Cogent Engineering

ISSN: 2331-1916 (Online) Journal homepage: www.tandfonline.com/journals/oaen20

Investigation of MobileNet-Ssd on human follower robot for stand-alone object detection and tracking using Raspberry Pi

Vidya Kamath & Renuka A

To cite this article: Vidya Kamath & Renuka A (2024) Investigation of MobileNet-Ssd on human follower robot for stand-alone object detection and tracking using Raspberry Pi, Cogent Engineering, 11:1, 2333208, DOI: 10.1080/23311916.2024.2333208

To link to this article: https://doi.org/10.1080/23311916.2024.2333208

COGENT ENGINEERING
2024, VOL. 11, NO. 1, 2333208
https://doi.org/10.1080/23311916.2024.2333208

COMPUTER SCIENCE | RESEARCH ARTICLE

OPEN ACCESS Check for updates

Investigation of MobileNet-Ssd on human follower robot for stand-alone object detection and tracking using Raspberry Pi

Vidya Kamath and Renuka A

Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka, India

ABSTRACT

Human following is a very useful task in the robotics industry. With modern compact-sized robots, there is a demand for further investigated computer-vision solutions that can perform effectively on them. A well-known deep learning model along this line of thought is the MobileNet-Ssd, an object detection model renowned for its resource-constrained usage. Available in popular frameworks like TensorFlow and PyTorch, this model can be of great use in deployments on robotic applications. This research attempts to investigate the MobileNet-Ssd model in order to evaluate its suitability for stand-alone object detection on a Raspberry Pi. To determine the effect of input size on the model, the model's performance has been investigated with speed in frames-per-second across different input sizes on both CPU and GPU-powered devices. To evaluate the model's effectiveness in the human following task, a Raspberry Pi-based robot was designed leveraging the tracking-by-detection approach with TensorFlow-Lite. Furthermore, the model's performance was evaluated using PyTorch while the model's inputs were adjusted, and the results were compared to those of other state-of-the-art models. The investigation revealed that, despite its modest speeds, the model outperforms other noteworthy models in PyTorch and is an ideal choice when working with Raspberry Pi using TensorFlow-Lite.

ARTICLE HISTORY
Received 13 March 2023
Revised 15 November 2023
Accepted 15 March 2024

KEYWORDS
Raspberry Pi; object detection; human follower robot; robot vision; deep learning; real-time

REVIEWING EDITOR
D. T. Pham, University of Birmingham, United Kingdom

SUBJECTS
Artificial Intelligence; Computer Engineering; Computer Science (General)

1. Introduction

In the field of robotics technology, tremendous progress has been made over the last few years. The success of deep learning models in the field of computer vision has been one of the main driving factors for this vast area of robot applications. Vision-based robots have become a major focus of the Internet of Things (IoT) industry's advancement in creating smart homes and smart cities (Singh et al., 2022). Making these applications small and affordable is a significant achievement that can increase their applicability, availability and affordability to a wide range of populations. Various models are being developed by researchers to perform the task of object detection. Recently there has been a drift towards developing models that are resource-friendly and can be used with embedded controllers and boards (Kamath & Renuka, 2023), making them extremely useful in the construction of robots. With the growing number of proposals of models that are based on deep learning, the scope of vision-based robots for tasks involving object detection and tracking has predominantly increased.

A robot's ability to track and follow a moving object can be used for a multitude of activities. Whether it is to aid humans and to make life easier for them, or for extremely important tasks like defense, surgeries, or handling hazards; the ability to rely on robots has become a prerequisite for their acceptance. The widespread usage of service robots in industrial settings, such as manufacturing, warehousing and healthcare, is a huge example (Islam et al., 2019; Malik et al., 2021). Increasingly, there has been an upsurge in the use of companion robots for social, medical and surveillance purposes (Chita-Tegmark & Scheutz, 2021).

1.1. Motivation

Several real-time challenges faced by domestic robots can be resolved by the use of lightweight

CONTACT Renuka A renuka.prabhu@manipal.edu Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India

object-detection models (Jiang et al., 2022). While building new models necessitates dataset preparation, training and numerous other factors related to model training such as sensitivity analysis, uncertainty analysis, model and weight optimization and data validation (Abbaszadeh Shahri et al., 2021, 2022; J. Alzubi et al., 2021; Asheghi et al., 2020; Movassagh et al., 2023), adopting pre-trained models from libraries provides a simple solution to the problem.

MobileNet-Ssd is one of the well-known object detection models, which is available in popular frameworks like TensorFlow (Abadi et al., 2015) and PyTorch (Paszke et al., 2019) and is well-known for its applicability on constrained devices like Raspberry Pi. This model can prove to be highly beneficial in robotic application deployments. Therefore, this work attempts to analyze the popular MobileNet-Ssd model (Howard et al., 2017; Sandler et al., 2018) for its performance on Raspberry Pi. The model is utilized to perform the task of detecting and following a human by deploying it on a human follower robot. For a better understanding of the model's underlying design advantages and deployment motives, an in-depth investigation of its performance in terms of speed in frames per second (FPS) is also carried out.

The primary goal of this investigation is to provide a greater understanding of the deployment aspects of the aforementioned model and to provide insights for people interested in working in the field of deep learning on constrained devices, particularly for building compact robots.

1.2. Organization

The Section 2 that follows contains related works and preliminaries on the MobileNet-Ssd model architecture. Section 3 describes the investigation's objective, as well as the problem description and research question. The overall methodology is described in Section 4. In Section 5, details on the design of the robot and tracking mechanism are provided and Section 6 contains the details on the result analysis. Section 7 includes the discussion and inference drawn from this investigation. Finally, Section 8 outlines the limitations of this work, and Section 9 summarizes the findings along with recommendations for future research.

2. Related work

This section provides a quick overview of the robots performing human-following tasks. It also mentions a few more works that employ the MobileNet-Ssd model for constrained usage, followed by a brief introduction to the model.

2.1. Background

The development of successful deep learning-based object detection models in the decade following 2012 has greatly advanced the field of robot vision (Algabri & Choi, 2021; Kastner et al., 2022; Yuan et al., 2021). One of the most helpful characteristics of service robots is visual servoing in robots that focus on the aim of tracking humans. Image-based (visual) servoing refers to a robot's autonomous navigation using visual feedback, in which the tracking mechanism relies on image-based features to choose the best pathway to follow the object (Gupta et al., 2017). The use of deep learning-based feature extraction to power these robots is a significant step forward, as long as resource needs are met adequately. However, developing new models requires significant resources and many factors must be taken into consideration for both data validation and complete model optimization (O. Alzubi et al., 2022; Chen et al., 2020; Ghaderi et al., 2022).

The MobileNet-Ssd (Howard et al., 2017; Sandler et al., 2018; Sanjay Kumar et al., 2021) gained popularity for its usage in constrained devices like the Raspberry Pi and other equivalent boards, performing a variety of applications like detecting driver inattention, sighting obstacles, or following a human with UAV drones (Han & Peng, 2020; Khare et al., 2020; Ou et al., 2019; Safadinho et al., 2020; Shakeel et al., 2019). On other widely used boards, such as the Jetson Nano, the model is also known to be utilized for pedestrian detection as well as for human following (Murthy et al., 2021; Zhou et al., 2022). The use of these pretrained models for any robot vision task has been made simple by communities that assist model development and training, such as TensorFlow (Abadi et al., 2015), PyTorch (Paszke et al., 2019) and TinyML (Shafique et al., 2021), to mention a few. In addition to providing the models with pretrained weights, these frameworks enable support for deploying the models in a wide range of devices, including several microprocessor boards.

2.2. Preliminaries: MobileNet-Ssd

The MobileNet-Ssd model is the deep-learning-based object detection model, which replaces the backbone of Single Shot Multibox Detector (SSD) (Liu et al., 2016) with the MobileNet architecture (Sandler

et al., 2018). This model is well known for tasks involving low-power devices such as mobile phones, embedded boards and controllers, which are constrained in memory and computation power (Karaman et al., 2021; Safadinho et al., 2020; Sanjay Kumar et al., 2021; Shakeel et al., 2019; Zhang et al., 2020).

The architecture of the SSD model using the MobileNet backbone is shown in Figure 1. The MobileNet architecture acts as the backbone, spanning a total of twelve layers with the last softmax layer replaced with a normal convolution layer and is followed by the AuxiliaryNet (Layers 13 to 16) from the SSD model design, which is utilized to discover features of different shapes. Each Layer in the MobileNet backbone consists of a depthwise convolution followed by a pointwise convolution (Sandler et al., 2018), while the Layers 13 to 16 from the AuxiliaryNet consists of only the traditional convolution layer. The output from layers 6, 12, 13, 14, 15 and 16 with dimensions $38 \times 38 \times 512$, $19 \times 19 \times 1024$, $10 \times 10 \times 512$, $5 \times 5 \times 256$, $3 \times 3 \times 256$, and $1 \times 1 \times 256$, respectively, are forwarded to a PredictionNet, which generates a total of 8732 default bounding boxes per class. These default boxes are matched with the ground truth boxes to calculate the multi-box loss during the training. A detailed explanation of the model's design can be found in Howard et al. (2017), Liu et al. (2016), and Sandler et al. (2018).

2.2.1. Library support

The MobileNet-Ssd model is supported by TensorFlow and is readily available for use with TensorFlow's libraries. It is also part of PyTorch's *torchvision.models* library. When using TensorFlow, there is a requirement to first convert the model to 'model.tflite' format, then install the TensorFlow Lite (TFLite) interpreter on the Raspberry Pi and use it to interpret the model. When using PyTorch, there is a requirement to install and load the torchvision library on the Raspberry Pi for importing and utilizing the model and its weights. In either case, the model can be quantized to improve performance speed at the cost of accuracy.

3. Objective of this study

Due to the advances in the development of tiny devices, previously enormous robots can now be made compact. The development of small processors like the Raspberry Pi has facilitated the creation of modern robots that are tiny in size yet can perform effectively. These devices have provided better and more affordable solutions in the past, and they seem to be promising products that can be utilized in the IoT and robotics industries in the years to come. However, not all object detection models perform well on constrained devices. Even though modern robots are sufficiently small in size, they still require large servers for real-time processing, with seamless connectivity. The ability to build tiny robots that can carry out stand-alone object detection and tracking, without the aid of any external computing support is still very distant from being achieved. With the possibility of this being accomplished, the robotics industry can benefit greatly and can become more cost-effective.

A range of resource-constrained applications have used the deep learning model MobileNet-Ssd, such as driver drowsiness detection (Dipu et al., 2021; Shakeel et al., 2019), crowd monitoring (Kastner et al., 2022) and so on. Additionally, it also became apparent that the model is capable of performing detection remotely on constrained devices such as the Jetson Nano (Murthy et al., 2021; Zhou et al., 2022). In many IoT applications while using even more constrained devices, such as Raspberry Pi, the model is able to perform effectively with the aid of

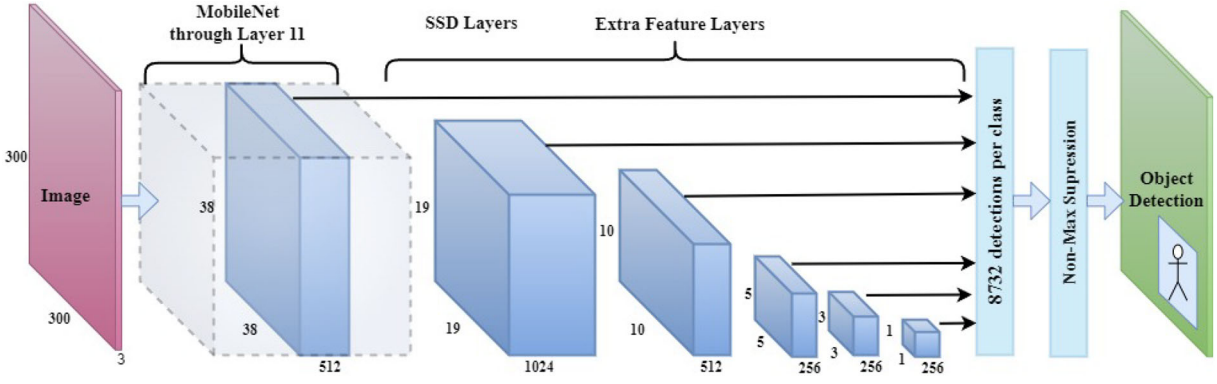


Figure 1. The network architecture of Single-Shot Multibox Detector (SSD) with the MobileNet backbone.

an accelerator or a cloud server. However, this raises another important concern.

