + 3 + 3)/4 = 3.5$. By adding a pooling layer, the network efficiently reduces the input parameters, thus reducing the feature map dimension, and minimizing overfitting. Applying different pooling techniques also significantly shortens the time needed for model training and improves feature extraction and compression [34].

3. Fully Connected Layer:
   The resultant feature maps are flattened into a vector after multiple layers of convolution and pooling, and it is then passed through one or more fully connected layers. A feedforward neural network is comparable to the fully connected layer, as Figure 7 illustrates. The layer is often located in the network's bottom layer. It gets input from the convolutional output layer or final pooling and flattens it before sending it to the next layer. Ninety percent of the CNN parameters are found in the last layer. The feed forward network forms a vector of a particular length to follow up processing. Since these layers contain most of the parameters, there is a high computational burden while training the data [35].

2.5 Architecture of Image Segmentation Models

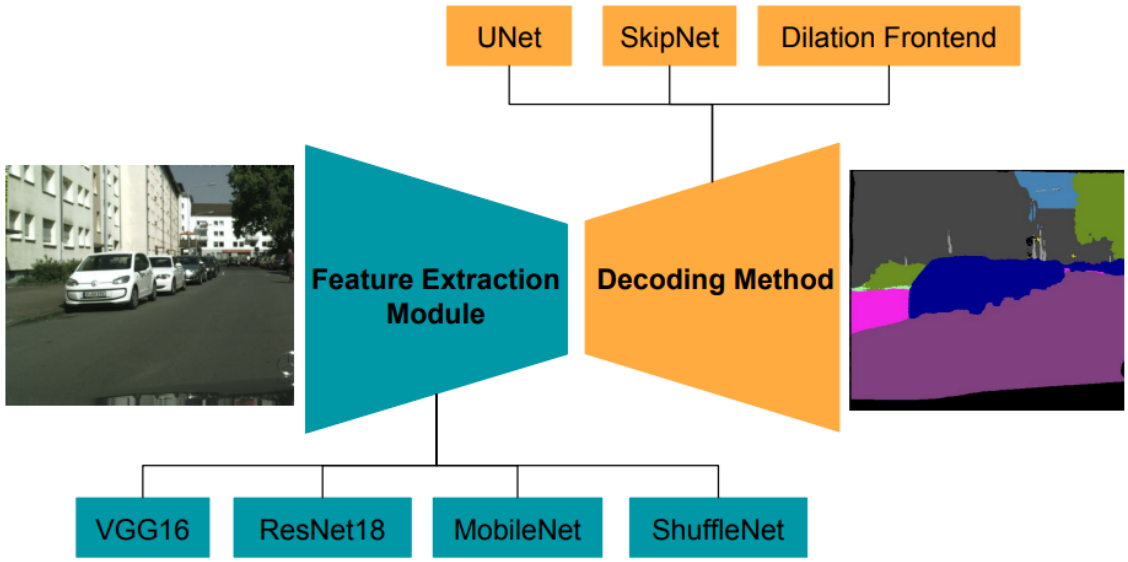

Figure 18: Overview of the different components in the framework with the decoupling of feature extraction module and decoding method [36]

As illustrated in figure 18, the architecture for semantic segmentation typically consists of two fundamental modules: the Decoding Method and the Feature Extraction Module. The input image is the first step in the process; it is supplied into the feature extraction module. High-level characteristics are necessary for distinguishing various objects and elements within the picture, and this module is in charge of extracting them from the image. Feature extraction can be performed using a number of popular architectures, such as VGG16, ResNet18, MobileNet, and ShuffleNet. These models are well-suited to handle the complex features required for segmentation tasks, and they have been widely employed in segmentation task.

Once the features have been extracted, the next step involves decoding them to generate a segmentation map. During the decoding phase, the architecture converts the high-level features into a classification map at the pixel level, designating each picture pixel with its corresponding category. Different decoding methods can be used to perform this task, such as UNet, SkipNet, and Dilation Frontend. These methods vary in terms of their complexity and ability to preserve spatial details from the original image.

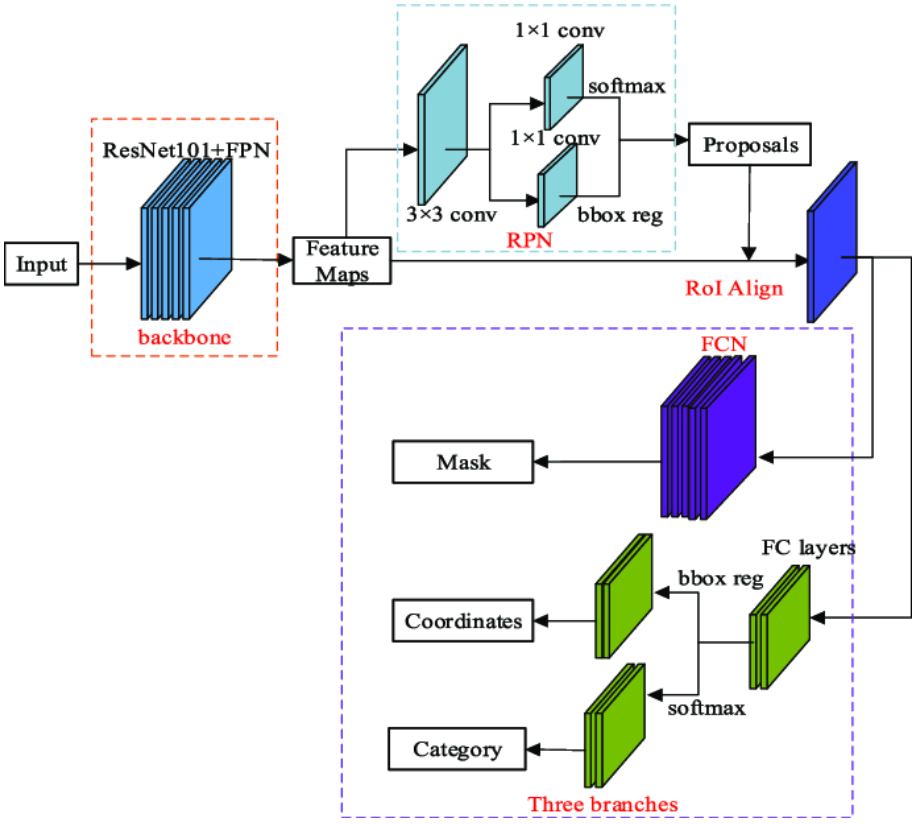

Figure 19: The Structure diagram of Mask-RCNN algorithm [37]

The purpose of instance segmentation models is to offer precise pixel-by-pixel masks for every instance of an object that is recognized, in addition to classifying objects within an image. In contrast, semantic segmentation treats all instances of the same object as a single class while classifying each pixel. As a result, the role of instance segmentation

models is more difficult because they have to distinguish between several instances of the same class in addition to segmenting the objects.

The Mask R-CNN (Region-based Convolutional Neural Networks) architecture is one of the most widely used instance segmentation models. It adds a branch to predict segmentation masks on each Region of Interest (RoI), extending Faster R-CNN's object detection capabilities. The architecture of Mask R-CNN can be understood in three key stages as shown in Figure 19. First, feature extraction is performed using a backbone network such as ResNet or VGG. The input image is passed through a Convolutional Neural Network (CNN), which generates a feature map, capturing crucial details necessary for identifying and segmenting objects [37]. Second, the Region Proposal Network (RPN) suggests regions in image that likely contain objects. These proposals are based on anchor boxes, and the regions with the highest "objectness" scores are passed on to subsequent layers.

The third important component is RoIAlign, which aligns these proposed regions with the feature map. RoIAlign corrects the slight misalignments that could occur due to down-sampling and up-sampling, ensuring that the regions being analyzed correspond accurately to the proposed object areas. This precise alignment is critical for the accurate localization of objects and their segmentation masks [37].

The model includes multiple heads for different tasks. One head is responsible for object classification and bounding box refinement, predicting what object class each region contains and adjusting the bounding box around it. The second head is the segmentation head, which predicts a pixel-wise mask for each object proposal. This mask is generated for every object proposal, giving high-precision, pixel-level segmentation that distinguishes one object from another [37].

2.6 Quantitative Comparison of Image Segmentation Models

In recent years, there has been significant development done to improve the image segmentation algorithms. While previous section discussed various architectures that form backbone of semantic segmentation, understanding how these models perform in practice is crucial. The model's computational efficiency and real-time applicability are determined by its segmentation accuracy. To assess the trade-offs across segmentation models in terms of accuracy, speed, and processing requirements, a quantitative comparison of the models is thus necessary. In order to highlight the advantages and disadvantages of modern segmentation models in diverse contexts, this section compares state-of-the-art models in depth using real data from numerous research investigations.

Siam et al. conducted a comprehensive comparative study on real-time semantic segmentation models using Cityscapes dataset [38]. The research evaluated several encoding architectures, including VGG16, ResNet18, MobileNet, and ShuffleNet, as well as different

decoding methods. (e.g., UNet, SkipNet, and Dilation Frontend). Additionally, comparisons were made with other segmentation networks, such as SegNet and DeepLab. Among the tested models, DeepLab emerged as superior approach, achieving the highest accuracy in terms of mean intersection over union (mIoU) [38]. But it is not computationally efficient. The study illustrated the benefits of DeepLab over alternative techniques for accurate urban scene segmentation and offered insightful information about trade-offs between model complexity and accuracy.

Following the previous discussion, Cao et al. conducted a study focused on the semantic segmentation of tree images using the YOLO v7 deep learning algorithm [39]. To address the difficulties in effectively segmenting trees in complicated backdrops, this study used tree samples from various contexts. The study employed a modified YOLO v7 model incorporating an attention mechanism and a weighted loss function to enhance segmentation accuracy, especially in cases of class imbalance [39]. Experimental results showed that the proposed Yolo v7 outperformed FCN, SegNet, U-Net, PSPNet and DeepLabV3+ algorithms in terms of both detection speed and segmentation precision [39]. This study demonstrates YOLO v7's potential as a useful tool for tasks requiring accurate and quick segmentation.

Based on the analysis presented in Varnyú and Szirmay-Kalos's paper titled "A Comparative Study of Deep Neural Networks for Real-Time Semantic Segmentation during the Transurethral Resection of Bladder Tumors" [40], this study undertakes a comparative analysis of various neural network architectures for semantic segmentation of bladder tumors from cystoscopic images. Eight state-of-the-art models—U-Net, UNet++, MA-Net, LinkNet, FPN, PAN, DeepLabv3, and DeepLabv3+ are assessed for their performance in terms of Dice score, classification accuracy, inference speed, and model size. The results indicated that DeepLabv3+ offered the fastest inference time at 6.73 ms and F-Score of 89.5522, highlighting its capability for real-time application [40].

In the study by Ruiz-Santaquitaria et al., different semantic and instance segmentation models were compared to detect microscopic algae [41]. The research evaluated several deep learning models like Mask R-CNN and SegNet to assess their performance in segmenting algae. The models were tested on dataset comprising various algae species captured under microscope. The study found that instance segmentation model like Mask R-CNN was more effective in distinguishing individual algae cells [41]. This comparison demonstrates the advantages and disadvantages of each method, offering insights into their suitability for tasks requiring either precise segmentation or broader detection coverage.

Prasetyo et al. conducted a comparative analysis of YOLO and Mask R-CNN to segment head and tail of fish using the Fish-gres dataset [42]. This dataset features images with diverse backgrounds, lighting conditions and instances of overlapping objects to provide challenging environment for segmentation tasks. With Mask R-CNN being a two-stage technique and YOLO representing a one-stage detector approach, the study sought to

assess the advantages and disadvantages of both models. Results showed that YOLO exhibited better overall efficiency and mean average precision (mAP). Thus it was more suitable for detecting distinct fish parts [42].

The paper "SSD vs. YOLO for Detection of Outdoor Urban Advertising Panels under Multiple Variabilities" by Ángel Morera et al. compared the performance of two popular object detection networks, SSD and YOLOv3 to detect advertisement panels in urban environments [43]. The study highlighted the strengths of each model under different conditions. YOLOv3 showed superior performance in detecting higher number of true positives and localizing panels more accurately [43]. Since there were no appropriate publicly accessible datasets for this task, both models were tested in variety of scenarios with varying panel sizes, occlusions and lighting levels using a constructed dataset.

The study conducted by Sapkota et al. compared the YOLOv8 and Mask R-CNN models for instance segmentation in agricultural applications, particularly focusing on segmenting tree trunks, branches, and immature apples [44]. The models were evaluated using two datasets, one from the dormant season (multi-class segmentation) and the other from the early growing season (single-class segmentation) in apple orchards. YOLOv8 outperformed Mask R-CNN across precision and recall metrics for both datasets. It achieved superior inference times of 10.9 ms for multi-class and 7.8 ms for single-class segmentation, compared to 15.6 ms and 12.8 ms for Mask R-CNN, respectively [44]. This demonstrated YOLOv8's effectiveness in real-time application.

As previously indicated, quantitative comparisons across multiple experiments show that YOLO performs better than other segmentation models, making it the best option for the robotic lawn mower application. YOLO demonstrates superior speed and efficiency over semantic segmentation models like DeepLab, which, although highly accurate, are computationally demanding and not ideal for real-time applications. Comparably, YOLO performs better than instance segmentation methods like Mask R-CNN, especially in terms of inference time and efficiency, which makes it more appropriate for situations demanding quick decisions and high frame rates. Additionally, YOLO's one-stage architecture offers a significant advantage over two-stage algorithms like SSD, providing faster detection while maintaining high precision.

YOLO's superior segmentation accuracy and processing economy make it a great choice for edge devices with constrained resources, such as the Raspberry Pi. By enabling rapid and accurate obstacle segmentation, YOLO enhances the safety and operational effectiveness of the robotic lawn mower, ensuring reliable performance in diverse and dynamic environments. We will examine the YOLO architecture in more detail in the next section, as well as how it has been evolved.

2.7 YOLO: You Only Look Once

YOLO (You Only Look Once) is a state-of-the-art algorithm for real-time object detection and segmentation. Introduced in 2015 by Joseph Redmon and collaborators, YOLO redefined object detection by framing it as a single regression problem [45]. YOLO is substantially faster than conventional methods since it completes both object localization and classification in a single forward pass of a convolutional neural network (CNN), as opposed to requiring separate stages for each operation.

YOLO (You Only Look Once) frames the detection process as a single regression problem rather than a series of classification tasks [45]. YOLO's design uses a single convolutional network to predict bounding boxes and related class probabilities for numerous objects simultaneously in a single evaluation, in contrast to more conventional techniques like R-CNN.

YOLOv1 detects all of the bounding boxes at once, streamlining the object detection process. In order to achieve this, YOLO splits the input image into a $S \times S$ grid and forecasts B bounding boxes of the same class for each grid element, along with its confidence for C distinct classes. Each prediction for a bounding box has five values: P_c, b_x, b_y, b_h, b_w, where P_c is the box's confidence score, which indicates the accuracy and model's level of confidence that the box contains an item. The box's centers in relation to the grid cell are indicated by the coordinates b_x and b_y, while the box's height and width in relation to the entire picture are shown by the coordinates b_h and b_w. The output of YOLO is a tensor of $S \times S \times (B \times 5 + C)$ optionally followed by non-maximum suppression (NMS) to remove duplicate detections [46].

The authors of the first YOLO article employed the PASCAL VOC dataset [47], which has 20 classes ($C = 20$), a grid of $7 \times 7 (S = 7)$, and a maximum of 2 classes per grid element ($B = 2$). This resulted in an output prediction of $7 \times 7 \times 30$.

A simplified output vector for eight values, a three-by-three grid, three classes, and one class per grid are shown in Figure 20. The result of YOLO in this simplified scenario would be $3 \times 3 \times 8$.

YOLOv1 achieved an average precision (AP) of 63.4 on the PASCAL VOC2007 dataset [46]. Real-time performance was made possible by YOLO's unique full-image one-shot regression and straightforward architecture, which made it far quicker than previous object detectors. YOLO has evolved significantly over time, with each version improving the model's precision, speed, and adaptability.

Non-Maximum Suppression (NMS)

As discussed earlier, YOLO uses NMS to remove duplicate detections. Non-maximum suppression (NMS) is a post-processing technique used in object detection algorithms to reduce the number of overlapping bounding boxes and improve the overall detection

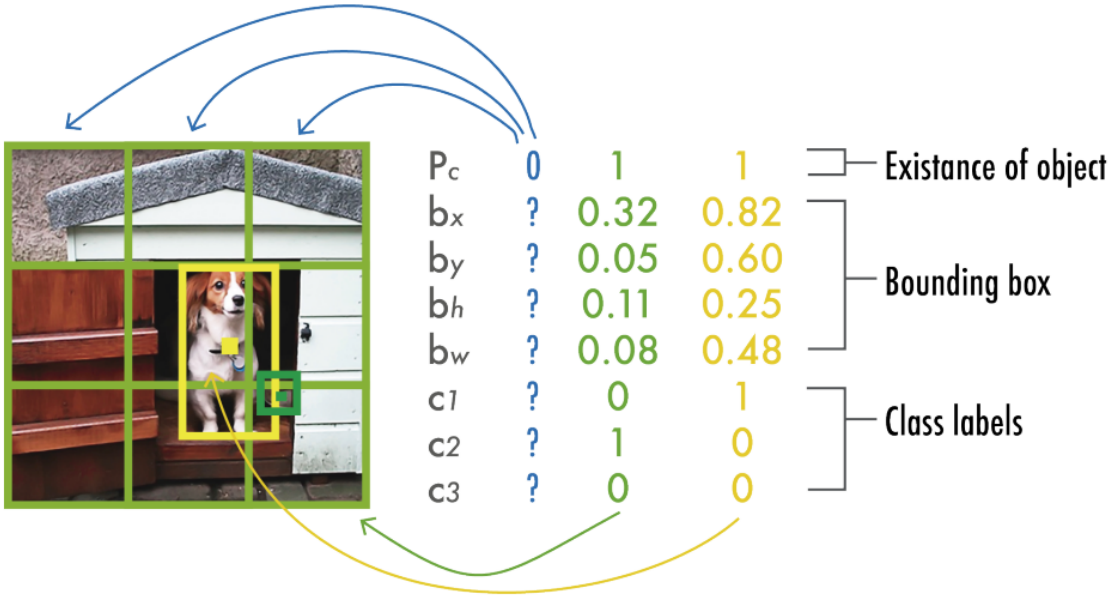

Figure 20: YOLO output prediction [46]

Algorithm 1 Non-Maximum Suppression Algorithm

Require: Set of predicted bounding boxes B, confidence scores S, IoU threshold τ, confidence threshold T
Ensure: Set of filtered bounding boxes F

$F \leftarrow \emptyset$
Filter the boxes: $B \leftarrow \{b \in B \mid S(b) \geq T\}$
Sort the boxes B by their confidence scores in descending order
while $B \neq \emptyset$ **do**
  Select the box b with the highest confidence score
  Add b to the set of final boxes F: $F \leftarrow F \cup \{b\}$
  Remove b from the set of boxes B: $B \leftarrow B - \{b\}$
  for all remaining boxes r in B **do**
    Calculate the IoU between b and r: $iou \leftarrow IoU(b, r)$
    if $iou \geq \tau$ **then**
      Remove r from the set of boxes B: $B \leftarrow B - \{r\}$
    end if
  end for
end while

Figure 21: Non-Maximum Suppression Algorithm [46]

Figure 22: Non-maximum suppression (NMS). (a) Typical output of an object detection model containing multiple overlapping boxes. (b) Output after NMS. [46]

quality [46]. Usually, object detection algorithms produce several bounding boxes with varying confidence scores surrounding the same object. NMS retains just the most accurate bounding boxes after eliminating superfluous and irrelevant ones. An algorithmic technique is described in Figure 21. The output following NMS and the typical output of an object detection model with numerous overlapping bounding boxes are displayed in Figure 22.

The version 8 of YOLO model was released by Ultralytics in January 2023 [48]. YOLOv8 has five scaled versions: YOLOv8x (extra-large), YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), and YOLOv8l (large) [49]. Numerous vision tasks, including object identification, segmentation, pose estimation, tracking, and classification, are supported by YOLOv8.

YOLOv8 uses an anchor-free model with a decoupled head to process objectness, classification, and regression tasks independently [48]. By allowing each branch to concentrate on its own role, this architecture enhances the overall accuracy of the model. The sigmoid function was employed as the activation function for the objectness score in the YOLOv8 output layer, which indicates the likelihood that an object is present in the bounding box. It uses the softmax function for the class probabilities, representing the objects' probabilities belonging to each possible class [48].

Additionally, YOLOv8 offers a semantic segmentation model known as the YOLOv8-Seg model. CSPDarknet53 feature extractor serves as the backbone, and in place of the conventional YOLO neck architecture, a C2f module comes next. The C2f module is followed by two segmentation heads, which learn to predict the semantic segmentation masks for the input image [48]. With great speed and efficiency, the YOLOv8-Seg model has produced state-of-the-art results on many object detection and semantic segmentation benchmarks.

2.8 Research Gap and Research Question

As discussed in the section 2.6, several studies have been done to test various segmentation models like Mask R-CNN, DeepLab, and YOLO in various fields such as medical imaging, autonomous systems and surveillance. The quantitative comparison also indicated that YOLO outperforms other segmentation models, making it the optimal choice for the autonomous systems. Still its use in real-time obstacle avoidance on Raspberry Pi 5 for robotic lawn mowers has not been tested extensively.
In several studies, segmentation models have been implemented for robotic lawn mowers to enhance their obstacle avoidance capabilities. For instance, the works of A. Pointinger (2019) [17] on autonomous lawn mowers have evaluated the effectiveness of segmentation models like DeepLabV3+, BiSeNet, GCN, PSPNet and DenseASPP. But their implementations relied on high-end hardware—GPUs, for example—which isn't appropriate for devices with limited resources, like the Raspberry Pi.
Thus, the primary research gap identified is the absence of studies exploring YOLO's segmentation capabilities for real-time obstacle avoidance in robotic lawn mowers, particularly those operating on hardware with limited computational power such as Raspberry Pi. One further area of this gap is the lack of analysis about the accuracy and frame rate of YOLO segmentation on platforms with limited resources.

Based on the identified research gap, the proposed master thesis aims to answer the following research question:

- **Which segmentation model optimally balances accuracy and computational efficiency among recent developments in Computer Vision?**
- **How can the identified segmentation model be implemented in a robotic lawn mower using Raspberry Pi 5 to enable real-time obstacle avoidance capabilities and ensure safety in a dynamic environment?**

Given the capabilities of YOLO demonstrated in previous sections, it has been established that YOLO excels in both speed and accuracy for segmentation tasks. Therefore, the focus now shifts to preparing the research design for the second question. The following sections will provide a detailed explanation of these elements.

3 Research Design/ Methodology

In the following section, the methodological approach that has been followed to develop and implement neural network is defined. Following this, how the dataset was prepared is discussed. The section then delves into different optimization techniques of YOLO model. It also discusses the evaluation metrics to outline the criteria for measuring model performance. Finally, a setup has been done to train and optimize YOLO model.

3.1 Overall Architecture

In a garden environment, robotic lawn mowers encounter various potential obstacles, such as chairs, tables, furniture, sports equipment, as well as living beings (e.g., humans, cats, dogs and small animals like hedgehogs). To train effective model in such a setting, the possible classes of these obstacles must be defined. However, establishing individual classes for each type of obstacle is considered impractical. Because it requires the extensive size dataset and it might increase model complexity and model size. To address this challenge, three primary classes have been identified: Lawn Area, Critical Objects, and Background. The "Lawn Area" class designates regions where mower can be permitted to operate and perform mowing operation. "Critical Objects" class includes living beings such as humans, cats, and dogs. They may present a potential risk of injury, so the mower might be programmed to detect and avoid to go closer to them by altering its path accordingly. Finally, "Background" class includes stationary items like chairs, furniture and also rest of the image pixels. They are considered low-risk for injury, thus the mower can be set to navigate around them to mow as closely as possible without colliding them. This approach optimizes data collection needs for this thesis.

The overall architecture for this research, as depicted in Figure 23, follows a structured approach for image segmentation using transfer learning. It includes stages from dataset preparation to model deployment, ensuring efficient model optimization for real-time performance. Each stage is designed to build upon the previous one, leading to an optimized deployment for Raspberry Pi.
The initial stage, dataset preparation, involves three key tasks: data collection, data annotation, and data splitting. The Raspberry Pi 5 and Camera Module 3 Wide were used to collect the images. Then this images are annotated with precise segmentation masks. In order to avoid over-fitting during the model training, dataset is divided into 3 sets: training, validation, and test sets. As discussed earlier in the section 2.6, the YOLO model is suitable in robotic lawn mower for it's optimal accuracy and computational efficiency. Thus in the model training stage, pre-trained YOLOv8s segmentation model is selected for transfer learning and then fine-tuned using the prepared dataset. Several experiments are carried out with different input image sizes in order to assess how image resolution

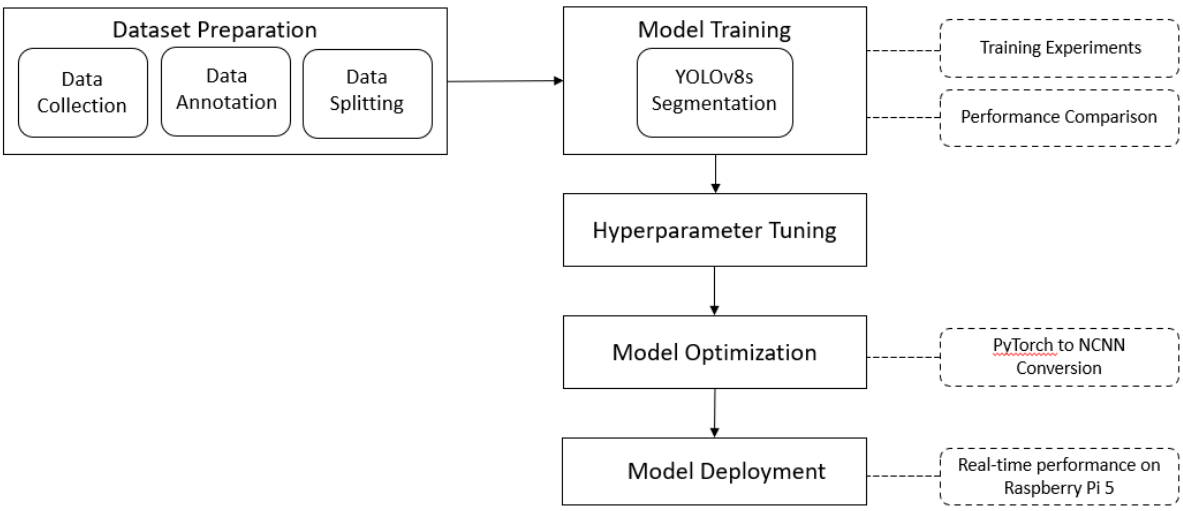

Figure 23: The methodological approach has been defined in different stages to design image segmentation model, optimize it and then deploy it in robotic lawn mower using Raspberry Pi 5.

affects model performance. The best-performing configuration is selected based on the defined evaluation metrics. Following the initial training phase, hyperparameter tuning is conducted to further optimize the model. By taking this step, it can be sure that the model is optimized to maximize accuracy while preserving computing efficiency. To verify improvement, the modified model's performance is assessed once more using the same evaluation metrics.

Once the model is fine-tuned, following stage focuses on model optimization for deployment on Raspberry Pi. The trained model is converted from PyTorch to NCNN framework, which is specifically designed for efficient inference on edge devices such as the Raspberry Pi. Model deployment is architecture's last phase. To evaluate the optimized model's real-time performance, it is deployed on Raspberry Pi 5. Key evaluation metrics are monitored during deployment in order to assess the model's performance within specified operating condition. Additionally, the accuracy of the model under real-world conditions is checked to make sure it fulfills the defined performance metrics set forth in the research.

3.2 Dataset Preparation

This section delves into the discussion of how the dataset has been prepared. The collection of data using ROBOLINHO, annotation of these data with ROBOFLOW tool and then data preprocessing have been discussed in detail.

Specification	Details
Sensor	Sony IMX708 wide
Resolution	11.9 megapixels
Sensor Size	7.4 mm sensor diagonal
Pixel Size	1.4 μm × 1.4 μm
Horizontal/Vertical Resolution	4608 × 2592 pixels
Focus Range	5 cm–∞
Focal Length	2.75 mm
Diagonal Field of View	120 degrees
Horizontal Field of View	102 degrees
Vertical Field of View	67 degrees

Table 1: Specifications of Camera Module 3 Wide [50]

3.2.1 Data Collection

As seen in Figure 24, the dataset for this study was gathered using a Raspberry Pi 5 equipped with the Camera Module 3 Wide on Robolinho. A total of 2176 pictures with a 16:9 aspect ratio and a resolution of 4608 × 2592 pixels were taken. The specifications of Camera Module 3 are mentioned in the Table 1 [50]. The data collection took place in the garden of AL-KO Company, capturing images under diverse environmental conditions to ensure model robustness.