- **Research Question** (RQ): Is the MobileNet-Ssd model adequate to perform stand-alone detection and tracking in real-time on a Raspberry Pi?

As a result, the main goal of this investigation is to determine how well the model MobileNet-Ssd works on a constrained device without the assistance of cloud servers or accelerators, and where it stands compared to other models that are in use for the object detection task.

4. Methodology

To achieve the objectives of this study, a specific paradigm in addition to the research question was required to be identified for conducting experiments and recording results. Because new facts needed to be discovered experimentally or existing facts needed to be validated to quantitatively measure how effectively the model performed in the real world, the pragmatic approach was chosen (Antoniou, 2021). The purpose was to carry out experiments and quantitatively document the outcomes.

To address this, a human follower robot was designed to investigate the deep learning-based object detection model MobileNet-Ssd. The stand-alone object detection performance of the MobileNet-Ssd model was investigated on a constrained device, specifically the Raspberry Pi, to more fully comprehend how well the robot can track objects in real-time. In this investigation, the pre-trained MobileNet-Ssd model available in the libraries was utilized to perform object detection. Further, the detection was limited to the 'Person' class to ensure that the model only detects humans.

Rather than attempting to explain how it is or is not practicable, this investigation focuses on assessing if the MobileNet-Ssd model is adequate to be deployed on a Raspberry Pi for an object detection task from an ontological perspective in terms of philosophical use (Merrill, 2011). In other words, the work mainly is focused on the outcomes or results of the model's real-time performance rather than the underlying processes.

Further, it was also necessary to choose the tracking-by-detection approach, which employs an object detection model, rather than alternative single object tracking algorithms, which execute tracking using other image processing techniques and do not rely on the detection model. This is mainly because these algorithms commonly fall under the category of detection-free trackers. As a result, they would be ineffective in investigations that focus solely on an object detection model.

Figure 2 depicts the overall research process by displaying each phase that needed a decision and how the overall research plan was developed.

4.1. Contributions

This investigation intended to document the outcomes so that they are available to both present and future researchers who are using the model, as well as to those who want to use a Raspberry Pi or other constrained device to undertake deep learning-based object detection and tracking. This might help in bridging the action knowledge gap by giving evidence-based insights that can influence future actions and decision-making.

This investigation tries to assess the practical applicability of the MobileNet-Ssd model through real-time deployment. This includes determining the model's peak-memory usage, suitability for use on Raspberry Pi, and its potential to be used as a base model in the process of model development for constrained devices. Additionally, this study might be a useful starting point and foundational research for researchers who want to expand and improve upon the model.

Furthermore, the model was also compared to other state-of-the-art models to see where it stands in comparison to other models when considering this type of constrained application. The key decisions made are thoroughly explained, along with their rationale, in the subsection that follows.

4.2. Design of experiments

To determine the cause-and-effect relations that assist in understanding the real-world scenario, an information-gathering investigation must be carried out. There aren't many precise experimental design procedures for robotics, which can indicate their appropriateness in terms of the usage of a particular object detection model. On the other hand, robots may be enhanced through careful experimental design, which would also aid in bench-marking research and creating standards for various deep-learning models. This practice becomes extremely crucial when working with robots powered by constrained devices because recurrence studies may become more common due to a lack of comparisons. Hence, an attempt has been made to design an

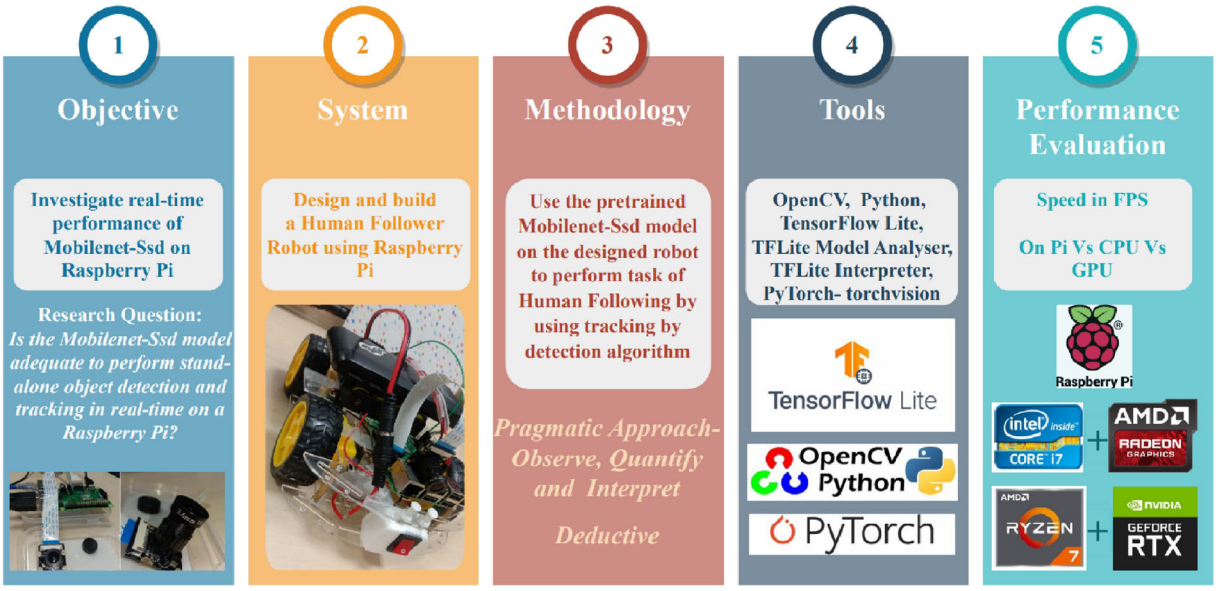


Figure 2. Illustration of the overall methodology and research strategy.

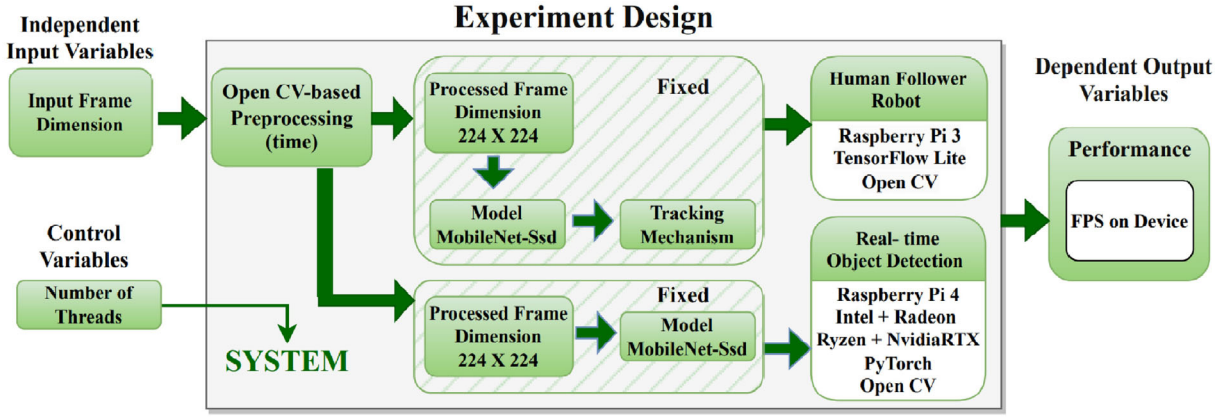


Figure 3. Experiment design depicting independent, dependent, and control variables in the system.

experiment and conduct quantitative comparisons of the model's performance to better comprehend the MobileNet-Ssd model's practical applicability. Figure 3 illustrates the full system with the independent, dependent and control variables chosen in the design of the experiments.

The input frame dimension as obtained from the camera may cause a change in the model's performance and impact various other factors. Hence, it is considered as an independent variable. The dimensions of the model are preset as they are obtained from the library, however, matching the dimensions of the camera input with that of the model is a usually overlooked factor that may significantly affect the model's performance. These are the variables that are tweaked in this investigation. The model's speed is mostly influenced by these input dimensions, along with numerous other parameters; hence, the speed of the model is regarded as the dependent variable, and it is therefore observed and recorded. The main purpose is to determine whether the model's speed in FPS will be sufficient for real-time applications. Therefore, the FPS on-device is regarded as the primary dependent variable that is investigated.

Aside from the independent and dependent variables, any factors that potentially influence the findings should be either controlled or fixed. Lack of control over significant variables may result in the inability to demonstrate that they did not affect the results. As a result, comprehending the system's cause-and-effect interactions will help in making more informed choices. Therefore, the model and the tracking mechanism were fixed to ensure that they do not have influence on the comparisons involving the input dimensions and FPS. Any other factor related to model training or model parameters is beyond the scope of this investigation and must

therefore be treated simply as an extraneous variable.

The relationship between the variables in the system when using the TensorFlow framework is illustrated in Figure 4. It also depicts how the experiment was organized and how the selection of the control variables were determined. Figure 5 illustrates the variables in the system being investigated while using the PyTorch library. Because the task of the human following is not feasible in this scenario without the necessary library support, evaluating the model's performance would be tricky; thus, the decision was made to compare the model's detection performance with other state-of-the-art models to provide a thorough understanding of how suitable the model is on the Raspberry Pi. Further, the number of threads utilized was considered as an additional control variable since it could have a substantial influence on the system (Rosebrock, 2015).

Due to the constraints imposed by the underlying frameworks, a proper comparison is not feasible between them, and this could prove to be a challenging task in the future. However, the model's applicability using both TensorFlow and PyTorch is investigated here to understand the model's potential usability and identify areas of improvement.

During real-time detection, the camera feed defines the input frame dimensions. However, a change in the scene and object position in the real-time feed may impact the model output unless each trial is given the same frame sequence. As a consequence, it was decided to determine the model's performance by changing input dimensions using recorded video sequences from four separate camera types that are often utilized with this model. This makes it easier to record and differentiate the effects of this input dimension on the model speeds. Table 1 lists the details of the videos captured and Figure 6 compares the videos that were recorded for the experiments, in terms of their duration and size. Care was taken to ensure that there was no multiple object presence during the recording, as this investigation does not aim to address aspects relating to illumination, occlusion and deformation, and it was essential to maintain these factors constant when comparing the model's performance across devices. This step was essential for neutralizing the effect from the input frame sequence, which influences many other factors that could directly or indirectly impact the detection and tracking speed. Hence, for a given set of video sequence, the changes in these parameters remained constant.

Table A1 (Appendix A) lists the specifications of the devices used in this study to collect input data or power the robot, as well as non-resource-constrained devices used for performance comparisons. To make sure that the observed performance is due to utilizing a constrained device, like the Raspberry Pi and not due to the model employed, two more devices were selected for the experiment design 2 using PyTorch. This consisted of a desktop computer with an Intel i7 CPU and an AMD Radeon R7 GPU and an Asus laptop with an AMD Ryzen 7 CPU and Nvidia RTX 3050 GPU.