The camera resolution of 4608 × 2592 pixels with Camera Module 3 Wide is well-suited to the requirements of this study. Because it provided the coverage of approximately 120 cm in front of the mower. Even though it left a blind spot of only 7-8 cm, this range is sufficient for detecting obstacles within the mower's field of view. Thus the mower could have enough visual information for accurate and timely decision-making. Additionally, the computational capabilities of the Raspberry Pi 5 are sufficient to support the processing of a YOLO model. This setup balances the needs for high-quality data capture, field-of-view adequacy and computational efficiency. So this setup is sufficient for the requirements of this study.

The AL-KO mechanical team put up the camera position with great care to guarantee that it covered the maximum possible field of view without compromising the mower's operational safety. This configuration allowed for comprehensive data collection while minimizing the risk to camera or surrounding objects.

To account for real-world variability, images were taken under different lighting conditions. Some images were taken in bright sunlight, while some were taken in shadows and low-light scenarios. The dataset also included instances where humans were present in the environment. To replicate varied light intensities, images were also taken at different times of the day. The garden environment featured obstacles such as trees, bushes, and different boundary lines to add further diversity in the image dataset.

Figure 24: ROBOLINHO with Raspberry Pi 5 and Camera Module 3 Wide

Due to the limited availability of images containing critical objects in natural settings, a small portion of the dataset was supplemented using an overlay technique. In this process, images of critical objects were sourced from open-access databases and then overlaid onto the images where there were only lawn area. As seen in the Figure 26, the image of hedgehog has been overlaid onto the image with lawn area. This method provided a wider variety of scenarios with critical objects in the dataset while maintaining a consistent and realistic environment.
As a result, the dataset comprises both naturally captured images and overlaid images. This comprehensive and diverse dataset forms the foundation for training and optimizing the image segmentation model. Thus it increases model's ability to generalize across various environmental conditions.

Data Collection Script

The Python script is depicted in Figure 25 below to get highlight of how the automated image capture was implemented. The data collection process was automated using a Python script, which was run on the Raspberry Pi 5 to access the Camera Module 3 Wide. The script could capture images at 2-second intervals over a predefined duration of one hour. The images were saved in JPG format to preserve quality. The captured images were stored in a designated directory with unique timestamps so that suplicated images can be easily avoided in preparing dataset.

3.2.2 Data Annotation

the data annotation was a crucial step in preparing the dataset. It required an accurate labeling of objects within each image. The primary objective was to categorize images

```python
import time
import subprocess

total_time = 3600

# Time interval between each capture (in seconds)
interval = 2

# Directory where the images will be saved
base_directory = "/media/pi/AL-KOMehrke1/data"

start_time = time.time()
while True:
    elapsed_time = time.time() - start_time
    if elapsed_time > total_time:
        break

    timestamp = time.strftime("%d%m%Y-%H%M%S")

    cmd = f"libcamera-still -t 1 -o {base_directory}/{timestamp}.jpg --rotation 180 -n"

    subprocess.call(cmd, shell=True)

    time.sleep(interval)

```

Figure 25: Data Collection Script

into distinct groups or classes to allow the model to differentiate between the areas where the mower should operate and those that require special attention.

As discussed earlier, there are three primary classes that have been identified: Lawn Area, Critical Objects, and Background. So the image frame was divided into these three classes: Lawn Area, Critical Objects and background as depicted in Figure 26. This categorization was essential for facilitating effective path-planning during the mower's operation. For critical objects, the mower should take evasive actions, such as turning in another direction to ensure safety. However, for non-critical objects like garden furniture or obstacles that do not pose a danger, the mower should attempt to navigate as close as possible to perform mowing tasks efficiently.

Lawn Area

This category included all visible lawn areas within the image frame. In Figure 26, yellow mask represents instance of class "Lawn Area".It was crucial to accurately label the lawn regions to help the model distinguish between the area that the mower should operate on and the other elements in the scene.

Critical Objects

This categories includes objects that could be harmed by the mower's blades, such as animals (e.g., dogs, hedgehogs) and humans, specifically parts such as bare hands or feet.

Figure 26: Labelled classes of image

In Figure 26, purple mask represents instance of class "Critical Objects". Accurate labeling of these objects is crucial to ensure that the segmentation model can help the mower avoid collisions and prevent accidents. When critical objects are detected, the mower should adjust its path to steer clear of them, enhancing the overall safety of the system.

Background

This categories comprises all other objects and image pixels that do not pose a risk to the mower's operation, including items like garden furniture, trees, or bushes. These objects, although obstacles, should be navigated closely by the mower to ensure effective mowing coverage. This approach allows the mower to follow its designated path while maintaining operational efficiency.

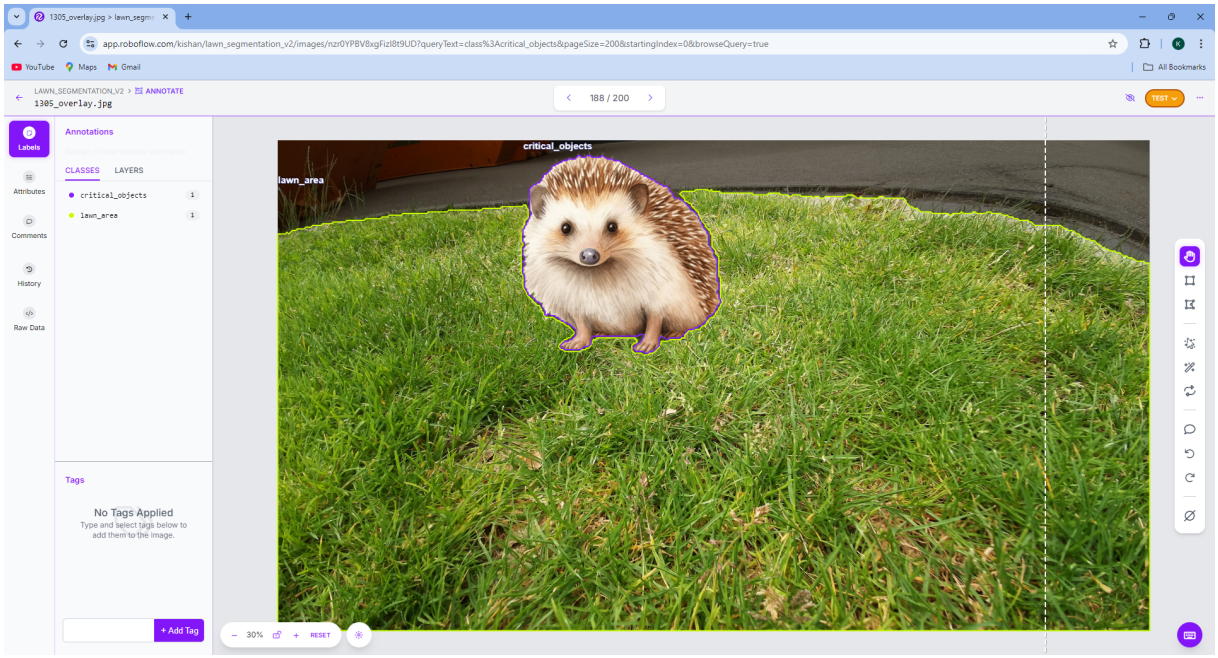

Figure 27: Data Annotation using Roboflow Tool [51]

The annotation process was performed using the Roboflow annotation tool [51], which enabled precise labelling of objects in each image. The tool supported manual annotation by allowing the user to highlight each object and assign it the appropriate class. The output of this annotation process was stored in a text file (.txt) format, with each image corresponding to a separate file. The text files contained detailed information about the location and classes of each annotated object in pixel coordinates.
An example of the annotated data using Roboflow tool is provided in Figure 27, where a hedgehog is labeled as a instance of "Critical Objects" class within the image. The Lawn is labeled as a instance of "Lawn Area" class and rest of the image pixels are automatically labeled as a "Background" class.

3.2.3 Data Preprocessing

The main preprocessing effort in this study was to separate the dataset into three sets: test, validation, and training. To ensure generalization and avoid over-fitting, this section makes sure the model is tested, evaluated, and trained on separate subsets of the data. Moreover, even though the current dataset contained only 2176 images, the number of images are constantly increased. In future,this would help the model to learn more from it and thus increase the accuracy.

Training set (64%): Out of total 2176 images, the model was trained on 1,392 pictures. Using this set helps the model learn by adjusting its weights according to the input data and labels.
Validation set (18%): 392 pictures were set aside for validation. Throughout the hyperparameter tuning phase, the validation set is used to track the model's performance and adjust the hyperparameters. By guaranteeing that the model's learnt patterns extend beyond the training set, it helps avoid overfitting.
Test set (18%): 392 photos were reserved for final testing. The test set is only used to assess the model's performance after training is finished, and it is entirely isolated from the training and validation procedures. This set offers an unbiased evaluation of the model's performance using unseen data.

Importantly, the data was split in a stratified manner, ensuring that each class was equally distributed across the training, validation, and test sets. This ensures that the distribution of images across all classes—such as lawn areas, critical objects, and background—is consistent in each subset. Stratified splitting prevents class imbalance in any one set, which could otherwise skew the model's training and evaluation results. Random splitting could lead to one set having a disproportionate number of samples from a certain class, negatively affecting model generalization and performance.
The necessity to make sure the model generalizes successfully to new, unseen data is

the justification for keeping the training, validation, and test sets apart. If the model were evaluated on the same data it was trained on, it could lead to overfitting—where the model performs exceptionally well on the training data but fails to generalize to new data [52]. By using distinct validation and test sets, a model's performance can be assessed objectively, ensuring that it has learned meaningful patterns rather than simply memorizing the training examples.

3.3 Selection of YOLO Variant and Transfer Learning

As discussed earlier in the section 2.7, YOLOv8 model provides five scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) and YOLOv8x (extra-large). In this study, pretrained YOLOv8s segmentation model from Ultralytics has been selected. Specifically, it is hypothesized that YOLOv8s segmentation would deliver faster training and convergence without sacrificing much accuracy. The rationale for this hypothesis is grounded in model's smaller size relative to other YOLO variants, such as YOLOv8l or YOLOv8x. While these large models potentially offer higher accuracy, it may require more computational resources during model training. Thus they may cause more cost for model training.

It was anticipated that the smaller YOLOv8 variant would achieve required Frames Per Second (FPS) and inference, because the model has to be deployed on Raspberry Pi 5. But Raspberry Pi provides low CPU capacity for computation of model. Therefore, the hypothesis is that YOLOv8s model will outperform larger models in terms of real-time processing speed and still maintain sufficient accuracy for image segmentation tasks. This hypothesis will be tested through evaluation metrics.

In this situation, transfer learning is essential to improve YOLOv8s segmentation model's accuracy and reduce training time. Transfer learning aims at improving the performance of target learners on target domains by transferring the knowledge contained in different but related source domains [53]. Pre-training on a large and varied dataset (e.g., COCO or ImageNet) provides model with generic features that are well-suited to particular task. This method greatly minimizes amount of time and computation resources required for training. It is helpful in situations when there is lack of labeled dataset. Moreover, transfer learning provides YOLOv8s model competitively accurate in segmentation. Thus it makes a viable option to choose for Raspberry Pi deployment. Consequently, this dual strategy of selecting a smaller model alongside the application of transfer learning is anticipated to optimize both processing speed and segmentation performance.

3.4 Optimizing YOLO Model: Selecting Effective Technique

For the deployment of trained YOLOv8s segmentation model on Raspberry Pi 5, model optimization was a critical step. Because Raspberry Pi 5 contains the limited computational

constraints, which might not be sufficient for the model to compute. There are several techniques for optimizing YOLO model, such as Pruning, Quantization or export in different format (e.g., NCNN, ONNX, TFLite).
To achieve this, the model was exported to the NCNN framework, a high-performance neural network inference computing framework specifically optimized for mobile and edge platforms [54]. NCNN was selected for this study due to its exceptional performance on mobile and edge devices. It offers several advantages that make it well suited for real-time applications. NCNN is a lightweight, cross-platform framework developed with a focus on mobile platforms, such as Android and iOS [55]. It is designed to function efficiently without third-party dependencies. So it allows for faster execution on devices with limited computational resources (e.g., Raspberry Pi).
Several features of NCNN contributed to its selection for optimizing YOLOv8s model:

- **Cross-Platform Compatibility**: NCNN is a pure C++ implementation and supports a wide range of platforms, including Android and iOS, as well as embedded systems like the Raspberry Pi [54]. Thus the optimized model can be deployed across various devices with minimal adjustments.

- **Efficient CPU Utilization**: NCNN has been optimized at assembly level, particularly for ARM NEON technology, which is widely used in mobile CPUs. This optimization ensures that model runs extremely fast on ARM-based devices, such as Raspberry Pi's CPU, without requiring external libraries or frameworks for acceleration [54].

- **Low Memory Footprint**: NCNN is known for its sophisticated memory management and data structure design, which results in a very low memory footprint. This feature is crucial when deploying models on devices with limited memory resources, such as mobile phones or embedded systems [54].

After exporting the model to NCNN framework, the model was refined for effective real-time inference. It could be deployed flawlessly on the Raspberry Pi with negligible performance losses. The model's smaller size and lower computational requirements by NCNN format provided fast and efficient operation in resource-constrained environments.

3.5 Defining Requirements and Evaluation Metrics

When deploying a model on Raspberry Pi in lawn mower, it is critical to establish requirements to evaluate performance of the developed model. So that the safety and operational efficiency can be evaluated. The evaluation metrics and the reasoning for choosing particular requirements are described in this section and are shown in Table 2. It is crucial to focus on following key evaluation metrics in order to assess the effectiveness

of developed model: Mean Average Precision (mAP), Minimum Safe FPS, Inference Time, and Recall. These metrics provide a thorough evaluation of the model's precision, effectiveness and instantaneous obstacle detection and segmentation capabilities.

No.	Requirement	Target Value
1	Safe FPS	$\geq$ 7.5 FPS
2	Inference Time	$\leq$ 133 ms
3	mAP50-95 (box)	$>$ 44.6 %
4	mAP50-95 (mask)	$>$ 36.8 %
5	Recall (mask-critical objects)	$>$ 96 %

Table 2: Requirements Checklist

3.5.1 Safe FPS

Frames Per Second (FPS) is one of the most important metrics for real-time applications. Basically, it means number of frames the model processes per second during inference. The minimum safe FPS for this application was determined by taking into account safety concerns, the speed of the mower, and the required reaction time.