5. Design of the human-follower robot

This section gives detail on the components of the human-follower robot design. The robot was constructed using a Raspberry Pi 3 to carry out the human-following task. Even though building a robot is simple enough for anyone, a brief description of the robot is included for evaluating the model, since the design of the robot can also influence

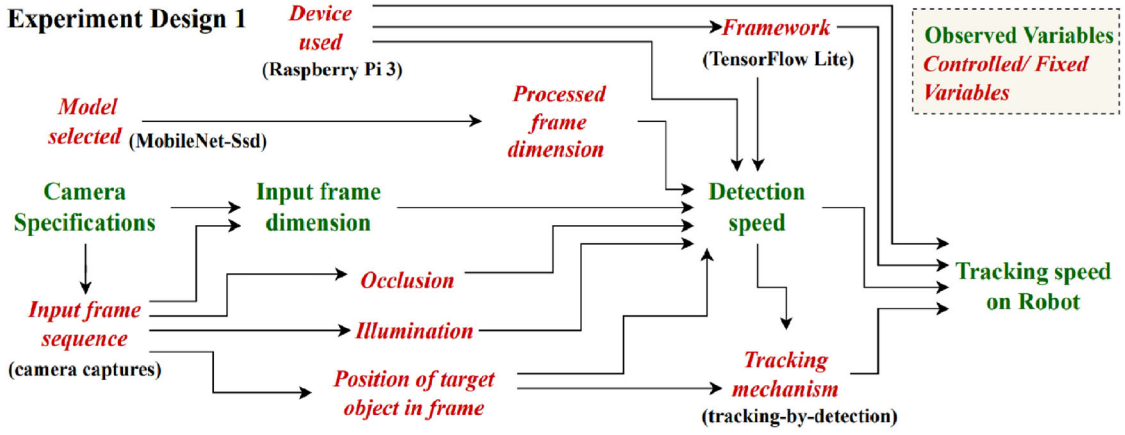


Figure 4. Variable relationship when using TensorFlow support: experiment design 1 (note: controlled/fixed variables have been set to neutralize their effect on the observed variable to provide a fair comparison across trials).

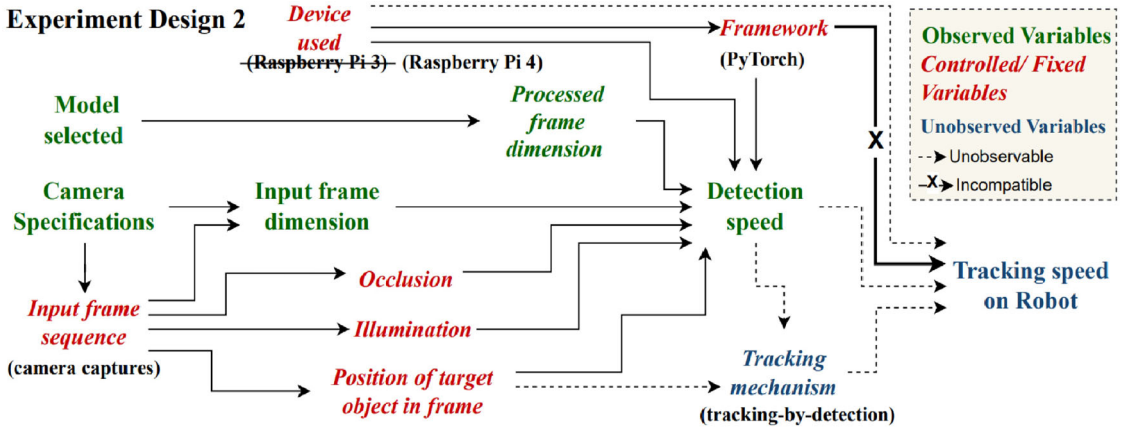


Figure 5. Variable relationship when using PyTorch support: experiment design 2 (note: because the Pi 3 was insufficient to run PyTorch, the device was updated to the Pi 4. PyTorch also lacked an onboard interpreter, which is necessary for tracking).

Table 1. Input data: videos captured for the experiment.

	Camera specifications	Captured on	Device resolution	Duration (s)	Size (MB)
Video_1	3.6 mm IR 1080p Pi Cam	Raspberry Pi 4 with VNC, remote desktop sharing over Wi-Fi	1080p@30fps	33	12.215
Video_2	720 P HD camera	Asus Laptop	720p@60fps	24	21.77
Video_3	Triple AI (13 MP, f/2.2, PDAF; 8 MP, f/2.2, 16 mm (ultrawide); 2 MP, f/2.4, (depth))	Vivo Y12 Smartphone	1080p@30fps	25	27.03
Video_4	8.0 MP AF	Samsung Galaxy A7 Tablet	1080p@30fps	29	60.09

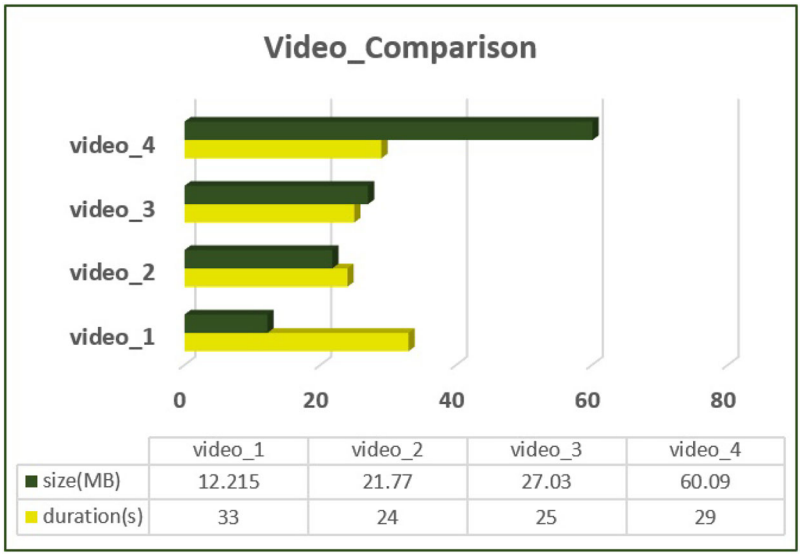


Figure 6. Video comparison.

performance. The entire system may be decomposed into three modules: the processing module, the controller module and the object detection and tracking module. The interactions between the three are shown in Figure 7. The robot's interface enables precise programming control of its physical movement.

The object detection and tracking module is the module in charge of the robot's vision, and it deals with the software and programming aspects that actually perform the detection and tracking logic, as well as the necessary logic for the robot's movements. This module works with the use of the MobileNet-Ssd model for performing the detection. The controller module is responsible for the mechanical aspects of functioning the robot car as well as providing interfaces for connected components. This

module is in charge of driving the robot and controlling its speed. Finally, the processing module consists of an embedded microcontroller on which the vision task is performed. The Raspberry Pi 3 board with a camera is used here. It connects the controller module to the object detection and tracking logic by storing, communicating and providing interfaces.

5.1. Robot structure

The Robot was structured as a four-wheeled car and measured 25 × 16 cm (Figure 8). The smart car 4WD chassis kit, which can sufficiently accommodate the Raspberry Pi board, controllers, motors and batteries, was used to assemble the car. Four 3.7-V Li-ion rechargeable batteries were wired in series to power the controller. This high voltage was necessary to pull the weight of the car. The Pi was powered by a separate power bank with a 5-V output. The final weight of the car after assembly was 950 g (Figure 8(c)).

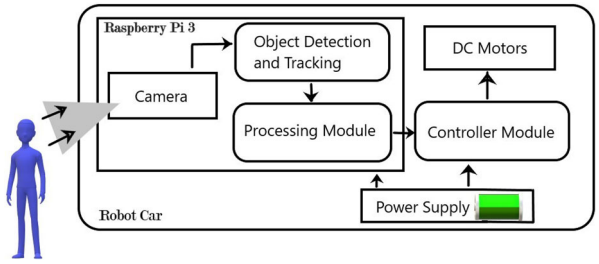


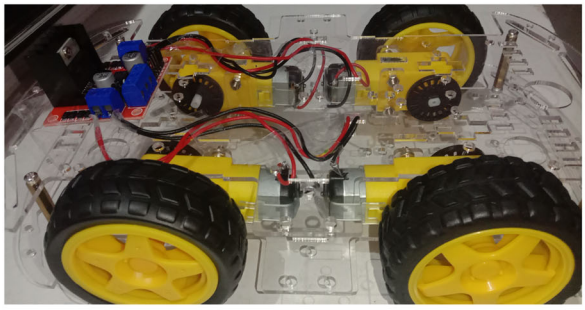
Figure 7. Modular design: depicting the interactions between the processing module, the controller module, and the object detection and tracking module.

5.2. Controller module

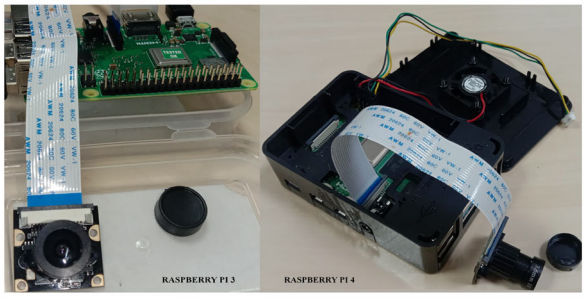
The L298N H-Bridge controller board was used to control the motors' speed and direction. It took a 12 V + battery power supply and converted it to the required voltage to power the left and right motors. The motors on each side were connected in parallel to the controllers' two output ports as shown in Figure 8(a). The motor direction and speed were controlled by connecting the PWM (Pulse Width Modulation) enable pins to the input pins.

5.3. Processing module

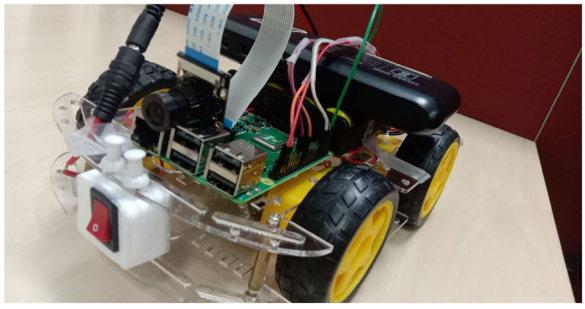
The processing module consisted of the Raspberry Pi 3 (Model B+) with a Pi Camera as shown in Figure 8(b). On the device, the Raspbian/Debian buster operating system with 32-bit armv7l architecture and distribution was installed. The Python version used was 3.7.3, and the opencv-contrib-python version was 4.1.0.25. The GPIO pins of the Raspberry Pi served as the interface for the communication between the processing and the controller modules. The connections between the two devices are shown in Figure 9. A 1080p camera, which is compatible with Pi, was



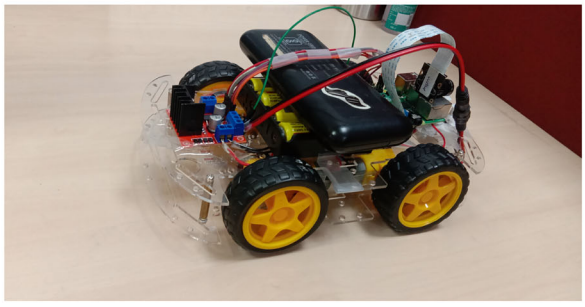
(a) Assembly with Bridge



(b) Raspberry Pi with Picam



(c) Robo Car Assembly



(d) Robo Car- Side view

Figure 8. System design and component requirements.

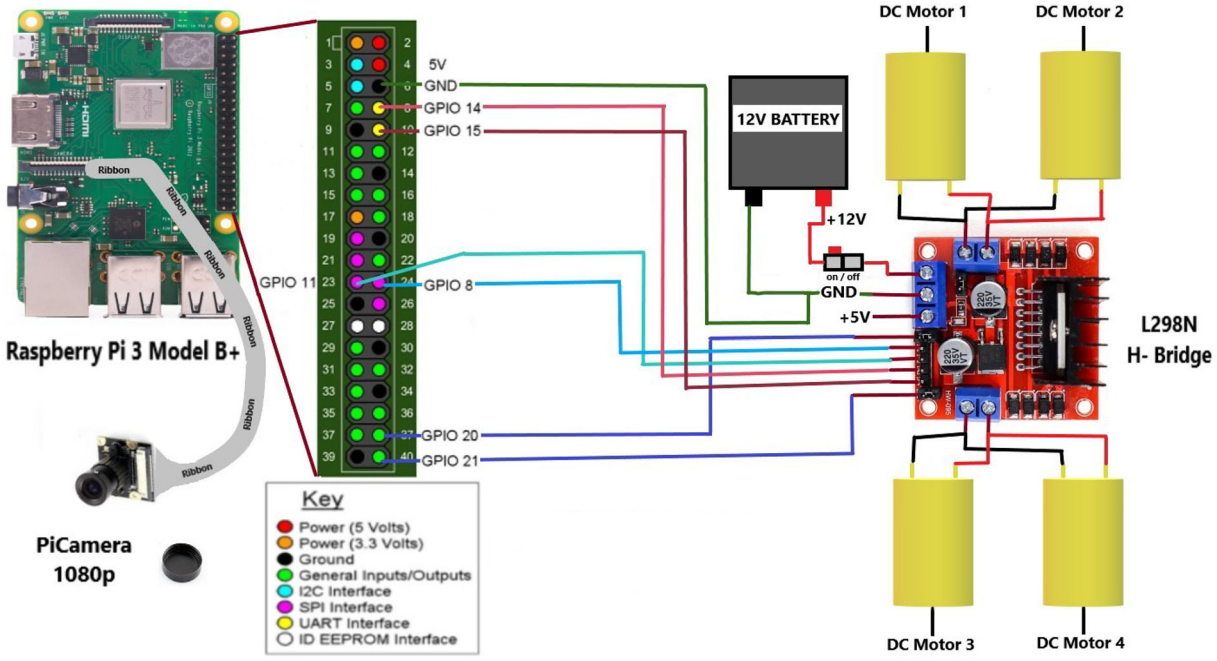


Figure 9. Schematic diagram - human follower robot, depicting the physical connections between the GPIO pins of the Raspberry Pi board, the L298N H-Bridge, the DC motors, and the battery.

used to stream the front view of the robot car in real-time (Figure 8(c)). The camera resolution during the captures was set to 720p with a capture frequency of 30fps. The captured frame sequence was resized as per the input size of the model.

5.4. Tracking by detection mechanism

A robot view was designed with elements that assist the tracking process to make the robot track objects using the tracking-by-detection approach. Figure 10 depicts the designed view of the robot. The state of the robot is shown in the top right corner of the view, and the speed in FPS is shown in the top left. The center of the view has a box that is defined by the threshold. This defines the threshold region. Overlapping of the real-time tracking on the camera feed with the robot view was achieved using the Python and OpenCV libraries.

Algorithm 1. Tracking Mechanism Used To Detect And Follow A Human

Input: The frame *f1* from the camera input feed and the trained object detection model, MobileNet-Ssd.

Output: Detected object on the input frame with yellow bounding box and motor directions for the wheels with speed.

Data: Initialize the global variables.

1: Choose the Model *Model* ← *Model_name.tflite*

2: Set label values *lbl* ← *labels.txt* ▷ Labels used in the Model

3: Set tolerance zone *tolerance* ← 0.1

4: Set the object classes to be detected *valid_object* ← ['*person*'] ▷ To detect only Human

5: Set the motor speed to maximum and set the motor values to OFF.

6: **while** True **do**

7: $f1 \leftarrow$ Read the Current Frame from the Camera Feed

8: $objs, labels \leftarrow OBJECT\_DETECTION(Model, f_1)$

9: *TRACK*(*objs*, *labels*)

10: **end while**

Procedure for TRACK:

1: **procedure** TRACK(*objs*, *labels*)

2: **if** *len*(*objs*) = 0 **then**

3: Stop the Robot and *Return*

4: **end if**

5: Use two variables, *k* and *flag* set to zero, for accessing the detected objects from *objs*

6: **for** obj in objs **do**

7: Get the *obj* label and store in *lbl* and set $k \leftarrow count(valid\_object)$

8: **if** $k \geq 1$ **then**

9: Read bounding box coordinates *BBOX*(*obj*)

10: Set *flag* ← 1 indicating the presence of Person and *Break*.

11: **end if**

12: **end for**

13: **if** *flag* = 0 **then**

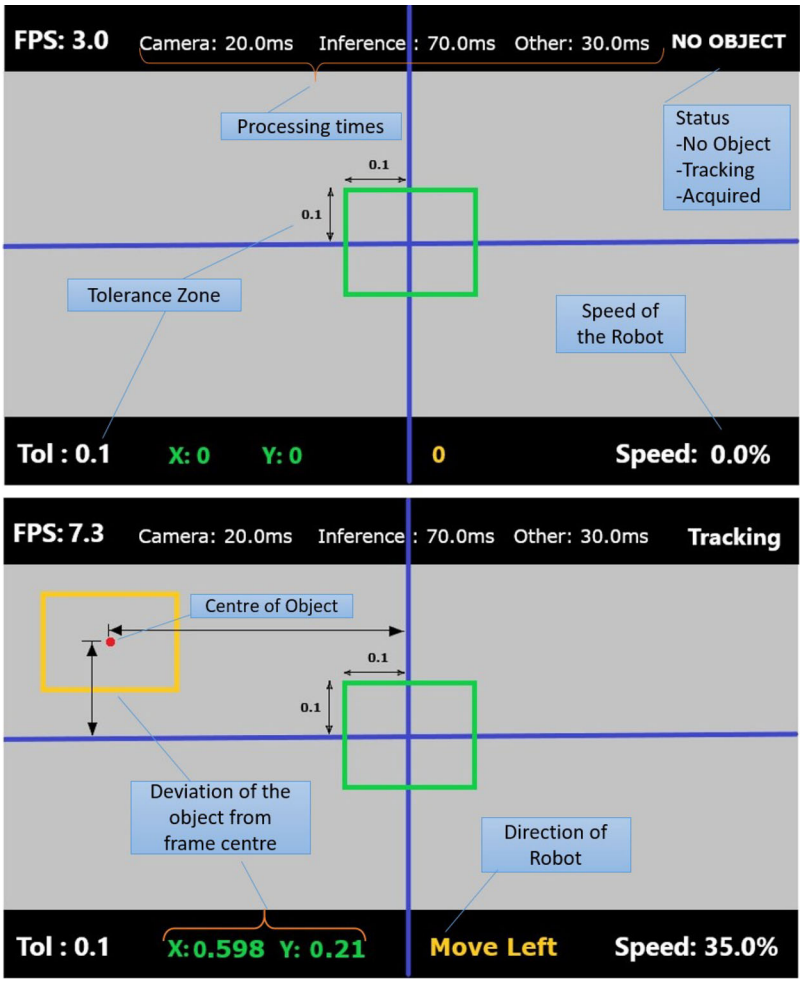


Figure 10. Design of the robot view; the center of the frame has a box defined by the threshold, the status of the robot is displayed in the top right corner, and the speed in FPS is displayed in the top left corner.

14: No object is Present *Return*
15: **end if** ▷ Object is present and it is a person
16: Calculate the height and width of the object using bounding box coordinates
17: Calculate the center of the bounding box
18: Calculate the deviation from the center of the frame.
19: Move the Robot using *MOVE_ROBOT*()
20: **end procedure**

Procedure: OBJECT_DETECTION(Mod, Frame) Applies the Model in Mod on the Frame and returns the list of detected objects and their corresponding labels.

Procedure: MOVE_ROBOT() Responsible for the movement of the Robot. Moves the Robot until the object center lies within the *tolerance* zone.

Procedure: BBOX(obj) Responsible for calculating and returning the bounding box coordinates for the given object *obj*.

Depending upon the position of the object in the frame, the robot can be in one of five states: 'Start', 'Tracking', 'Acquired', 'No Object', or 'Stop'. The transitions within the states are depicted in the state transition diagram in Figure 11. In the middle of the frame, a threshold region is defined. This determines when an object is considered to be in the 'Acquired' state. When an object is detected outside of this threshold region (shown in green box in Figure 10), the state is changed to 'Tracking', and the robot is moved using a tracking mechanism to get the detected object back inside the threshold region.

Algorithm 1 demonstrates the logic behind the tracking mechanism used to move the robot. The variable *valid_object* is set to 'Person' to track only objects of this class as the main task of this robot is to follow humans. This ensures that tracking is confined to this object class and that the detection of any other object class is interpreted as the absence of any detection. If no object is detected in the frame, then the state of the robot changes to 'No Object'. The tracking procedure runs in an infinite loop to continuously identify and track the object in the frames until stopped by the user; i.e. when there

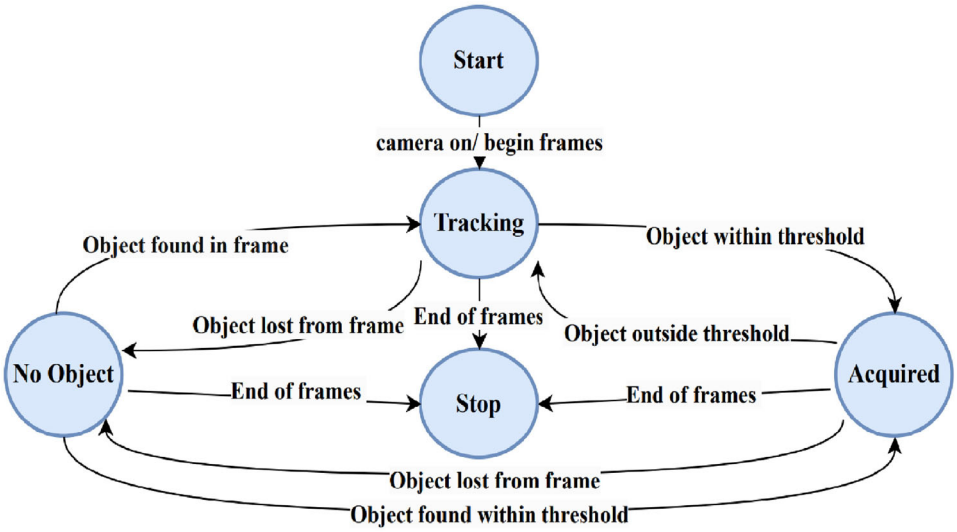


Figure 11. State transition diagram; 'start' and 'stop' refer to the begin and end of frames, 'no object' and 'acquired' states cause the robot to stop, and 'tracking' state causes the robot to move and follow the human.

are no more frames to process and the robot is in the 'Stop' state. It should be noted that any object that the model can recognize (based on the dataset it was trained on) can be tracked by the algorithm. Even though in this case it is restricted to tracking only the 'Person' class, it is also capable to identify several other object classes.

6. Result analysis

This section comprises all the outcomes of the experiments. First, the speed and memory of the model were evaluated in real time using the designed robot system. The model's performance on the Raspberry Pi in terms of its average speed was compared to that on other devices based on experiment design 1 using the library support offered by TensorFlow. The subsections 6.1 and 6.2 contain the details of the same.

Following that, investigations were carried out on the Pi using the captured video sequences to demonstrate how the results on the Raspberry Pi camera differed from those on other cameras when using PyTorch, following experiment design 2. In this scenario, tracking was not achievable, therefore only detection speed could be compared to indicate probable relevance on constrained devices. A summary of this is provided in the subsection 6.3.

6.1. Analysis of the model on the robot in real-time

When planning to use deep learning on a constrained device like the Raspberry Pi, one of the most common concern is 'What is the peak memory utilization of the model when operating with any given data?'. In edge computing applications, where the model must operate on the edge, as in a virtual environment, this is extremely important. This becomes substantially more crucial when many models are operating on the same device. It is essential to know the peak memory utilization of a model in order to prevent many models from reaching their peaks at the same time, which might cause the system to crash. As a result, the quantized MobileNet-Ssd model's memory utilization was examined in order to estimate the model's peaks.