The maximum speed of lawn mower was 0.97 km/h, which is equivalent to 0.269 meters per second (m/s). To prevent lawn mower from colliding with obstacles, the system needs to process enough frames per second to detect objects, make decisions, and send control signals to the mower in timely manner.

This involves both the model's inference time and the time required by the system to control the mower. For this analysis, it was assumed that total reaction time consists of the model's inference time and the additional control time required by the system. Reaction time is the time required for lawn mower to detect an obstacle and take evasive action. Based on hypothetical data, the total reaction time was estimated to be 0.4 seconds (0.3 seconds for model inference and 0.1 seconds for system control).

Reaction Distance Calculation

The reaction distance means the distance the mower covers during the total reaction time. It was calculated using following formula:

$$\text{Reaction distance} = \text{Speed} \times \text{Total Reaction Time} \tag{3.1}$$

Substituting the values:

$$\begin{aligned} \text{Reaction distance} &= 0.269\,\text{m/s} \times 0.4\,\text{seconds} \\ &= 0.1076\,\text{meters} \end{aligned} \tag{3.2}$$

This means the mower would cover approximately 0.1076 meters before it can react to an

obstacle. To ensure the mower can process frames fast enough to detect obstacles within this distance, the minimum safe FPS was derived from the total reaction time.

Minimum Safe FPS

The minimum safe FPS is the inverse of total reaction time. It makes sure that the system processes at least one frame for each reaction time duration:

$$\begin{aligned} \text{Minimum Safe FPS} &= \frac{\text{Speed}}{\text{Reaction Distance}} \\ &= \frac{0.269\,\text{m/s}}{0.1076\,\text{meters}} \\ &\approx 2.5\,\text{frames per second} \end{aligned} \tag{3.3}$$

Therefore, the mower must process a minimum of 2.5 FPS to ensure that it can detect obstacles within reaction distance and take appropriate action. Still it is only possible when the mower identifies the static objects. However, the processing of 2.5 FPS by lawn mower may still pose a risk in dynamic environment. In dynamic environments, when some living being comes rapidly towards the lawn mower, there must be a consideration of safety factor.

Safety Factor Consideration

To account for real-world uncertainties and potential variations in operating conditions, a safety factor is applied to calculate Recommended safe FPS. A safety factor of 3 was considered reasonable for robust performance in diverse conditions. By applying this safety factor, recommended FPS is calculated as follows:

$$\begin{aligned} \text{Recommended FPS} &= \text{Minimum safe FPS} \times \text{Safety Factor} \\ &= 2.5 \times 3 \\ &= 7.5\,\text{frames per second} \end{aligned} \tag{3.4}$$

Thus, the system should be designed to achieve recommended minimum 7.5 FPS during inference. So that lawn mower will have sufficient frames to detect and react to obstacles within its path in a timely manner. This value accounts for both the system's reaction time and real-world uncertainties. It provides robust safety margin that reduces the risk of collisions. By maintaining this FPS, the model can operate effectively on the Raspberry Pi and meets the necessary real-time processing standards for obstacle detection and path planning.

This FPS requirement forms the basis of the first metric in the Requirements Checklist, as shown in Table 2.

3.5.2 Inference Time

Another important parameter to assess the real-time performance of robotic lawn mower is inference time. It describes how long it takes the model to analyze each frame and produce a prediction. In real-time systems, maintaining a low inference time is crucial for mower to make timely decisions when detecting and responding to obstacles.
The inference time requirement is closely tied to Frames Per Second (FPS) requirement established earlier. The system must process minimum of 7.5 FPS to ensure timely detection and response to obstacles. The inference time per frame is calculated as inverse of the FPS:

$$\begin{aligned} \text{Inference time} &= \frac{1}{\text{FPS}} \\ &= \frac{1}{7.5\,\text{FPS}} \\ &\approx 0.133\,\text{milliseconds} \end{aligned} \tag{3.5}$$

This means the model must complete inference in 133 milliseconds or less to achieve recommended FPS.

3.5.3 mAP50-95 (box)

A complete set of metrics that assess the segmentation model's accuracy and reliability in a variety of settings is needed to evaluate YOLOv8's performance. Among them, recall, precision and the Precision-Recall (PR) curve are essential metrics. Together, these variables help to calculate Mean Average Precision (mAP). This performance metrics is frequently employed in tasks involving object detection and segmentation.

Precision
The precision denotes the proportion of the retrieved samples which are relevant and is calculated as the ratio between correctly classified samples and all samples assigned to that class [56]. Mathematically, precision can be defined as [56]:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{3.6}$$

Precision is a critical metric in object detection because it evaluates the model's ability to avoid false alarms (incorrectly identifying objects). A high precision value indicates that the model has a low rate of false positives.

Recall
The recall, also known as the sensitivity or True Positive Rate (TPR), denotes the rate of positive samples correctly classified, and is calculated as the ratio between correctly

classified positive samples and all samples assigned to the positive class [56]. It is calculated as [56]:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{3.7}$$

In order to evaluate the model's capacity to identify every relevant object in the scene, recall is crucial. A high recall score means that most, if not all, of the things present can be identified by the model. However, a high recall may sometimes come at the expense of precision if the model produces more false positives in an attempt to detect all objects.

IoU

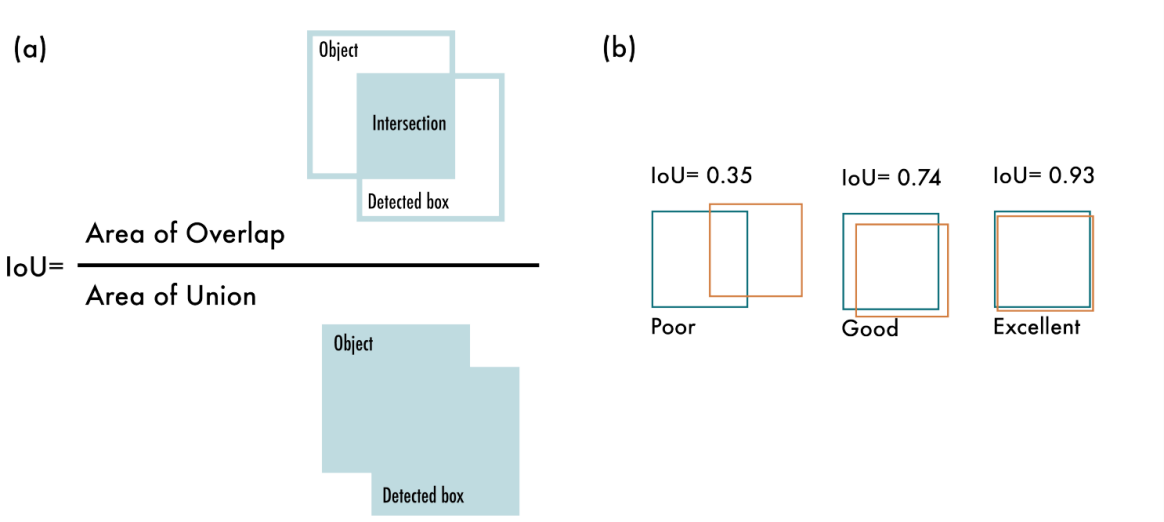

Figure 28: Intersection over Union (IoU). (a) The IoU is calculated by dividing the intersection of the two boxes by the union of the boxes; (b) examples of three different IoU values for different box locations. [46]

By anticipating bounding boxes, object detection and segmentation seeks to precisely locate items in images. The AP metric incorporates the Intersection over Union (IoU) measure to assess the quality of the predicted bounding boxes. IoU is the ratio of the intersection area to the union area of the predicted bounding box and the ground truth bounding box (see Figure 28) [46]. The overlap between the projected bounding boxes and the ground truth is measured.

Precision-Recall (PR) Curve

The Precision-Recall (PR) curve is a graphical representation that illustrates the trade-off between precision and recall for different confidence thresholds of the model [46]. The precision and recall values of the model fluctuate as the confidence threshold changes. The

PR curve provides the model's performance over a range of IoU thresholds.

Low recall combined with high precision suggests that model is conservative in its predictions, correctly identifying fewer objects overall. Low precision and high recall imply that the model identifies most items, but at the expense of producing more false positives. An overall impression of the model's performance can be obtained from the area under the Precision-Recall curve. Greater precision and recall across range of thresholds are indicated by a bigger area under the curve.

Mean Average Precision (mAP)

The AP metric is based on precision–recall metrics, handling multiple object categories, and defining a positive prediction using Intersection over Union (IoU) [46]. To analyze trade off between precision and recall, the AP metric incorporates the precision–recall curve that plots precision against recall for different IoU thresholds. This metric provides a balanced assessment of precision and recall by considering area under precision–recall curve.

In order to address this, the AP metric calculates the average precision (AP) for each class independently. Then the APs are averaged over all classes. So this metric is also known as mean average precision (mAP). This method guarantees that the model's performance is assessed for every class separately. Thus it provides more thorough evaluation of model's overall performance.

The AP for each class is calculated by integrating the PR curve and considering the precision at different recall levels. The mAP is then obtained by averaging the AP scores for all object classes in the dataset [57]:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{3.8}$$

where N is the number of object classes, and AP_i is the average precision for class i.

mAP50 refers to the mAP calculated at a single Intersection over Union (IoU) threshold of 0.51 [58]. By calculating whether the predicted bounding box overlaps the ground truth box by at least 50%, this metric assesses performance of the model. Even though mAP50 is helpful for quickly assessing performance, it might not accurately reflect model's performance at higher IoU thresholds.

mAP50-95 (box)

In contrast to mAP50, mAP50-95 $_{\text{(box)}}$ computes the mean Average Precision (mAP) for a range of IoU thresholds. IoU threshold starts from 0.50 to 0.95, in the increment of 0.05. This method makes sure that the model is assessed on its ability to anticipate bounding boxes that closely match the ground truth in addition to its performance in detecting

objects at a lower IoU threshold.
The formula for mAP50-95 $_{(\text{box})}$ can be represented as:

$$\text{mAP}_{50-95} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{T} \sum_{t=0.50}^{0.95} \text{AP}_{i,t} \tag{3.9}$$

where N is the number of object classes, T represents number of IoU thresholds, and $\text{AP}_{i,t}$ is average precision for class i at IoU threshold t.
The mAP50-95 $_{(\text{box})}$ metric evaluates quality of the bounding box predictions across range of IoU thresholds. So it provides more detailed evaluation of model's ability to accurately detect and localize objects. It is particularly useful in cases where a higher degree of overlap between predicted and actual bounding boxes is required.

In this study, the mAP50-95 $_{(\text{box})}$ metric was chosen with a threshold of 44.6 %, as this value corresponds to the performance of the YOLOv8s segmentation model on the widely-used COCO dataset [59]. The COCO dataset is a benchmark for both object detection and segmentation tasks [60]. By achieving a comparable mAP50-95 score ensures that the model deployed in this application maintains a high standard of accuracy.
The decision to use mAP50-95 over mAP50 alone is motivated by the need for a robust evaluation across varying IoU thresholds. By setting the requirement of mAP50-95 more than 44.6 %, the model's performance in detecting objects is expected to align with industry standards. So that accurate detection and segmentation of obstacles can be achieved in real-world environments.

3.5.4 mAP50-95 (mask)

In addition to evaluating model's performance in object detection using bounding boxes, it is crucial to assess its accuracy in segmentation, where the task involves segmenting objects at pixel level. For this purpose, the mAP50-95 $_{(\text{mask})}$ metric is used to evaluate the precision of the predicted segmentation masks.
The model's capacity to predict precise masks for objects across several Intersection over Union (IoU) thresholds, spanning from 0.50 to 0.95, is measured by the mAP50-95 $_{(\text{mask})}$ metric. This metric assesses how well the model captures precise boundaries of objects in image, providing a more detailed evaluation of its segmentation capabilities than bounding box-based metrics.

The threshold of mAP50-95 $_{(\text{mask})} > 36.8$ % was selected based on the performance of the YOLOv8s segmentation model on COCO dataset [59] as discussed earlier. Accurate segmentation masks are especially important for segmentation of small or irregularly shaped objects that may not be adequately captured by bounding boxes alone.

Thus, the mAP50-95$_{(\text{mask})} > 36.8$ % requirement ensures that the model's segmentation accuracy aligns with industry standards and provides the precision necessary for safe and effective operation in dynamic environment.

3.5.5 Recall (mask - critical objects)

For obstacle avoidance in robotic lawn mower, achieving high recall is crucial for ensuring that critical objects, especially those that pose safety risks, are correctly identified and segmented. Recall measures the ratio of the number of true positives to the number of positive ground truths [57]. In this context, recall refers to the model's ability to correctly identify all relevant instances of critical objects, such as animals or humans, within the scene. Specifically, the metric The recall of real critical objects that are accurately segmented by the model's predicted masks is measured by recall $_{\text{(mask - critical objects)}}$.

Recall is particularly important for critical objects since missing an object like a hedgehog or a small child could result in safety hazards. In this project, the Recall $_{\text{(mask - critical objects)}}$ requirement is set to be greater than 96 %, meaning the model should correctly identify at least 96% of all critical objects present in the environment.

This can be illustrated with an example: if there are 100 hedgehogs (a critical object class) present in the environment, the model should be able to segment at least 96 of them correctly. The remaining 4 may be classified as background, but under no circumstances should they be misclassified as lawn area where the robotic mower operates. Ensuring a high recall rate minimizes the risk of the mower failing to detect critical objects, thereby enhancing overall safety.

The decision to set the recall threshold at 96 % for critical objects is grounded in safety considerations. In the operation of an robotic lawn mower, missing even a small percentage of critical objects could lead to serious consequences, such as running over small animals or failing to avoid humans in the mowing area. High recall ensures that almost all critical objects are detected and segmented, which is paramount for obstacle avoidance and operational safety.

Achieving recall > 96 % for critical object masks ensures that the model's segmentation is highly reliable for detecting objects that must be avoided by the mower. While a recall of 100 % is ideal, the threshold of 96 % provides a balance between practical model performance and computational limitations, ensuring that most critical objects are captured without significantly increasing the false-positive rate.

By maintaining this high recall for critical objects, the model ensures that safety-critical decisions can be made in real-time, reducing the risk of collisions or accidents.

3.6 Configuration of YOLO Model Training and Optimization

In this section, the setup and procedures for training and optimizing the YOLO model are detailed. The discussion begins with the hardware configuration used to support the computational requirements of model training. Next, the specific YOLO training experiments is outlined. Following this, hyperparameter tuning of the trained model is reviewed. Finally, optimization technique applied to the tuned model are presented to improve efficiency and reduce computational load during deployment.