As the model deployed in this tracking algorithm is a quantized '.tflite' model, the memory use and structure of the model could be visualized using TensorFlow's TFLite Model analyzer. Figure 12 shows the plot of the memory usage of the model for each layer, and Figure 13 shows the tensor table output from the model analyzer, showing the shape of inputs and their corresponding size in RAM.

The peak memory usage as found by the model analyzer was 1204.224 Kb. This is the amount of memory utilized by the model regardless of the device. Given that the Raspberry Pi used on the robot had just 1GB of RAM, this appears to be a feasible model option. It is also worth noting that memory spikes occur in the first few layers, and the model gradually approaches a stable memory utilization of 200 Kb, indicating that the likelihood of a crash due to a memory surge from this model is fairly low. This is the major reason the model is so adaptable to constrained applications. Furthermore, this architecture allows for more flexibility in accommodating features and attributes in subsequent

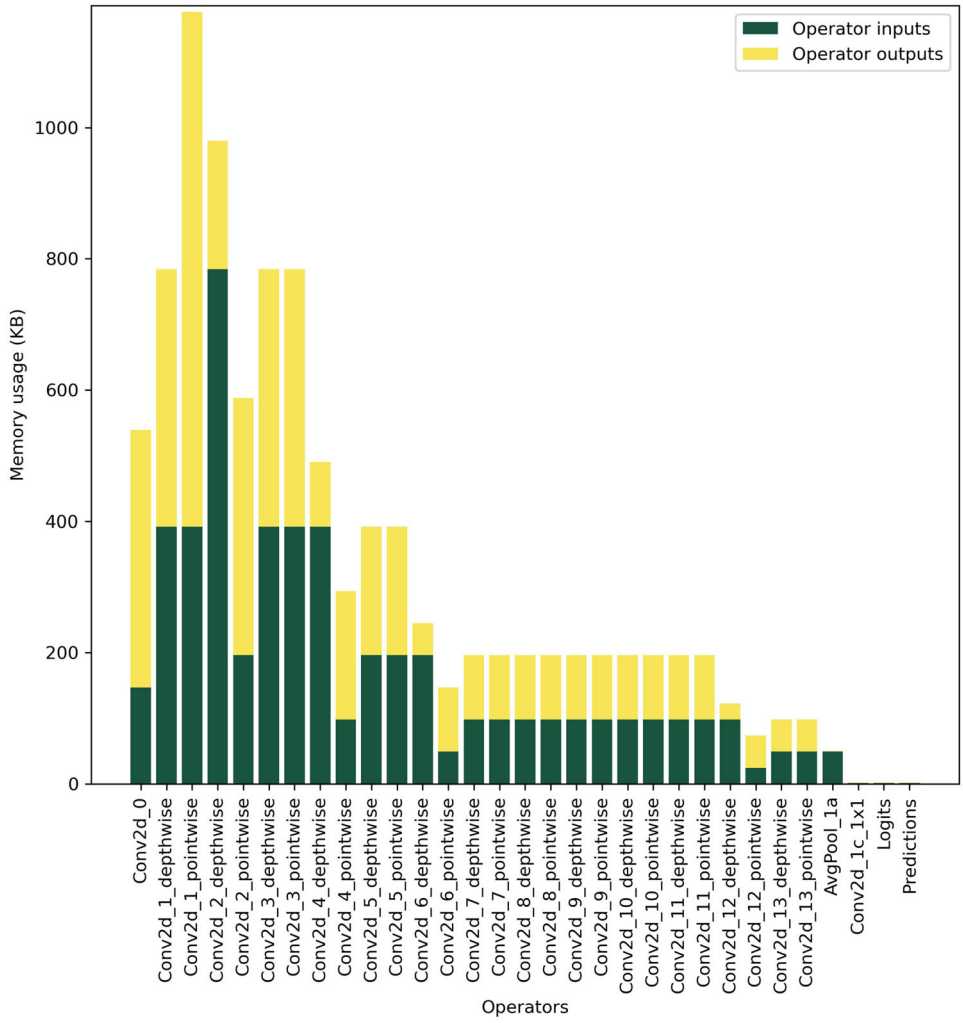


Figure 12. MobileNet-Ssd quantized model memory usage plotted using the TFLite Model analyzer.

layers without exceeding the peak memory usage, making it an excellent choice as a base model, for researchers seeking to develop new models for constrained usage.

Despite the model's apparent suitability for the application, its real-time performance on the designed robot during the trials was less than ideal. Although the designed robot was performing decently, there was a lag in detection, which further delayed the tracking process, increasing the likelihood of the tracking system losing focus of the object. This is due to the use of the Raspberry Pi without any support from accelerators or servers. The tracking speed was around 0.5-1.2 FPS, resulting in a slowdown in the detection phase. This speed was too low for performing any kind of real-time following. While using accelerators, the model's performance on a similar robot design has been said to achieve speed up to 9 FPS (Saini, 2021). However, the credibility of such outcomes is uncertain and may require additional investigation.

Identifying the extraneous factors that led to this inadequate performance might be critical finding for future efforts that address this issue from an epistemological perspective. A few snapshots of the robot during real-time testing are shown in Figure 14. As long as the person moved slowly, the robot tracked and followed. Whenever the person moved faster, the robot would lose focus of the person and come to a halt until the person moved back into focus and was detected.

Even though the MobileNet-Ssd model detects the person in real-time, the performance in terms of speed is quite low in any real-time applications involving Raspberry Pi, as evidenced by this implementation. This is especially true if the task at hand involves more than just object detection and requires another mechanism to be run in addition to the model's interpretations. The use of accelerators is one option for improving performance, but having a model that can function without additional processing aid would make the robot more cost-effective

Id	Tensor	Shape	Size in RAM (B)
0	AvgPool_1a	(1, 1, 1, 1024)	1,024
1	Conv2d_1c_1x1	(1, 1, 1, 1001)	1,001
4	Logits	(1, 1001)	1,001
7	Conv2d_0	(1, 112, 112, 32)	401,408
9	Conv2d_10_depthwise	(1, 14, 14, 512)	100,352
13	Conv2d_10_pointwise	(1, 14, 14, 512)	100,352
15	Conv2d_11_depthwise	(1, 14, 14, 512)	100,352
19	Conv2d_11_pointwise	(1, 14, 14, 512)	100,352
21	Conv2d_12_depthwise	(1, 7, 7, 512)	25,088
25	Conv2d_12_pointwise	(1, 7, 7, 1024)	50,176
27	Conv2d_13_depthwise	(1, 7, 7, 1024)	50,176
31	Conv2d_13_pointwise	(1, 7, 7, 1024)	50,176
33	Conv2d_1_depthwise	(1, 112, 112, 32)	401,408
37	Conv2d_1_pointwise	(1, 112, 112, 64)	802,816
39	Conv2d_2_depthwise	(1, 56, 56, 64)	200,704
43	Conv2d_2_pointwise	(1, 56, 56, 128)	401,408
45	Conv2d_3_depthwise	(1, 56, 56, 128)	401,408
49	Conv2d_3_pointwise	(1, 56, 56, 128)	401,408
51	Conv2d_4_depthwise	(1, 28, 28, 128)	100,352
55	Conv2d_4_pointwise	(1, 28, 28, 256)	200,704
57	Conv2d_5_depthwise	(1, 28, 28, 256)	200,704
61	Conv2d_5_pointwise	(1, 28, 28, 256)	200,704
63	Conv2d_6_depthwise	(1, 14, 14, 256)	50,176
67	Conv2d_6_pointwise	(1, 14, 14, 512)	100,352
69	Conv2d_7_depthwise	(1, 14, 14, 512)	100,352
73	Conv2d_7_pointwise	(1, 14, 14, 512)	100,352
75	Conv2d_8_depthwise	(1, 14, 14, 512)	100,352
79	Conv2d_8_pointwise	(1, 14, 14, 512)	100,352
81	Conv2d_9_depthwise	(1, 14, 14, 512)	100,352
85	Conv2d_9_pointwise	(1, 14, 14, 512)	100,352
87	Predictions	(1, 1001)	1,001
88		(1, 224, 224, 3)	150,528

Figure 13. MobileNet-Ssd quantized model tensor information obtained through TFLite Model analyzer.

and applicable. Other factors that may have a significant impact on real-time performance include the size of the input frame obtained from the Raspberry Pi camera, the library from which we imported the model, the dataset on which the model was pre-trained, the peak memory usage of the tracking method used and the number of threads handling the process.

6.2. Performance comparisons of input data quality on the robot

The tracking mechanism was investigated on four distinct videos recorded on different devices to examine the effect of camera image quality on performance (Refer: Table 1). The results have been recorded in Table 2. It is evident that improved tracking rate emerges from higher-quality input. While tracking with a lower-resolution camera is more error-prone, the number of inaccurate detection decreases as the pixel count in the input video increases. Increasing input resolution, on the other hand, produces a large drop in detection speed since more processing needs to be performed, slowing down the process.

The successful tracking rate for the model is calculated mathematically as follows:

$$Tracking\ rate = \left(\frac{n}{N}\right) \times 100 \tag{1}$$

where, n is the number of successfully tracked frames, N is total frames processed.

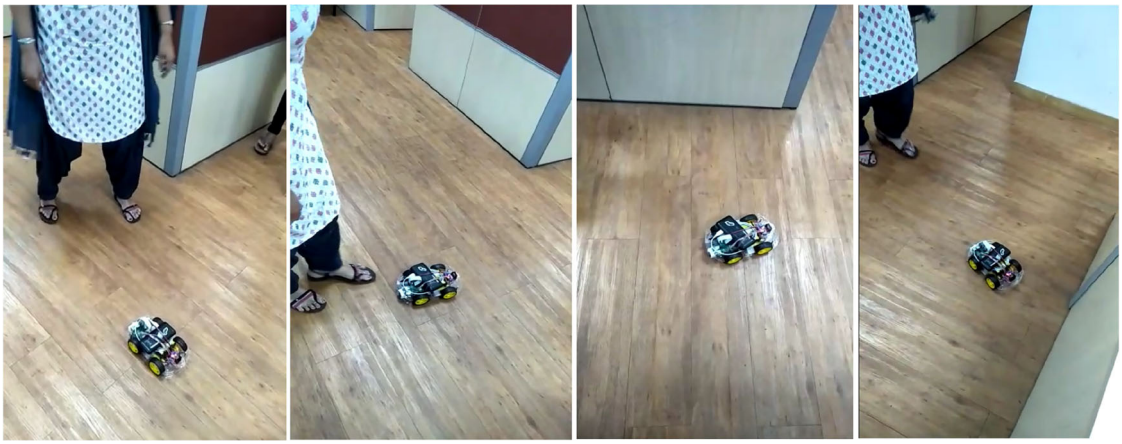


Figure 14. Snapshots of the robot during real-time testing.

Several factors impact the tracking mechanism, including illumination, occlusion and the presence of multiple objects. The MobileNet-Ssd model struggles to recognize objects, especially when the illumination changes dramatically. As a result, tracking is compromised. Furthermore, when multiple objects are present, the MobileNet-Ssd model appears to identify smaller objects better than larger objects, which in some cases may not be ideal and may result in unsuccessful tracking even though detection appears to be successful. Figure 15 shows a few snapshots of the robot view when tested on Video_1 and Video_2; Figures 16 and 17 show the same when tested on Video_3 and Video_4, respectively.

First, it can be seen that Video_1 provides tracking at a rate of 70% whereas Video_4 provides tracking at 100% even when multiple objects are present. This demonstrates how the model advances as the input image quality improves. Furthermore, as the MobileNet-Ssd model was trained to identify 'Person' even when the object in question is only partially visible, it can be seen that it is resilient to partial occlusions. However, because of the low-quality images captured by the PiCam, the model frequently fails to detect changes in illumination.

Second, the dimensions of Video_2 and Video_3 are the same, with one in portrait mode and the other in landscape mode. However, the resolution in dpi of Video_3 is higher than that of Video_2, allowing it to achieve a tracking rate of 99.47% as opposed to 72.06% in Video_2. It is also disputable that the enhanced tracking is a result of utilizing portrait mode, which mostly covers a person's whole silhouette. To avoid this element from influencing the outcome, it was ensured that the whole silhouette of the person was visible both in Video_2 as well as Video_3 to neutralize the effect of occlusion.

Finally, it is not always possible to obtain high-quality images in constrained edge devices. The results on Video_1 (which is obtained through a Raspberry Pi Camera) demonstrate that there is still a significant amount of work to be done to improve the model detection accuracy, model speed, model size optimization and robustness to the external factors that can affect the tracking rate.

The best results were obtained when tested on Video_4, which was recorded using a Samsung Galaxy Tablet. This could be useful in edge applications, but it is more likely to be useful in scenarios of federated learning rather than stand-alone detection. It is therefore advisable to make use of MobileNet-Ssd on inputs from such handheld devices compared to using Raspberry PiCam, as the model performs accurate detection in Video_3 and Video_4, both of which were recorded on smart Android-based handheld devices. However, this holds good only for applications with no time constraints, as the speed is dramatically reduced in this case, and is not recommendable for applications involving object tracking.

6.3. Performance comparisons with other devices

In order to assess the MobileNet-Ssd model's performance across non-constrained devices and determine the exact percentage of processing performance that would be lost by using a constrained device, the following devices were used: an Intel i7 desktop computer with an AMD Radeon R7 GPU, an Asus laptop with an AMD Ryzen 7 CPU and an Nvidia RTX 3050 GPU and a Raspberry Pi 4 board. The evaluation was conducted without the use of a tracking mechanism.

Table 2. Tracking results obtained for the different input video sequences.

Parameters	Video_1	Video_2	Video_3	Video_4
Frame size	640 × 480	1280 × 720	720 × 1280	1080 × 1920
Frame rate (FPS)	30	30	30	30
Camera orientation	Landscape	Landscape	Portrait	Portrait
Aspect ratio	4:3	16:9	9:16	9:16
Total frames (N)	1250	673	764	847
Successfully tracked (n frames)	881	485	760	847
Lost track of object (frames)	369	188	004	000
Multiple objects present	×	✓	✓	✓
Illumination	✓	×	✓	✓
Occlusion	✓	×	×	×
Tracking rate (%)	70.48	72.06	99.47	100
Processing rate (FPS)	1.2–1.5	0.8–1.1	0.9–1.1	0.5–0.8

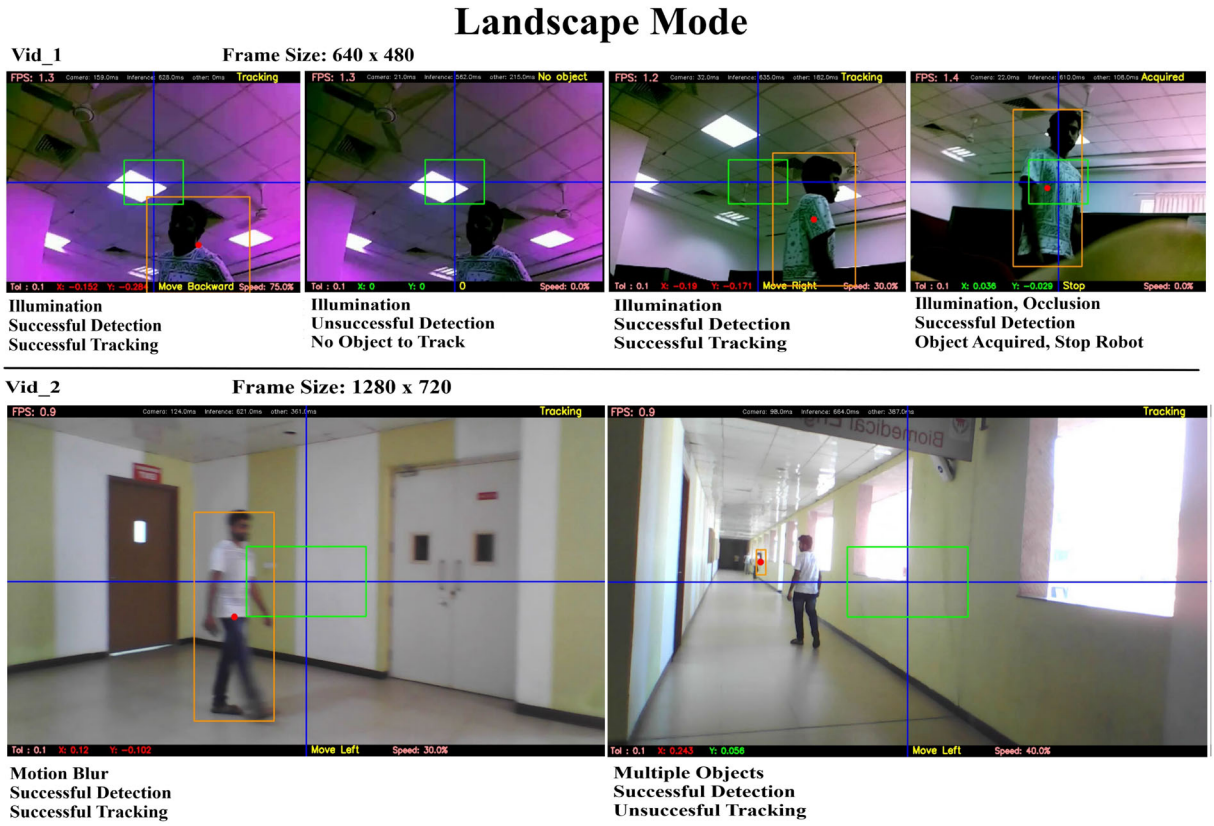


Figure 15. Sample frames of the robot view tested on Video_1 and Video_2.

The model was tested on the video sequences listed in Table 1 to investigate how the input frame size and quality affected the model's speed. Each experiment consisted of ten runs, with the recorded data averaged throughout the runs to see how the speed was influenced. Furthermore, all experiments were repeated twice, first without specifying any threads and then again with two threads (Rosebrock, 2015), to check whether multi-threading had any influence on performance. This was done to investigate if shifting I/O operations to a separate thread affected performance. The average values of FPS and the number of iterations the model ran on the entire video were recorded and a full factorial design table of the same is shown in Tables 3 and 4, respectively.

A graph was plotted to further visualize the aforementioned outcomes, as illustrated in Figure 18. This shows the relationship between the number of iterations the model took to run on the provided set of frames from a video, and the average performance speed observed.

Figure 19 shows the effect of threading on all four videos for all runs, as recorded on Raspberry Pi 4. When no threading is used, it is clearly visible that the model's performance fluctuates significantly. This can also differ depending on how many threads are used by default on a given device when explicit threading is not requested. Because the Raspberry Pi uses all of its cores by default, threading improves the model's stability but substantially affects its

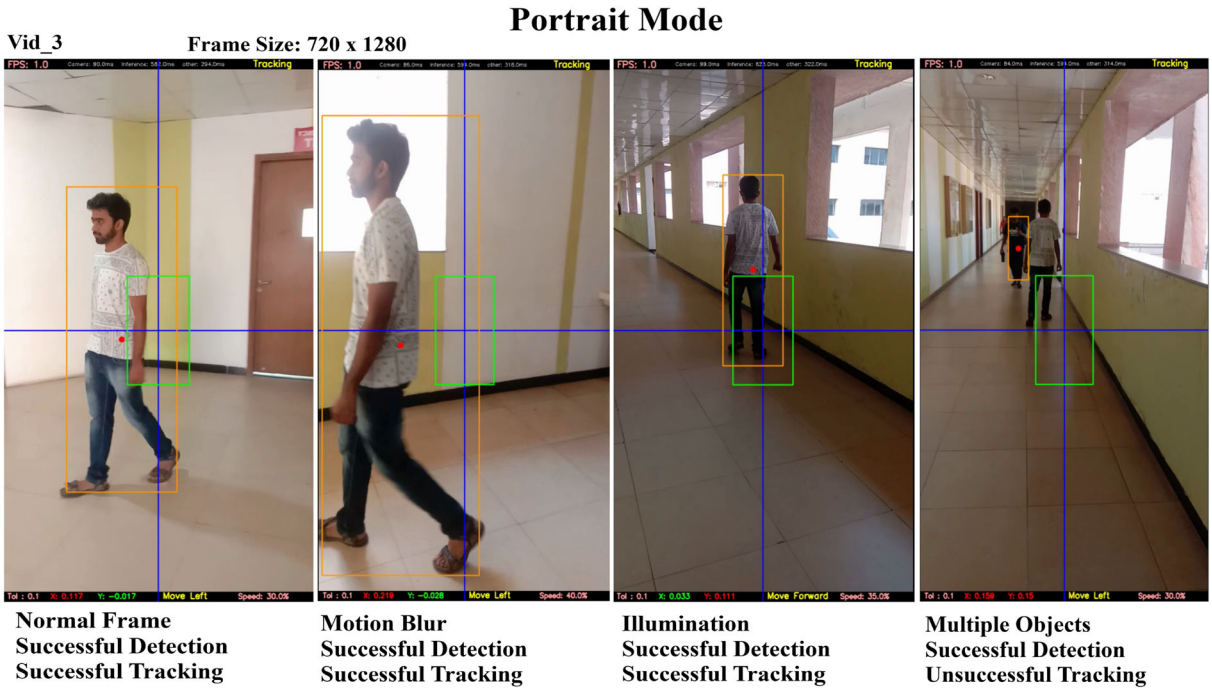


Figure 16. Sample frames of the robot view tested on Video_3.

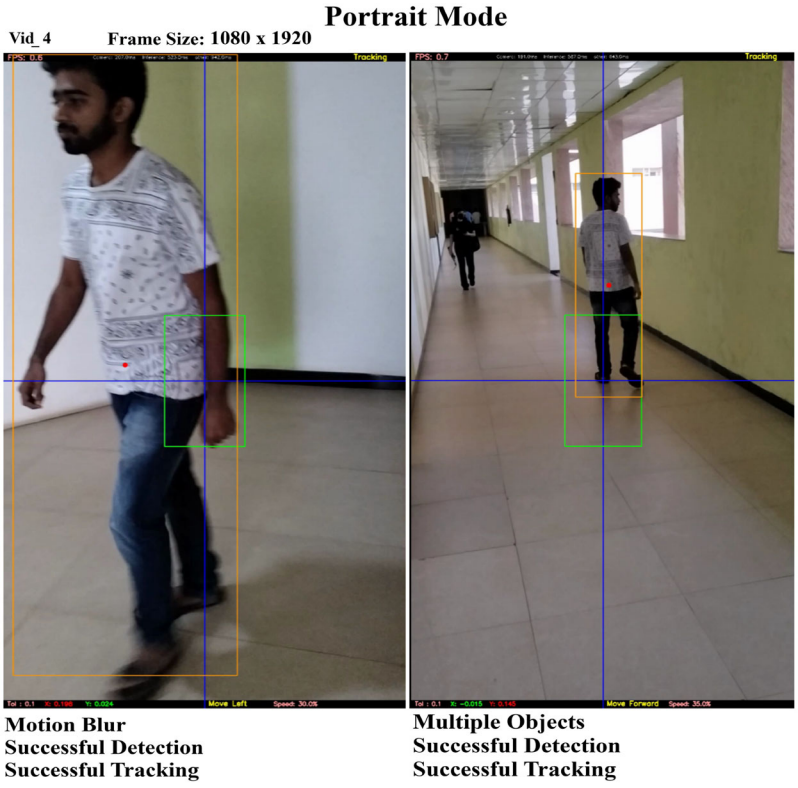


Figure 17. Sample frames of the robot view tested on Video_4.

performance. Except for small variations in video length, it is evident how the number of iterations required to parse the video changes substantially when threading is not employed. This is quite evident when comparing the Raspberry Pi camera input to a high-resolution input (Video_1 Vs Video_4).