3.6.1 Hardware Configuration

The training experiments were conducted using AWS SageMaker with a p3.8xlarge instance. This instance included Tesla V100 GPUs with total 64 GB of GPU memory. The p3.8xlarge instance provided significant parallel processing capability. It is essential for training deep learning models efficiently when working with high-resolution images and large datasets. The choice of Tesla V100 GPUs allowed for rapid model training and experimentation due to their support for high-throughput computation and accelerated deep learning operations.

3.6.2 Setup for YOLO Model Training Experiments

Two sets of experiments were conducted to evaluate the performance of the YOLOv8s segmentation model on different image sizes. The goal was to understand the trade-off between input image size, model accuracy, and inference speed.

Parameters	**Experiment 1**	**Experiment 2**
Image Size	640 × 384 pixels	1280 × 736 pixels
Epochs	100	100
Batch Size	16 (Default)	16 (Default)
Learning Rate	0.001 (Default)	0.001 (Default)
Optimizer	AdamW (Default)	AdamW (Default)

Table 3: List of Parameters for Training Experiment 1 and Experiment 2

As mentioned in the Table 3, the first experiment used an input image size of 640 pixels in the horizontal dimension, allowing YOLOv8 to adjust the vertical size automatically based on the aspect ratio of the original images. This means that original resolution of the image (4608 × 2592 pixels) was reduced to 640 × 384 pixels. This setting was chosen to minimize the computational load while maintaining a reasonable resolution for detecting and segmenting objects. The second experiment was conducted with an input image size of 1280 pixels in the horizontal dimension, again allowing the vertical size to be calculated based on the aspect ratio. In this case, the original resolution of the image was reduced to

1280 × 736 pixels. The larger image size was tested to evaluate whether increasing the resolution improves segmentation accuracy, especially for small or distant objects.

For both experiments, the model was trained for 100 epochs and most hyperparameters were kept at their default values for these experiments. This included:

- Batch Size: A batch size of 16 was used for each experiment, balancing memory usage with model performance.
- Learning Rate: The learning rate was set to 0.001, which is a standard value for YOLOv8 models.

3.6.3 Hyperparameter Tuning of the Trained Model

After conducting the initial training experiments, further tuning of the model was performed to improve its performance. Based on the evaluation metrics derived in the section 3.5, the best model was selected for hyperparameter tuning. Following the selection of the model with the best accuracy, hyperparameters were adjusted to further optimize the model.

Parameters	**Values**
Image Size	640 × 384 pixels
Epochs	300
Batch Size	16 (Default)
Learning Rate	0.001 (Default)
Optimizer	AdamW (Default)

Table 4: List of Parameters for Tuning

The model trained with an input image size of 640 × 384 pixels was chosen for hyperparameter tuning, as it demonstrated a favorable trade-off between accuracy and computational load. During this tuning process, the number of training epochs was increased from 100 to 300 to allow the model to learn more complex patterns and better generalize on the dataset.
Other hyperparameters, including the batch size (16) and learning rate (0.001), were kept at their default values (Refer Table 4). This was done to ensure consistency and to maintain the balance between training time and model performance. By extending the training duration to 300 epochs, the model was expected to converge more effectively, especially for difficult segmentation cases, without overfitting to the training data.

3.6.4 Optimization of the Tuned Model to NCNN Format

Once the model was fine-tuned and trained with the optimal hyperparameters, the next step involved optimizing the model for real-time inference on the Raspberry Pi 5. To accomplish this, the tuned PyTorch model was converted to the NCNN framework as

```python
from ultralytics import YOLO

# Load the tuned model
model = YOLO("/v8s_640_tune/train4/weights/best.pt")
model.export(format = "ncnn")

```

Figure 29: YOLO PyTorch to NCNN Conversion Code

shown in the Figure 29. Optimization to the NCNN format was selected for this study due to its exceptional performance on edge devices as discussed in the section 3.4. Ultralytics provided the required package to export PyTorch model to NCNN format. This step was essential for reducing the model's computational requirements while maintaining its accuracy, enabling faster inference on the resource-constrained Raspberry Pi 5. After conversion, the model was ready for deployment, ensuring that it could run with low latency while adhering to the performance requirements established for the lawn mower's obstacle avoidance system.

4 Results

This section describes the results of YOLO model's development and deployment. This begins with an analysis of training experiment performance, followed by an evaluation of the impact of hyperparameter tuning on model accuracy using evaluation metrics. The section then examines the performance of the optimized model to highlight improvement in computational efficiency. Lastly, real-time performance on the Raspberry Pi 5 is reviewed to analyze model's operational effectiveness in real-world environment.

4.1 Performance of Training Experiments

The performance of two training experiments with YOLOv8s segmentation model was assessed using these evaluation metrics: mAP50-95 $_{(box)}$ and mAP50-95 $_{(mask)}$ for all classes, Recall $_{(mask)}$ for "Critical Objects" class. This section provides comparative results obtained from both experiments during validation and testing.

Validation Results of Training Experiments

Evaluation Metrics	**Experiment 1**	**Experiment 2**
mAP50-95 (box)	87.1 %	63.4 %
mAP50-95 (mask)	84.2 %	65.2 %
Recall (mask-critical objects)	87.8 %	85.4 %

Table 5: Validation Results of Training Experiments

The results of both experiments are summarized in Table 5. It compares the performance of models on validation data when the input image sizes are resized to 640×384 (Experiment 1) and 1280×736 (Experiment 2). It was expected that using larger image sizes in Experiment 2 would improve model performance due to the increased resolution, allowing the model to capture more detailed features. However, the validation results indicate that Experiment 1 outperformed Experiment 2 in terms of mAP50-95 $_{(box)}$, mAP50-95 $_{(mask)}$ and recall mAP50-95 $_{(mask\text{-}critical\ objects)}$.

This performance discrepancy may suggest that the model in Experiment 2 did not converge effectively. A possible reason for this could be that there were only 100 training epochs in each experiment. While this was sufficient for smaller input sizes, it may not have been enough for the larger image size in Experiment 2, which demands more computational resources and training time to effectively learn from higher-resolution data.

Test Results of Training Experiments

Experiment 1 exhibited improved test performance across all metrics when compared to the validation results (Refer Table 6). The mAP50-95 $_{(box)}$ increased from 87.1 % in validation to 89.8 % in testing, and mAP50-95 $_{(mask)}$ improved from 84.2 % to 87.9 %.

Evaluation Metrics	Experiment 1	Experiment 2
mAP50-95 (box)	89.8 %	67.1 %
mAP50-95 (mask)	87.9 %	68.1 %
Recall (mask-critical objects)	89.3 %	89.8 %

Table 6: Test Results of Training Experiments

The recall for critical objects slightly improved as well, indicating better object detection in unseen data.

Experiment 2 showed a similar trend, with mAP50-95 $_{(box)}$ increasing from 63.4 % in validation to 67.1 % in testing, and mAP50-95 $_{(mask)}$ rising from 65.2 % to 68.1 %. However, these improvements were marginal compared to Experiment 1. These findings imply that the model performs even better in testing than in validation, demonstrating good generalization to unseen images.

Both trained models were tested on an identical test image with a hedgehog that was identified as an instance of "Critical Objects" in order to access their efficacy even more. The two images below represent the predictions from the models trained on input sizes of 640 × 384 pixels and 1280 × 736 pixels, respectively.

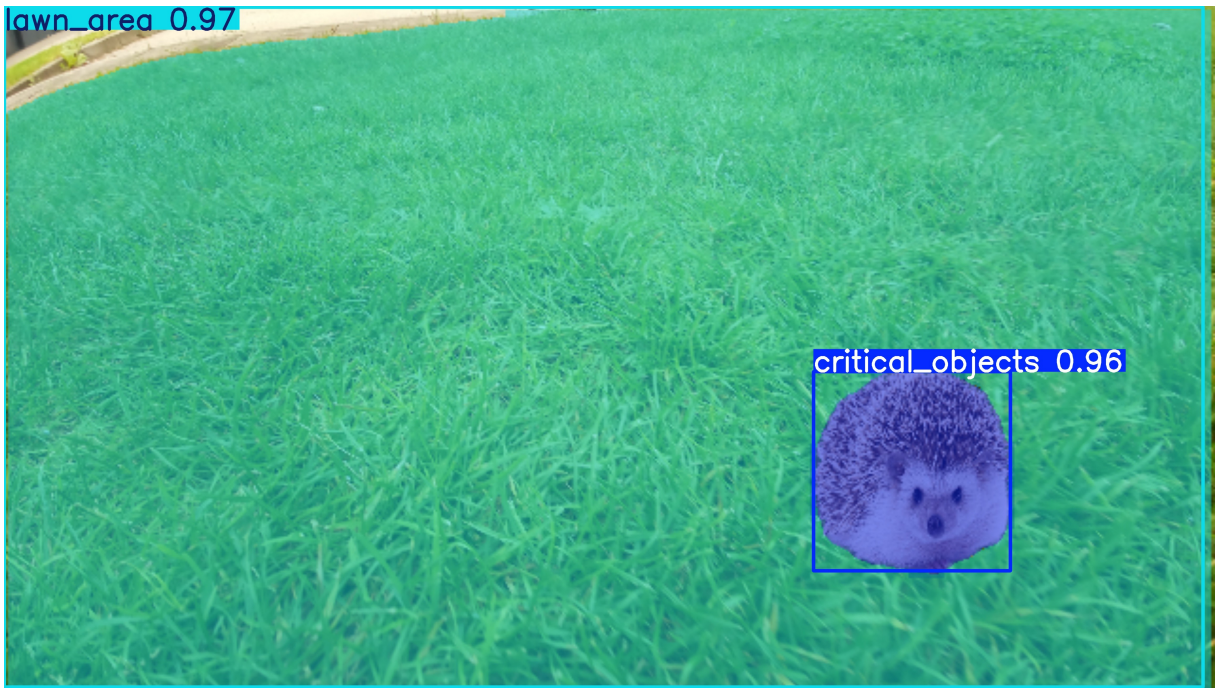

Figure 30: Prediction on Test Image with the model from Experiment 1

The model from experiment 1 accurately identified the hedgehog with a high confidence of 96 % for the "Critical Objects" class as depicted in the Figure 30. The "Lawn Area" class was segmented with a confidence of 97 %, effectively distinguishing between the lawn and the critical object. The hedgehog's segmentation borders were accurate, showing

Figure 31: Prediction on Test Image with the model from Experiment 2

that the model's capacity to distinguish between classes in this situation was unaffected by the lower image size. When the prediction was created with model from experiment 2, the hedgehog was detected as a "Critical Objects" with a confidence score of 96 %. However, the confidence score for "Lawn Area" was slightly lower at 80 %, indicating a slight variance in model behavior when using a higher resolution (see Figure 31). Moreover, the predicted lawn area was also not perfect compared to the same annotated image in test dataset. It only generated mask on approximately 70 % of the lawn area. The possible reason behind this might be that model in Experiment 2 did not converge within 100 epochs. While this 100 epochs were sufficient to converge during training for smaller input size, it may not have been sufficient for the larger image size in Experiment 2. Thus it might need more computational resources and training epochs to effectively learn from higher-resolution data.

4.2 Performance of Hyperparameter Tuning

As the model from Experiment 1 provided better results compared to the model from Experiment 2, it was selected for further hyperparameter tuning.

Training and Validation Losses

The first set of graphs presents the training and validation losses across various categories, including box, segmentation, classification, and distribution focal losses as shown in Figure 32. Both Box and Segmentation Losses exhibit a clear decreasing trend, indicating effective

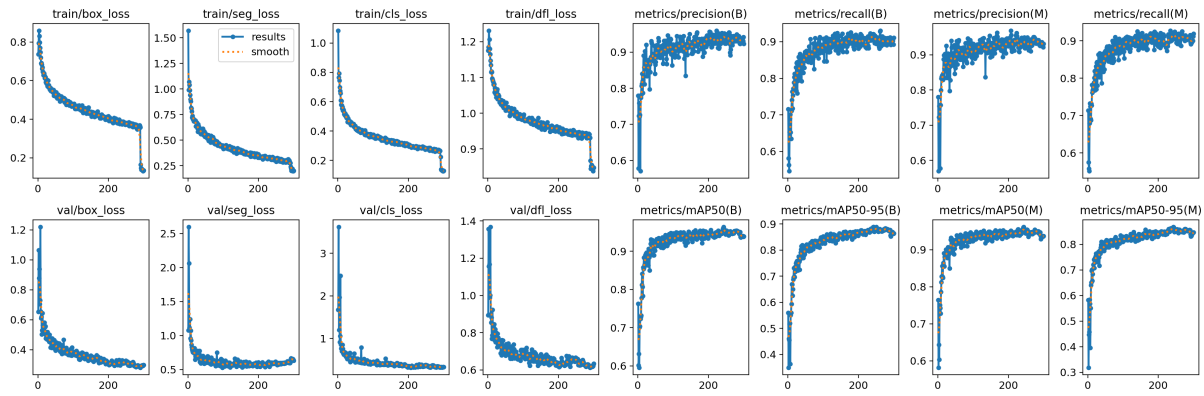

Figure 32: Training and Validation Losses

learning as epochs progress. The smooth reduction in these losses suggests the model successfully minimizes errors in object localization and segmentation. Classification and Distribution Focal Losses (DFL) decrease significantly over time, stabilizing towards the end. The decline in classification loss reflects improvements in object class prediction, while lower distribution focal loss indicates better handling of bounding box offsets.

During training, there was sudden drop of box loss, segmentation loss, class loss and DFL loss noticed after approximately 285 epochs. The reason behind this might be that the model might have converged after roughly 285 epochs. Thus it might have achieved maximum possible results during training process.

In the second set of graphs, illustrated in Figure 32, the metrics for precision, recall, mAP50, and mAP50-95 for both box and mask show a positive trend. All metrics are improving as the number of epochs increases. These improvements imply that as the model is exposed to more training epochs, it becomes increasingly accurate and reliable in detecting and classifying objects, both in terms of localization (boxes) and segmentation (masks).

Confusion Matrix Analysis

The confusion matrix illustrates the normalized classification accuracy for each class (Critical Objects, Lawn Area, and Background).

The model achieved an accuracy of 92 % for the "Critical Objects" class, indicating it correctly identifies these instances most of the time as shown in the Figure 33. With an accuracy of 94 %, the model distinguished the instances of "Lawn Area" class from other classes. However, some misclassifications are also occurred. 8 % of Critical Objects" instances are misidentified as "Background". While this error rate may still be acceptable for a lawn mower, as it would treat those objects as part of the background and avoid mowing over them.

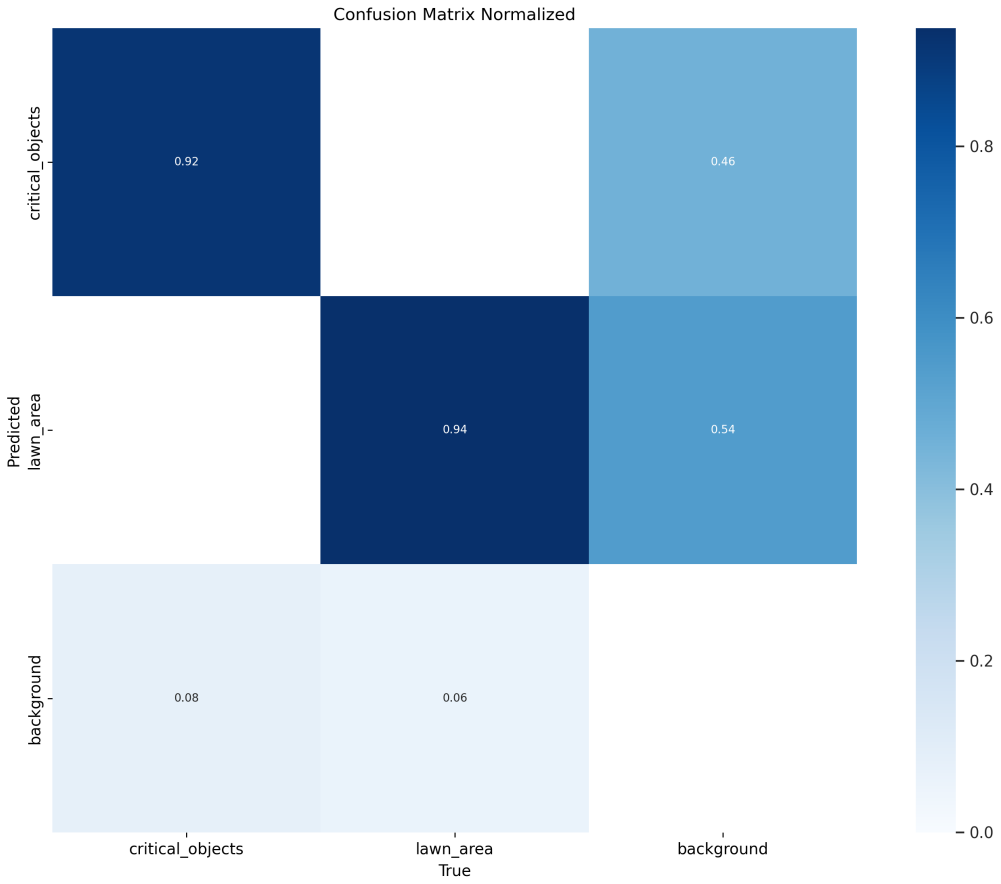

Figure 33: Normalized Confusion Matrix

Hyperparameter Settings

The hyperparameter tuning employed various settings as seen in the Figure 34. This configuration optimized the model's performance by fine-tuning parameters that affect training dynamics. Notably, warmup momentum and warmup epochs contributed to stabilizing early training phases, while parameters like hsv augmentation and scale provided data variability to enhance generalization. Moreover, rotataion (degrees), vertical flip (flipud), and shear parameters did not improve the results. So these parameters were kept to null.

Validation and Test Results after Hyperparameter Tuning

After performing hyperparameter tuning, the evaluation metrics demonstrate a clear improvement in the model's performance across both validation and test sets as shown in the Table 7. The mAP50-95 $_{(\text{box})}$ increased from earlier experiments to 88.8 % in validation and 90.2 % in the test results, indicating enhanced object detection accuracy. Similarly, the mAP50-95 $_{(\text{mask})}$, which measures the segmentation performance, improved to 86.5 % in validation and 88.2 % in the test set.

Additionally, the recall for the mask-critical objects also shows improvement after tuning,

```
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
box: 7.5
cls: 0.5
dfl: 1.5
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
bgr: 0.0
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
```

Figure 34: Best Hyperparameters after Tuning

Evaluation Metrics	**Validation Result**	**Test Result**
mAP50-95 (box)	88.8 %	90.2 %
mAP50-95 (mask)	86.5 %	88.2 %
Recall (mask-critical objects)	91.6 %	87.1 %

Table 7: Validation and Test Results after Hyperparameter Tuning

with validation result of 91.6 % and test result of 87.1 %. The enhancements in mAP and recall demonstrate that the model is now better tuned for recognizing and segmenting objects in real-time, leading to more reliable obstacle avoidance performance in lawn mower.

4.3 Performance of the Optimized Model

The optimization process focused on converting the model from its original PyTorch framework to NCNN, a lightweight neural network inference framework. The optimized NCNN model's performance was evaluated on validation and test datasets. Using evaluation metrics, the optimized model was evaluated to assess the impact of optimization on detection accuracy and segmentation quality.

Validation Results with NCNN model

Evaluation Metrics	**Value**
mAP50-95 (box)	88.5 %
mAP50-95 (mask)	85.5 %
Recall (mask-critical objects)	91.1 %

Table 8: Validation Results with NCNN model

In the PyTorch model, the mAP50-95 for box predictions was 88.8 %, and for mask predictions, it was 86.5 % as shown in Table 7. After conversion to NCNN, the validation mAP50-95 values slightly decreased to 88.5 % for box predictions and 85.5 % for mask predictions (Refer Table 8).

Test Results of NCNN model

Evaluation Metrics	**PyTorch Model**	**NCNN Model**
FPS	0.67 FPS	2.3 FPS
Inference Time	1492 ms	435 ms
mAP50-95 (box)	90.2 %	89.4 %
mAP50-95 (mask)	88.2 %	87.1 %
Recall (mask-critical objects)	87.1 %	87.9 %

Table 9: Comparison of Test Results of PyTorch and NCNN model

In terms of the mAP50-95 $_{(box)}$ and mAP50-95 $_{(mask)}$ metrics, both PyTorch and NCNN models showed strong performance, with only a slight decrease in accuracy after the conversion to NCNN as reflected in the Table 9. For mAP50-95 $_{(box)}$, the PyTorch model achieved 90.2 %, while the NCNN model showed a marginal drop to 89.4 %. Similarly,

for mAP50-95 $_{(\text{mask})}$, the PyTorch model achieved 88.2 %, and the NCNN model slightly decreased to 87.1 %. This minimal reduction reflects that the NCNN model maintains almost the same level of performance as the original PyTorch model but still it losses some necessary information. As for Recall $_{(\text{mask-critical objects})}$, the NCNN model actually showed an improvement, increasing from 87.1 % to 87.9 %. It suggests that the NCNN version was slightly better at correctly identifying and recalling critical objects. The reason for this increment might be that the NCNN model was generalizing the instances of "Critical Objects" class bettter than PyTorch Model on the unseen real-world data.

The final model was tested in real time on a Raspberry Pi 5 by deploying it with its original PyTorch framework after hyperparameter tuning. Subsequently, the model was converted to NCNN format and deployed again on Raspberry Pi 5 to assess the improvements. The primary objective of this phase was to measure and compare the evaluation metrics of Frames Per Second (FPS) and inference time.
The NCNN model achieved 2.3 FPS whereas PyTorch model achieved 0.67 FPS. NCNN model reduced the inference time from 1492 ms to 435 ms, approximately a 4 times better improvement. This comparison highlights that NCNN model delivers better inference on Raspberry Pi 5 while maintaining adequate accuracy for both detection and segmentation tasks. The improvement in FPS and reduction in inference time confirm the optimization's success and makes the model suitable for deployment in resource-constrained environments.

4.4 Real-Time Performance on Raspberry Pi 5

The following subsection explores the real-time performance of developed model when deployed on Raspberry Pi 5. This includes an overview of the deployment setup and then observations are provided regarding the model's responsiveness, processing speed, and detection accuracy in real-time conditions on the lawn mower.

4.4.1 Deployment of the model on Raspberry Pi 5

In order to assess real-time performance of NCNN model on Raspberry Pi 5, experiments were carried out at AL-KO's garden utilizing the ROBOLINHO robotic lawn mower. The python script for this task is mentioned in Appendix A. The script also stores the prediction results of last frame in JSON file. This file can be later used for path-planning of lawn mower. The goal of this evaluation was to assess the lawn mower's capability to accurately detect and segment obstacles in a dynamic outdoor environment. The experiments focused on measuring both the segmentation accuracy and the model's inference time. The purpose of this testing phase was to ascertain whether the optimized model could withstand the challenges of operating in real-time while accurately detecting Critical Objects like humans, pets, and non-lawn elements, ensuring the robotic mower would avoid these obstacles and

operate safely within the predefined lawn area.

4.4.2 Performance Observations

The following list outlines some key observations during real-time performance test on Raspberry Pi 5:

- **Observation 1: Accurate Lawn Area Segmentation in Complex Environments**

Figure 35: Accurately Segmented Lawn Area

In this scenario as shown in the Figure 35, the model was tested in a complex environment with trees and uneven terrain. The segmentation model performed exceptionally well by accurately distinguishing the lawn area from the tree in the garden, identifying the tree as part of the background and correctly masking the lawn area. This is crucial for ensuring that the lawn mower only focuses on mowing the designated lawn regions without interference from other non-relevant objects, such as trees or other items. The clear segmentation in this complex setting demonstrates the model's strong ability to handle varied terrain and obstacles efficiently, enhancing its applicability in outdoor, unstructured environments.

- **Observation 2: Successful Detection of Human as Critical Object**

When a human stepped into the frame during testing, the model correctly identified the person as a critical object as expected. Not only was the human correctly

Figure 36: Correctly Classified Human as Critical Object

identified, but the model also applied an accurate mask around the person as seen in Figure 36.It ensures that the robotic lawn mower would halt operation or avoid the individual in operational environments where humans or pets may enter the lawn mower's working area to prioritize safety.

- **Observation 3: False Positive Case with Leaves on the Ground**

  In another test case, where there were leaves scattered on the ground (see Figure 37), the model showed some limitations in its segmentation. Some of the leaves were mistakenly identified as Critical Objects, resulting in false positives. Although this false detection could be problematic, the identification of small objects like leaves as obstacles is less critical for safety since the mower can avoid them. However, this indicates that further tuning or training would be required to reduce such false positives, especially in environments where small objects frequently appear on the ground.

- **Observation 4: Missed Detection and Segmentation of People**

  In a few frames, the model encountered a situation with two people in the frame but successfully segmented only one of them, leaving the other unsegmented. However, the legs of another person were still marked as background and not as part of the lawn. Although the system did not label the them as "Critical Objects", the mower's safety mechanism remained intact, as it recognized the region as non-lawn, meaning the mower would not operate in that area. This adds an additional layer of safety

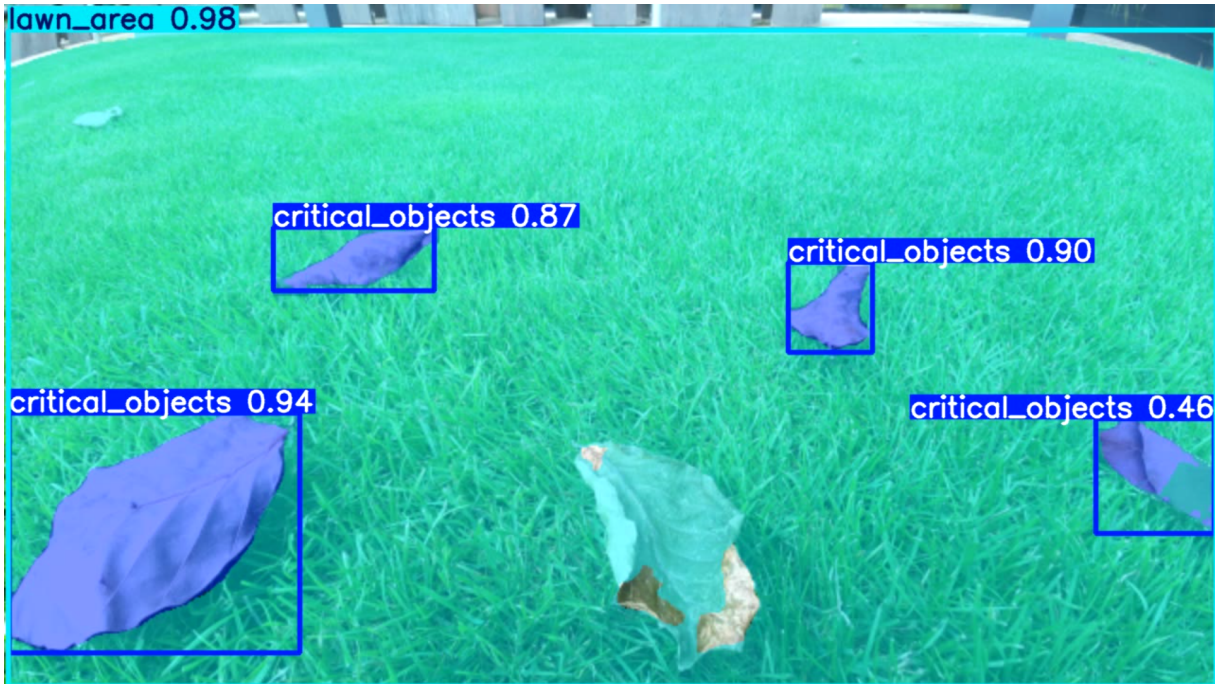

Figure 37: Leaves false classified as a Critical Objects

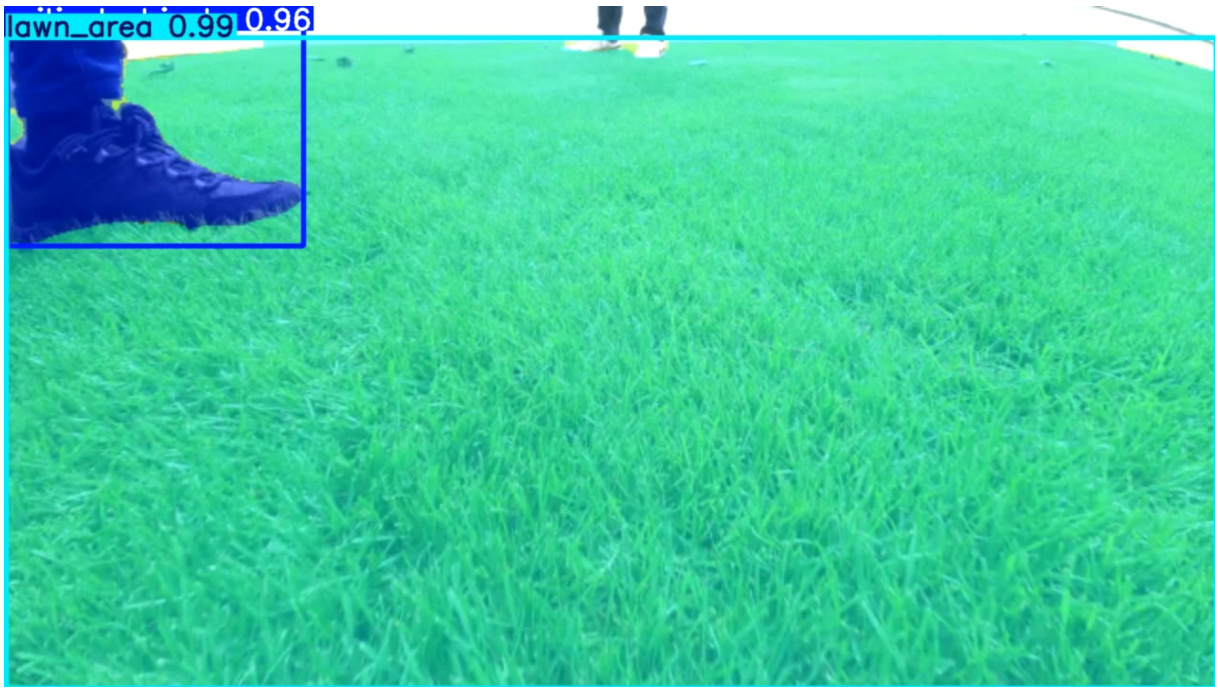

Figure 38: Missed Segmentation of a People

even in cases of classification errors, minimizing the risk of mowing in unsafe zones.