A better depiction is provided in Figure 20(a), which additionally shows how threading affects the model's performance on either of the two non-constrained devices. Threading appears to provide almost constant model performance across all devices, although at the cost of performance loss.

Table 3. Full factorial design table recording FPS using PyTorch.

	Control variables							
	No threading				Two threads			
Device	Video_1	Video_2	Video_3	Video_4	Video_1	Video_2	Video_3	Video_4
Raspberry Pi 4	6.83	2.39	2.35	1.02	4.21	1.44	1.43	0.63
AMD Ryzen 7 CPU + Nvidia GPU	5.32	1.57	1.99	0.92	4.55	1.56	1.46	0.63
Intel i7 CPU + AMD Radeon R7 GPU	4.47	1.48	1.47	0.65	4.52	1.40	1.43	0.63

Table 4. Full factorial design table recording number of iterations per video using PyTorch.

	Control variables							
	No threading				Two threads			
Device	Video_1	Video_2	Video_3	Video_4	Video_1	Video_2	Video_3	Video_4
Raspberry Pi 4	141	230	259	455	200	337	383	847
AMD Ryzen 7 CPU + Nvidia GPU	168	318	303	847	200	337	384	847
Intel i7 CPU + AMD Radeon R7 GPU	201	336	382	847	200	336	382	847

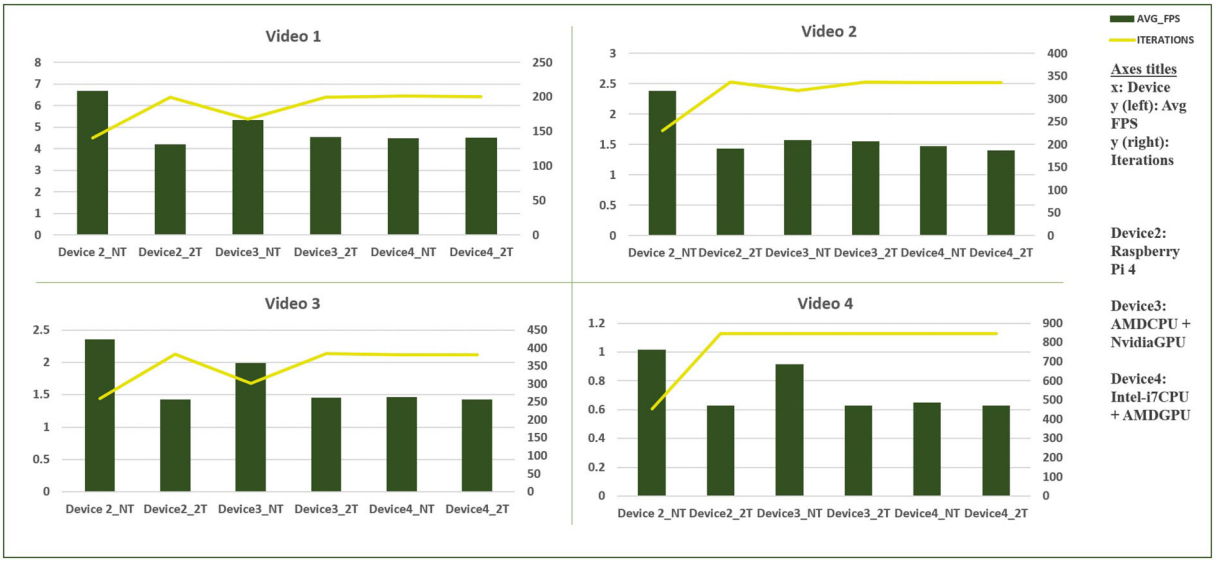
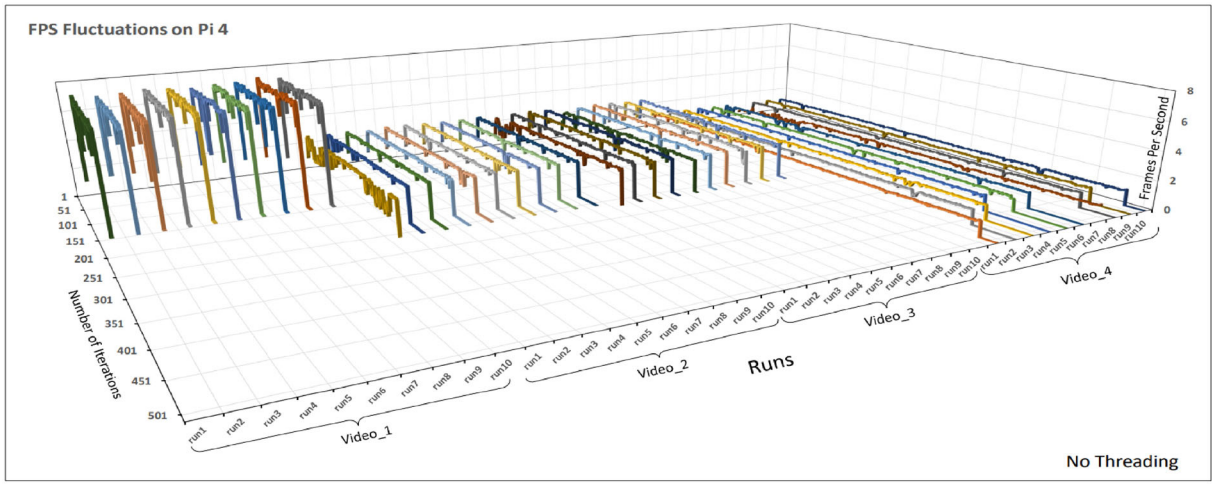


Figure 18. A graph showing how overall speed and iterations are related across the devices.

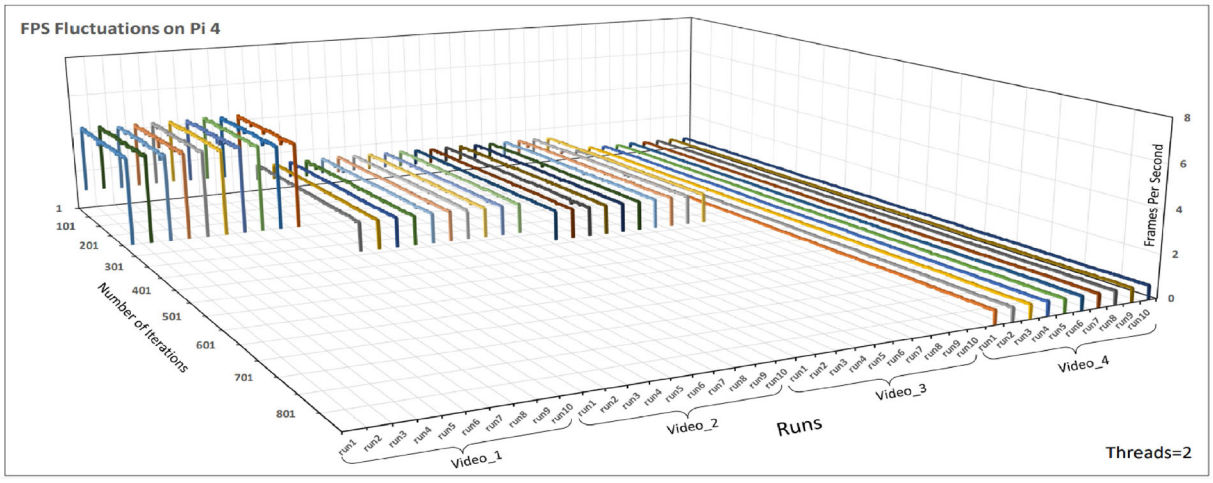
In addition, it can be observed that threading has a more significant impact on the Raspberry Pi than it does on the other two non-constrained devices.

The size and quality of the video, as observed, have a major influence on the model's performance. This explains why downsizing the input frames to the lower size required by the model (in this case 224×224) might use up device resources if a frame arrives from any of the high-quality cameras. For a better comprehension of this effect, the Video_4 was preprocessed offline to reduce its height and width to 224. Device performance for the video processed offline significantly improved, with all devices exceeding 30 FPS. The comparison of the same is shown in Figure 20(b). This demonstrates the impact of online preprocessing on performance, which is directly related to the camera's input dimensions. This also explains why the model performs better on the PiCam recording than other cameras used to record.

As a result, the performance of a particular model may differ depending on the camera used to record real-time streaming when installed on a robot. This extraneous component might impact the model's performance, which could be undermined while performing trials due to the lack of standardization for reporting outcomes in this field of study. Despite being the longest of the four videos, it is obvious that Video_1 produces the best results across the line, on both constrained and unconstrained devices. This is due to the Raspberry Pi Camera's reduced quality, which collects less information per frame and lessens the device's load.

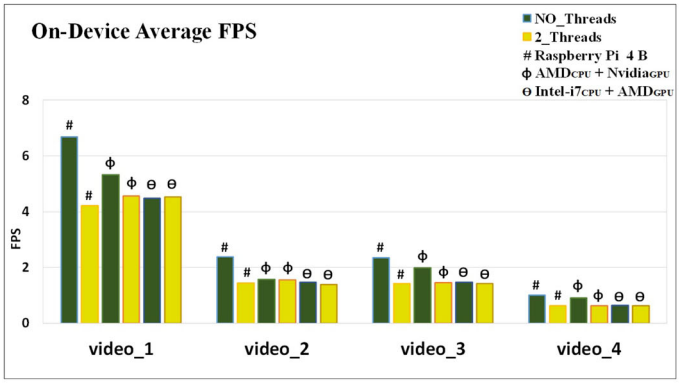


(a) Without Threads

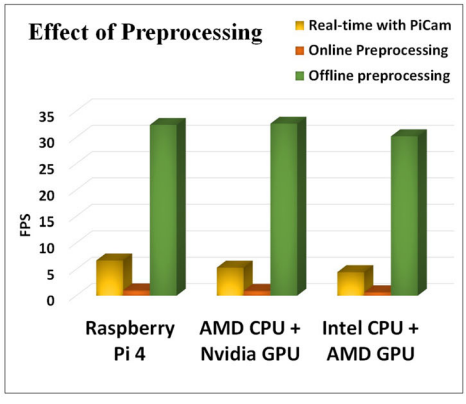


(b) With Two Threads

Figure 19. FPS fluctuation recorded on Raspberry Pi 4.