These outcomes offer insightful information about how well the model functions on the Raspberry Pi 5. The model demonstrated solid accuracy in correctly identifying and segmenting objects relevant to the task, especially in complex and dynamic environments, while also exposing areas for potential improvement, such as reducing false positives for smaller objects like leaves.

5 Discussion

This section delves into the critical factors affecting the performance of the robotic lawn mower. The requirements that we defined in the section 3.5 are checked whether these were fulfilled or not. As summarized in Table 10, while the model achieved satisfactory results in terms of mAP and recall for box and mask predictions, it fell short of meeting the safety benchmarks for FPS and inference time. The implications of these findings will be examined for improving the model's effectiveness in Future.

Requirements	**Expected Results**	**Achieved Results**	**Fulfilled**
Safe FPS	$\geq$ 7.5 FPS	$\sim$ 2.3 FPS	No
Inference time	$\leq$ 133 ms	$\sim$ 435 ms	No
mAP50-95 box	$>$ 44.6%	89.4%	Yes
mAP50-95 mask	$>$ 36.8%	87.1%	Yes
Recall mask (Critical Objects)	$>$ 96%	90.2%	No

Table 10: Comparison of Expected and Achieved Results

5.1 Impact of Dataset

For optimal training, it is generally recommended to have at least 1,500 images per class and a minimum of 10,000 labeled instances per class [61]. From these wide range of images, the model learns a various features and variations associated with each class. Thus it provides effective generalization in real-world scenarios.

In this study, the dataset had only 2,176 images including all classes, which was not sufficient as per the recommendation. The major reason for this limitation of dataset was limited time available for the thesis. It took much more time to collect images from different gardens and also to annotate each image. Due to this time limitation, the dataset lacked the necessary variation and size. This caused performance issues about false positives in the case of leaves as discussed earlier. Because of this, model trained on a limited number of examples misclassified leaves as critical objects. This highlights the importance of wide range of training images that contains various environmental conditions, lighting variations and object orientations.

Moreover, this limited size of dataset also impacted the model's ability to consistently identify humans in the environment. There were less examples of human instances captured in diverse poses and contexts in "Critical Objects" class. This caused a model to miss a detections of humans in few frames. This presents a safety concern, as the failure to detect human presence can lead to dangerous situation.

There was also issue of class imbalance. In this project, there were total 2,176 images in the dataset, but only 521 images contained instances of the class labeled as "Critical Objects." While there were 2154 images that contained instances of the class labeled as

"Lawn Area". This imbalance can lead to the model being biased towards the majority class, resulting in poor performance when detecting critical objects. To address this issue, data augmentation techniques or oversampling of the minority class may be necessary. A more balanced class representation would ideally improve the model's capacity to pick out the distinctive features of important objects.

Background images are essential in refining model performance by minimizing false positives (FP). In the current dataset, 19 images were identified as null images, which do not contain any instance of classes. To effectively reduce FP, it is recommended to include about 0-10% background images in the dataset [61]. For reference, the COCO dataset includes 1,000 background images, accounting for approximately 1% of its total [61]. These background images serve as a counterbalance to images with objects, helping the model differentiate between areas with and without relevant objects. So, increased number of background images can help model to generalize better.

To enhance the model's performance and mitigate both false positives and missed detections, future work should prioritize expanding the dataset. This could involve allocating additional time for collecting images from a variety of garden settings and ensuring thorough annotation. Additionally, by using data augmentation techniques, the effective size of the dataset can be increased, giving the model the diversity it need to perform consistently in changing situations.

5.2 Safety Consideration

As presented in the Table 10, the system did not meet the required standards for safe Frames Per Second (FPS) and inference time. It achieved only 2.3 FPS compared to the minimum requirement of 7.5 FPS. This deficit raises concerns regarding the safety and responsiveness of the robotic lawn mower in dynamic environments that may contain various obstacles.

Achieving a 7.5 FPS is critical for lawn mower to detect obstacle, make decision and respond accordingly. The mower must process images quickly enough to take appropriate actions and avoid collisions. There may be safety risk because of system's existing performance. With current 2.3 FPS, the mower may not detect and avoid obstacle quickly. Thus it may collide with obstacle and put risk to it's surrounding.

To address this limitation, incorporating more advanced hardware accelerators is essential. For example, the Coral Edge TPU might be suitable option to improve performance on Raspberry Pi 5. it can greatly increase inference performance while consuming less power. Additionally, edge devices equipped with powerful GPUs offer substantial computational resources that can greatly improve the performance of the model (e.g., NVIDIA Jetson Orin). The integration of these additional hardware can also be useful when model gets larger in size in Future.

By upgrading to these advanced hardware solutions, it can be ensured that robotic lawn mower operates safely and effectively in real-world scenarios. Future studies should be included to investigate how these hardware accelerators may be used to meet required performance metrics and increase the safety of lawn mower's surrounding.

5.3 Limitations

A critical limitation of the camera based robotic lawn mower with this developed YOLO model is the potential blockage of the camera lens due to environmental factors such as dirt, dust, or water drops. When the lens is covered by any reason and becomes obstructed, the camera would not be able to see the complete field of view. Thus it may miss the seeing the instances of any classes and it may not recognize and segment it properly. This could result in dangerous situations, because the mower may fail to recognize humans, pets, or other critical objects in its path, thereby increasing risk to it's surrounding.
To address this issue, one effective solution is to integrate an additional object detection model specifically designed to identify blockage caused by water or dirt on camera lens. This model would analyze the visual input from camera and detect any obstructions that could impair field of view for camera. By employing this kind of model, it could determine whether the lens is blocked or not.
Once detected, the system could trigger appropriate responses, such as alerting user to clean the lens or temporarily pausing mower's operation until visibility is restored. Additionally, the integration of this model could enhance overall lawn mower's reliability, because robotic lawn mower would operate safely and effectively even in challenging weather conditions. Thus the mower can maintain optimal performance and reduce risk of accidents caused by undetected obstacles.

6 Conclusion and Future Outlook

In recent years, advancement in deep learning has opened up new opportunities across various fields like autonomous systems. An example of where this progress can be applied is in automating lawn mowing activity.

This thesis provided the insights of fundamental concepts in the area of AI, ML and DL. Exploration of an advancements in Computer Vision provided the importance of the image segmentation in autonomous systems. In this thesis, a comparative study of different image segmentation models has been conducted to compare their accuracy and computational efficiency. This quantitative analysis of segmentation model highlighted that YOLO model could be the suitable algorithm to enable real-time obstacle avoidance in robotic lawn mowers. A thorough review of architecture of YOLO model was then conducted. This thesis identifies a primary research gap in assessing YOLO's segmentation performance on Raspberry Pi. This includes the lack of comprehensive evaluations on accuracy and frame rate for real-time processing on such hardware. To address this, the research focused on examining how YOLO can be implemented on a Raspberry Pi 5 for real-time obstacle avoidance in a dynamic garden environment.

To achieve this objective, a methodological approach was designed in specific stages. To train the YOLO model, a custom dataset consisting of 2,176 images from AL-KO company's garden was prepared. The dataset was labeled using Roboflow tool to enable the model to train and test on them.

Two different training experiments were conducted with different image sizes (640 x 384 pixels, 1280 x 736 pixels) during training of the model. Training results of the experiments on test dataset highlighted that the model trained with 640 x 384 pixels achieved 89.8 % mAP50-95 $_{(box)}$, 87.9 % mAP50-95 $_{(mask)}$ and 89.3 % Recall $_{(mask\text{-}critical\ objects)}$. This performance of the model was higher compared to the model trained with image size of 1280 x 736 pixels. Thus after training the model, hyperparameter tuning was performed with this image size to further improve its accuracy. Tuning results indicated that performance of the model was increased. It achieved 90.2 % mAP50-95 $_{(box)}$, 88.2 % mAP50-95 $_{(mask)}$ and 87.1 % Recall $_{(mask\text{-}critical\ objects)}$ on test dataset. Though increased mAP metrics, Recall on test dataset was decreased after tuning. This indicates that the tuned model might not generalize well for the instances of "Critical Objects" class on unseen data.

The model was subsequently optimized for deployment on Raspberry Pi 5 by exporting it to NCNN format. Both the tuned model with PyTorch format and the model with NCNN format were deployed on the Raspberry Pi 5 for the evaluation of FPS and inference time. The performance of another evaluation metrics were changed a bit but there was not significant change in it. The model with NCNN model achieved 89.4 % mAP50-95 $_{(box)}$, 87.1 % mAP50-95 $_{(mask)}$ and 87.9 % Recall $_{(mask\text{-}critical\ objects)}$ on test dataset. However,

this optimization improved both frames per second (FPS) and inference time. The tuned PyTorch achieved 0.67 FPS and inference time of 1492 ms. Whereas model with NCNN format was able to process 2.3 FPS with inference time of 435 ms.

The NCNN model was implemented on lawn mower with Raspberry Pi 5 to evaluate the performance in real-world. The test was carried out at AL-KO's garden. Key observations showed that the model could correctly identify and segment lawn area and critical objects in outdoor settings. It accurately segmented the lawn area from the complex background in the garden. It was also able to properly segment person as a critical object with confidence of 96 %. In some test observation, it missed to detect some of the presented humans as critical objects. Also, in some frames the leaves were identified as critical objects by mistake and not as a lawn area. Lack of such images in training dataset might be the reason for these false positives.
When compared to the expected results of evaluation metrics, it did not fulfilled all the expected results. It did not fulfilled the expected results of Safe FPS, Inference Time and Recall $_{\text{(mask-critical objects)}}$. The major reason of not fulfillment of these metrics was due to the need for large datasets. The current dataset only consisted 2176 images including all classes, but recommended image dataset should have at least 1500 images per class. In addition, the current dataset contained only 19 null images. But the dataset should contain more null images to help model differentiate between areas with and without relevant objects. Another challenging situation was regarding lack of FPS achieved. The model with NCNN format achieved only 2.3 FPS compared to expected 7.5 FPS for safe operation of the lawn mower. To address this situation, additional hardware accelerators (e.g., Coral Edge TPU) might be used to increase the FPS. Also, the edge devices with GPUs (e.g., NVIDIA Jetson Orin) could improve the computation and provide faster inference time. One limitation of the developed model in this research is that it focused on just three classes (lawn area, critical objects, background). So in case if the camera lens is blocked by some environmental factors like dust, it would not be detect and segment anything. Thus it would put the risk to its surrounding.

6.1 Future Work

Although the model showed satisfactory real-time performance, future work should focus on increasing the dataset size. It is crucial to enhance the detection of smaller objects and also to reduce false positive cases. Increasing the variety of training photos and incorporating data augmentation techniques might improve the robustness of the model. Additionally, exploring advanced edge devices (e.g., NVIDIA Jetson Orin) or hardware accelerators (e.g., Coral Edge TPU) and implementation of them could further improve FPS and inference time. Furthermore, future work should focus on leveraging the prediction results

generated by the model for path planning of the robotic lawn mower. Currently, the model's segmentation predictions are saved in JSON format, providing detailed information about the locations of detected objects such as lawns, trees, and critical objects. The next step should be to integrate these prediction outputs into a path planning and decision making system that will allow the lawn mower to autonomously navigate around obstacles while efficiently covering the lawn area. Future advancements in these areas may open the door to the creation of safer, more efficient autonomous lawn mowers that can function well in unstructured outdoor environments.

References

[1] K. Baluprithviraj, R. Harini, M. Janarthanan, and C. Jasodhasree, "Design and development of smart lawn mower," Oct. 2021, pp. 1215–1219. DOI: 10.1109/ICOSEC51865.2021.9591825.

[2] C. Kang, P. K. Ng, and L. Kia Wai (K. W. Liew), "The conceptual synthesis and development of a multifunctional lawnmower," *Inventions*, vol. 6, p. 38, May 2021. DOI: 10.3390/inventions6020038.

[3] I. Khansa, G. D. Pearson, K. Bjorklund, A. Fogolin, M. O'Brien, and R. E. Kirschner, "Pediatric lawnmower injuries: A 25-year review," *JPRAS Open*, 2021. DOI: 10.1016/j.jpra.2021.05.001.

[4] J. Siaulys and A. Paulauskaite-Taraseviciene, "Comparative evaluation of deep learning architectures for environment and obstacle recognition in robotic lawn care," *Aachen : CEUR-WS*, vol. 3575, pp. 55–63, 2023.

[5] H. Brennan. "Innovating for hedgehog safety: The rise of facial recognition in robot lawn mowers." Accessed: 2024-10-25. (2024), [Online]. Available: https://seerbi.uk/author/hannahb.

[6] Allied Market Research. "Robotic lawn mower market size, share, competitive landscape and trend analysis report, by range, by end user, by distribution channel : Global opportunity analysis and industry forecast, 2023-2032." Accessed: 2024-10-06. (2023), [Online]. Available: https://www.alliedmarketresearch.com/robotic-lawn-mower-market.

[7] AL-KO Gardentech Webpage. Accessed: 2024-10-06. (2024), [Online]. Available: https://al-ko.com/bereiche/gardentech.

[8] AL-KO Gardentech. "Robotic lawn mowers - lawn care tools." Accessed: 2024-10-06. (2024), [Online]. Available: https://alko-garden.de/gartengeraete/rasenpflege/maehroboter/.

[9] J. Zhou, F. Chen, A. Berry, M. Reed, S. Zhang, and S. Savage, "A survey on ethical principles of ai and implementations," *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 3010–3017, 2020. DOI: 10.1109/SSCI47803.2020.9308437.

[10] S. Suhaib Kamran, A. Haleem, S. Bahl, M. Javaid, C. Prakash, and D. Budhhi, "Artificial intelligence and advanced materials in automotive industry: Potential applications and perspectives," *Materials Today: Proceedings*, vol. 62, pp. 4207–4214, 2022. DOI: https://doi.org/10.1016/j.matpr.2022.04.727.

[11] A. Luckow, K. Kennedy, M. Ziolkowski, *et al.*, "Artificial intelligence and deep learning applications for automotive manufacturing," Dec. 2018, pp. 3144–3152. DOI: 10.1109/BigData.2018.8622357.

[12] M. Helm, A. Swiergosz, H. Haeberle, *et al.*, "Machine learning and artificial intelligence: Definitions, applications, and future directions," *Current Reviews in Musculoskeletal Medicine*, vol. 13, 2020. DOI: 10.1007/s12178-020-09600-8.

[13] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," Apr. 2023. DOI: 10.1016/j.cogr.2023.04.001.

[14] B. Mahesh, "Machine learning algorithms -a review," *International Journal of Science and Research (IJSR)*, vol. 9, Jan. 2019. DOI: 10.21275/ART20203995.

[15] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016, Recent Developments on Deep Big Vision, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.09.116. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231215017634.

[16] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning – erratum," *APSIPA Transactions on Signal and Information Processing*, vol. 3, Dec. 2014. DOI: 10.1017/ATSIP.2014.4.

[17] A. Pointinger, "Masterarbeit: Semantische segmentierung von rasen-bilddaten durch deep convolutional neural networks," Retrieved from https://www.dropbox.com/scl/fi/dzz0o25fmke18qywvaafp/Masterarbeit_Armin_Pointinger_compressed.pdf?rlkey=6tr0dcogtd0n62r99yhf3g7kg&e=2&st=h83d8htv&dl=0, M.S. thesis, Fachhochschule Oberösterreich, Jun. 2019.

[18] E. Grossi and M. Buscema, "Introduction to artificial neural networks," *European journal of gastroenterology hepatology*, vol. 19, pp. 1046–54, Jan. 2008. DOI: 10.1097/MEG.0b013e3282f198a0.

[19] M. Manoharan and M. Louzazni, "Analysis of artificial neural network: Architecture, types, and forecasting applications," *Journal of Electrical and Computer Engineering*, vol. 2022, pp. 1–23, Apr. 2022. DOI: 10.1155/2022/5416722.

[20] A. Vasileiadis, E. Alexandrou, L. Paschalidou, M. Chrysanthou, and M. Hadjichristoforou, *Artificial neural network and its applications*, Apr. 2024.

[21] I. Basheer and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application," *Journal of Microbiological Methods*, vol. 43, no. 1, pp. 3–31, 2000, Neural Computting in Micrbiology, ISSN: 0167-7012. DOI: https://doi.org/10.1016/S0167-7012(00)00201-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167701200002013.

[22] K. Çağrı and U. Ayşegül, "A brief survey and an application of semantic image segmentation for autonomous driving," *Smart Innovation, Systems and Technologies*, p. 1, 2018. DOI: 10.1007/978-3-030-11479-4_9.

[23] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, "A survey on modern trainable activation functions," *Neural Networks*, vol. 138, pp. 14–32, 2021, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.01.026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608021000344.

[24] M. Sazli, "A brief review of feed-forward neural networks," *Communications Faculty Of Science University of Ankara*, vol. 50, pp. 11–17, Jan. 2006. DOI: 10.1501/commua1-2_0000000026.

[25] V. Srilakshmi, G. Uday Kiran, M. Mounika, *et al.*, "Evolving convolutional neural networks with meta-heuristics for transfer learning in computer vision," *Procedia Computer Science*, vol. 230, pp. 658–668, 2023, 3rd International Conference on Evolutionary Computing and Mobile Sustainable Networks (ICECMSN 2023), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2023.12.121. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050923021269.

[26] S. Raquel and O. Salazar, "An artificial neural network model to analyze maize price behavior in mexico," *Applied Mathematics*, vol. 09, pp. 473–487, Jan. 2018. DOI: 10.4236/am.2018.95034.

[27] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A critical review of recurrent neural networks for sequence learning*, 2015. arXiv: 1506.00019 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1506.00019.

[28] D.-Y. Kang, H. Duong, and J.-C. Park, "Application of deep learning in dentistry and implantology," *The Korean Academy of Oral and Maxillofacial Implantology*, vol. 24, pp. 148–181, Sep. 2020. DOI: 10.32542/implantology.202015.

[29] D. Ma, "Recent advances in deep learning based computer vision," in *2022 International Conference on Computers, Information Processing and Advanced Education (CIPAE)*, 2022, pp. 174–179. DOI: 10.1109/CIPAE55637.2022.00044.

[30] W. Bakasa and S. Viriri, "Pancreatic cancer survival prediction: A survey of the state-of-the-art," *Computational and Mathematical Methods in Medicine*, vol. 2021, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:238473913.

[31] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, Jul. 2019. DOI: 10.1186/s40537-019-0197-0.

[32] P. Purwono, A. Ma'arif, W. Rahmaniar, *et al.*, "Understanding of convolutional neural network (cnn): A review," vol. 2, pp. 739–748, Jan. 2023. DOI: 10.31763/ijrcs.v2i4.888.

[33] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *ArXiv e-prints*, Nov. 2015.

[34] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, Mar. 2024. DOI: 10.1007/s10462-024-10721-6.

[35] R. K. Sinha, R. Pandey, and R. Pattnaik, *Deep learning for computer vision tasks: A review*, 2018. arXiv: 1804.03928 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1804.03928.

[36] M. Siam, M. Badawy, M. Abdelrazek, S. Yogamani, M. Jagersand, and H. Zhang, "A comparative study of real-time semantic segmentation for autonomous driving," Jun. 2018. DOI: 10.1109/CVPRW.2018.00101.

[37] Z. Zhou, M. Zhang, J. Chen, and X. Wu, "Detection and classification of multi-magnetic targets using mask-rcnn," *IEEE Access*, vol. 8, pp. 187 202–187 207, Jan. 2020. DOI: 10.1109/ACCESS.2020.3030676.

[38] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, and M. Jagersand, "Rtseg: Real-time semantic segmentation comparative study," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 1603–1607. DOI: 10.1109/ICIP.2018.8451495.

[39] L. Cao, X. Zheng, and L. Fang, "The semantic segmentation of standing tree images based on the yolo v7 deep learning algorithm," *Electronics*, vol. 12, p. 929, Feb. 2023. DOI: 10.3390/electronics12040929.

[40] D. Varnyú and L. Szirmay-Kalos, "A comparative study of deep neural networks for real-time semantic segmentation during the transurethral resection of bladder tumors," *Diagnostics*, vol. 12, p. 2849, Nov. 2022. DOI: 10.3390/diagnostics12112849.

[41] J. Ruiz-Santaquiteria, G. Bueno, O. Deniz, N. Vállez, and G. Cristobal, "Semantic versus instance segmentation in microscopic algae detection," *Engineering Applications of Artificial Intelligence*, vol. 87, p. 103 271, Jan. 2020. DOI: 10.1016/j.engappai.2019.103271.

[42] E. Prasetyo, N. Suciati, and C. Fatichah, "A comparison of yolo and mask r-cnn for segmenting head and tail of fish," in *2020 4th International Conference on Informatics and Computational Sciences (ICICoS)*, 2020, pp. 1–6. DOI: 10.1109/ICICoS51170.2020.9299024.

[43] A. Morera, A. Sanchez, A. Moreno, A. Sappa, and J. Vélez, "Ssd vs. yolo for detection of outdoor urban advertising panels under multiple variabilities," *Sensors*, vol. 20, p. 4587, Aug. 2020. DOI: 10.3390/s20164587.

[44] R. Sapkota, D. Ahmed, and M. Karkee, *Comparing yolov8 and mask rcnn for object segmentation in complex orchard environments*, Dec. 2023. DOI: 10.1016/j.aiia.2024.07.001.

[45] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1506.02640.

[46] J. Terven, D.-M. Cordova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, pp. 1680–1716, Nov. 2023. DOI: 10.3390/make5040083.

[47] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, Jun. 2010. DOI: 10.1007/s11263-009-0275-4.

[48] G. Jocher, A. Chaurasia, and J. Qiu. "YOLO by Ultralytics." Accessed: 2024-10-06. (2024), [Online]. Available: https://github.com/ultralytics/ultralytics.

[49] A. Agarwal, A. Thombre, K. Kedia, and I. Ghosh, "Itd: Indian traffic dataset for intelligent transportation systems," in *2024 16th International Conference on COMmunication Systems NETworkS (COMSNETS)*, 2024, pp. 842–850. DOI: 10.1109/COMSNETS59351.2024.10427394.

[50] Camera Module 3 Wide Description PDF. "Raspberry pi camera module 3." Accessed: 2024-10-06. (2024), [Online]. Available: https://datasheets.raspberrypi.com/camera/camera-module-3-product-brief.pdf.

[51] Roboflow Webpage. "Roboflow: Computer vision tools for developers and enterprises." Accessed: 2024-10-06. (2024), [Online]. Available: https://roboflow.com/.

[52] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, p. 022 022, Feb. 2019. DOI: 10.1088/1742-6596/1168/2/022022.

[53] F. Zhuang, Z. Qi, K. Duan, *et al.*, *A comprehensive survey on transfer learning*, 2020. arXiv: 1911.02685 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1911.02685.

[54] Tencent NCNN Webpage. "Tencent - ncnn." Accessed: 2024-10-06. (2024), [Online]. Available: https://github.com/Tencent/ncnn.

[55] Ultralytics NCNN Webpage. "Ultralytics - ncnn format." Accessed: 2024-10-06. (2024), [Online]. Available: https://docs.ultralytics.com/integrations/ncnn/#how-do-i-export-ultralytics-yolo11-models-to-ncnn-format.

[56] S. Hicks, I. Strumke, V. Thambawita, *et al.*, "On evaluation metrics for medical applications of artificial intelligence," *Scientific Reports*, vol. 12, Apr. 2022. DOI: 10.1038/s41598-022-09954-8.

[57] L. D. Quach, K. Nguyen, A. Nguyen Quynh, and H. Tran Ngoc, "Evaluating the effectiveness of yolo models in different sized object detection and feature-based classification of small objects," *Journal of Advances in Information Technology*, vol. 14, pp. 907–917, Sep. 2023. DOI: 10.12720/jait.14.5.907-917.

[58] J. Zhao, A. Lipani, and C. Schillaci, "Fallen apple detection as an auxiliary task: Boosting robotic apple detection performance through multi-task learning," *Smart Agricultural Technology*, vol. 8, p. 100 436, 2024, ISSN: 2772-3755. DOI: https://doi.org/10.1016/j.atech.2024.100436. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772375524000418.

[59] M. Sohan, T. Ram, and V. Ch, "A review on yolov8 and its advancements," in Jan. 2024, pp. 529–545, ISBN: 978-981-99-7999-8. DOI: 10.1007/978-981-99-7962-2_39.

[60] COCO Webpage. "Coco: Common objects in context." Accessed: 2024-10-06. (2024), [Online]. Available: https://cocodataset.org/#home.

[61] Ultralytics Webpage. "Tips for best training results." Accessed: 2024-10-06. (2024), [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/.

A Appendix: Implementation Code on Raspberry Pi 5

```python
import cv2
from picamera2 import Picamera2
import time
from ultralytics import YOLO
import json
import supervision as sv
from collections import deque

# Initialize the Picamera2
picam2 = Picamera2()
picam2.preview_configuration.main.size = (4608,2592)
picam2.preview_configuration.main.format = "RGB888"
picam2.preview_configuration.align()
picam2.configure("preview")
picam2.start()

# Load the YOLOv8 models
ncnn_model = YOLO("v8s_640_tune_ncnn_model", task="segment")

frame_count = 0
prev_time = time.time()
all_predictions = deque(maxlen=1)
try:
    while True:
        # Capture frame-by-frame
        frame = picam2.capture_array()

        current_time = time.time()
        elapsed_time = current_time - prev_time
        prev_time = current_time

        # Calculate FPS (1 / time taken for one frame)
        fps = 1 / elapsed_time if elapsed_time > 0 else 0

        # Run YOLOv8 inference on the frame
        resized_frame = cv2.resize(frame, (640,384))
        result = ncnn_model(resized_frame)[0]
        detections = sv.Detections.from_ultralytics(result)
        # Collect predictions for the current frame
        predictions = []
        if detections.xyxy is not None and detections.confidence is not None and detections.class_id is not None and detections.mask is not None:
            for box, score, cls, mask in zip(detections.xyxy, detections.confidence, detections.class_id, detections.mask):
                prediction = {
                    "bounding_box": box.tolist(),
                    "score": float(score),
                    "class": int(cls),
                    "mask": mask.tolist() if mask is not None else None
                }
                predictions.append(prediction)

        # Visualize the results on the frame
        annotated_frame = result.plot()

        cv2.putText(annotated_frame, f"FPS: {fps:.2f}", (20, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        # Display the resulting frame
        cv2.imshow("Camera", annotated_frame)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) == ord("q"):
            break

except KeyboardInterrupt:
    print("Interrupted by user")

finally:
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    output_file = f"predictions_{timestamp}.json"
    with open(output_file, 'w') as f:
        json.dump(predictions, f, indent=4)
    print(f"Last frame predictions saved to {output_file}")

    # Release resources and close windows
    cv2.destroyAllWindows()
```

Figure 39: Model implementation and saving prediction results on Raspberry Pi 5 with FPS monitoring

B Appendix: Models

The following GitHub repository access is restricted. Please check permissions beforehand.

- GitHub Link: `https://github.com/kishan2910/Master_Thesis_AL_KO.git`
- In "v8s_640/train5" folder, the model and relevant training results after Training Experiment 1 are provided.
- In "v8s_1280/train2" folder, the model and relevant training results after Training Experiment 2 are provided.
- In "v8s_640_tune/train4" folder, the model and relevant tuning results after Hyperparameter Tuning are provided.
- In "v8s_640_tune_ncnn_model" folder, the exported NCNN model is provided.
- In addition, the video "test.avi" provides the video of test results of NCNN model at AL-KO's garden.