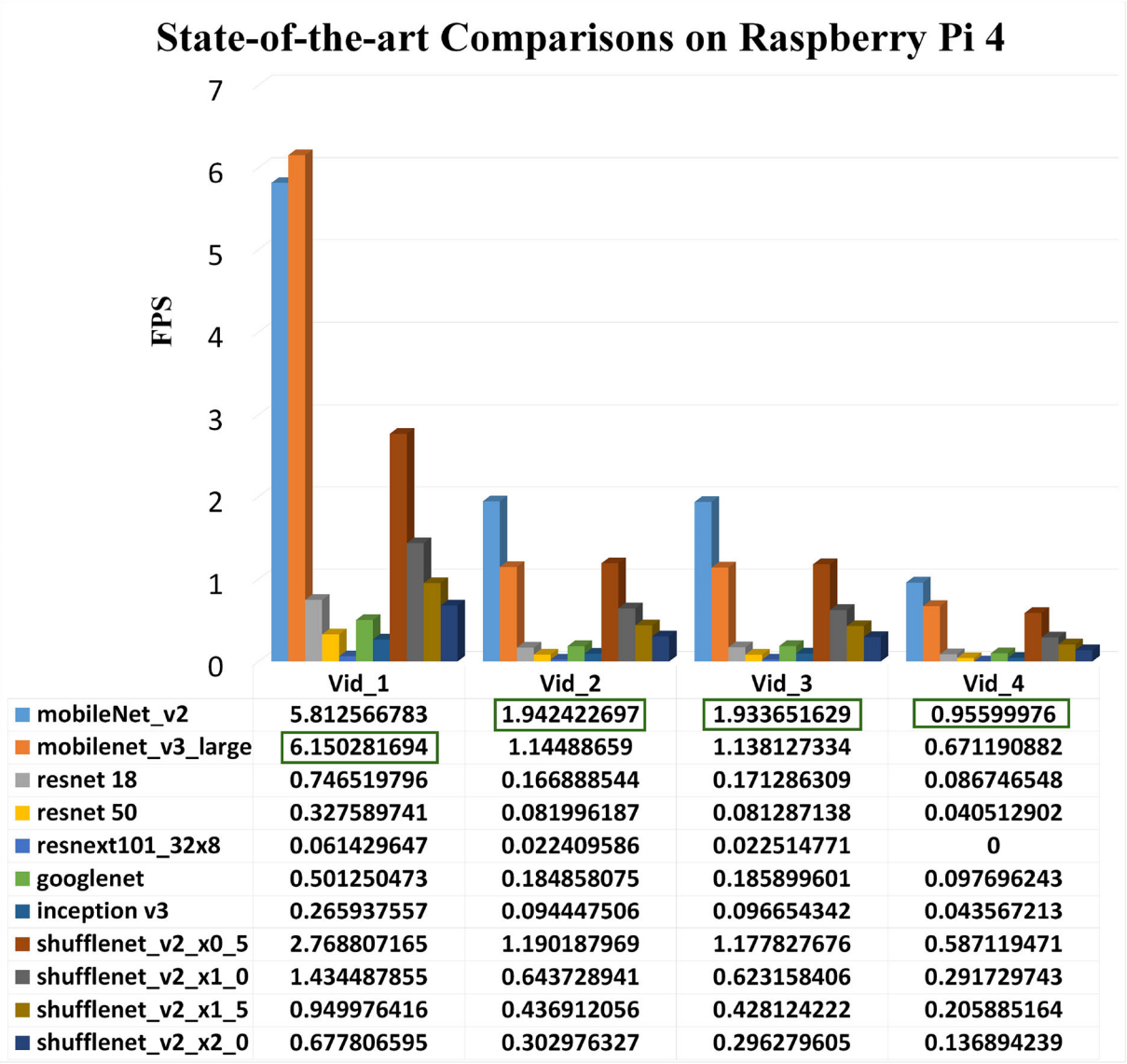
(a) On device comparison of average FPS



(b) Effect of preprocessing on the speed

Figure 20. Performance comparisons.

Figure 21 shows a comparison of the model's performance on the Raspberry Pi 4 to that of other state-of-the-art models. Furthermore, the performance of all these models was assessed across all three devices, to understand why the MobileNet model is regarded as the best alternative for constrained device usage. Figure 22 shows the graph depicting the same.



	Vid_1	Vid_2	Vid_3	Vid_4
mobileNet_v2	5.812566783	1.942422697	1.933651629	0.95599976
mobilenet_v3_large	6.150281694	1.14488659	1.138127334	0.671190882
resnet 18	0.746519796	0.166888544	0.171286309	0.086746548
resnet 50	0.327589741	0.081996187	0.081287138	0.040512902
resnext101_32x8	0.061429647	0.022409586	0.022514771	0
googlenet	0.501250473	0.184858075	0.185899601	0.097696243
inception v3	0.265937557	0.094447506	0.096654342	0.043567213
shufflenet_v2_x0_5	2.768807165	1.190187969	1.177827676	0.587119471
shufflenet_v2_x1_0	1.434487855	0.643728941	0.623158406	0.291729743
shufflenet_v2_x1_5	0.949976416	0.436912056	0.428124222	0.205885164
shufflenet_v2_x2_0	0.677806595	0.302976327	0.296279605	0.136894239

Figure 21. State-of-the-art model comparisons of detection speed in FPS on a Raspberry Pi 4.

7. Discussion

Even though the MobileNet-Ssd model detects the person in real-time, the performance in terms of speed is quite low in any real-time applications involving Raspberry Pi, as evidenced by this implementation. This is especially true if the task at hand involves more than just object detection and requires another mechanism to be run in addition to the model's interpretations. The use of accelerators is one option for improving performance, but having a model that can function without additional processing aid would make the robot more cost-effective and applicable. Other extraneous factors that may have a significant impact on real-time performance include the size and quality of the input frame obtained from the Raspberry Pi camera, the library from which the pretrained model is imported, the dataset on which the model was pretrained, the peak memory usage of the model used and the number of threads handling the process.

As the performance of the MobileNet-Ssd model seems to be inadequate in real-time object tracking, this necessitates the development of more advanced object detection models in the future that can infer at greater speeds. However, the MobileNet-Ssd model appears to perform effectively when compared to other available models, making it an appealing option for those who plan to train and deploy custom deep learning-based models on a Raspberry Pi.

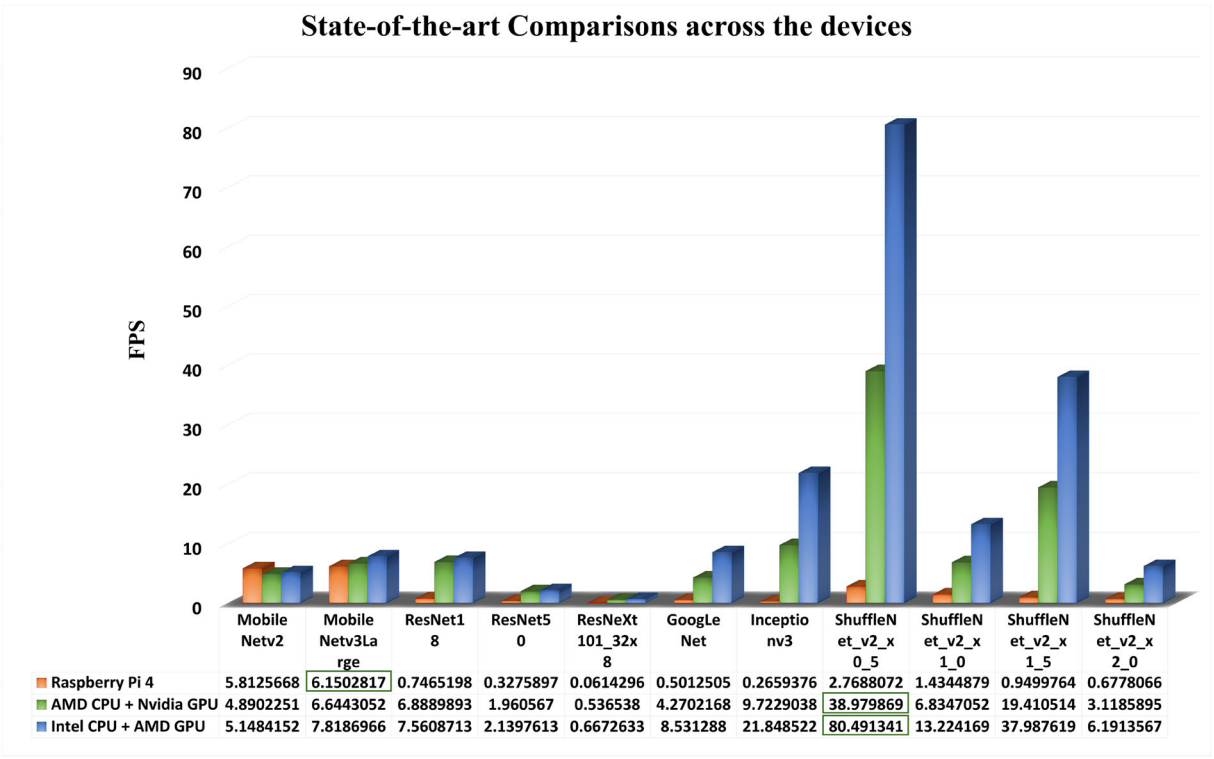


Figure 22. State-of-the-art model comparisons of detection speed in FPS across all the devices for the Pi camera feed.

This investigation produced the following insights during the deployment of the MobileNet-Ssd model:

- The model is not ideal for doing stand-alone detection, but it is unquestionably the most efficient option available among pretrained models.
- The model is better suited for less demanding tasks or scenarios that include an accelerator with the Raspberry Pi, or in a federated learning system.
- TensorFlow library may be of great assistance given the demonstrated ability to do tracking and detection on a constrained device with as little as 1 GB RAM. Therefore, it serves as a better choice for pretrained model deployment compared to PyTorch.
- PyTorch offers quantization engines for the model, which might help with deployment on the Raspberry Pi. However, it required more memory and processing power than the TensorFlow counterpart. The fact that Raspberry Pi library support is still in its early stages could explain why.
- This model's peak memory utilization is only 1204 Kb, making it an excellent choice for constrained applications. As a result, researchers interested in creating their custom models can choose the MobileNet-Ssd as a starting point for their neural architectural search.
- The model appears to be a good match for camera feeds from smart devices, which proves why it is an excellent candidate for Android applications.
- When the input frame dimension from the camera and the input dimension of the model are remarkably close, the model performs its best.

When comparing the support provided by both libraries, it should be noted that TensorFlow provides a lightweight interpreter that can be easily accommodated on the Pi, whereas PyTorch requires the entire library and quantization engine to be installed on the device, which may not be ideal when dealing with constrained devices. Future research should try to provide comparable PyTorch-supporting libraries that can efficiently leverage a stored model to execute on-the-fly detection.

7.1. Inference drawn

The observations recorded during the investigations are summarized in Table 5 to collectively answer the research question. Based on the foregoing observations and analyses, it can be implied that the MobileNet-Ssd model is a viable option for performing tasks such as human following on the Raspberry Pi, provided that the application is not time-

Table 5. Summary of observations.

Experiment design 1: framework used- TensorFlow	
1. Peak memory usage of the quantized model:	1204.224 Kb
2. Real-time detection speed on robot without using external accelerators:	0.5–1.2 FPS
3. Tracking-by-detection average speed on robot:	
Pi camera feed (640 × 480):	1.35 FPS
Asus TUF laptop camera feed (1280 × 720):	0.95 FPS
Vivo smartphone camera feed (720 × 1280):	1.0 FPS
Samsung Galaxy A7 tablet camera feed (1080 × 1920):	0.65 FPS
4. Tracking rates on robot:	
Pi camera feed (640 × 480):	70.48%
Asus TUF laptop camera feed (1280 × 720):	72.06%
Vivo smartphone camera feed (720 × 1280):	99.47%
Samsung Galaxy A7 tablet camera feed (1080 × 1920):	100%
Experiment design 2: framework used- Pytorch	
5. Average speed of detection on Raspberry Pi 4 (without tracking mechanism):	
Pi camera feed (640 × 480):	6.68 FPS
Asus TUF laptop camera feed (1280 × 720):	2.39 FPS
Vivo smartphone camera feed (720 × 1280):	2.35 FPS
Samsung Galaxy A7 tablet camera feed (1080 × 1920):	1.02 FPS
6. Average speed without threading using Picam:	6.68 FPS
7. Average speed with two threads using Picam:	4.21 FPS
8. Average number of iterations to parse 1250 frames with no threading:	141 iterations
9. Average number of iterations to parse 1250 frames with two threads:	200 iterations
Effect of preprocessing time on the model	
10. Average speed of detection on feed from Device_6, tablet (1080 × 1920) when preprocessing to 224 × 224 done offline :	
On Raspberry Pi 4:	32.36 FPS
On Laptop AMD Ryzen7 CPU + Nvidia RTX 3050 GPU	32.6 FPS
On Desktop Intel i7 CPU + AMD Radeon 7 GPU	30.2 FPS
11. Total number of iterations to parse 847 frames with offline preprocessing:	
On Raspberry Pi 4:	26 iterations
On Laptop AMD Ryzen7 CPU + Nvidia RTX 3050 GPU	26 iterations
On Desktop Intel i7 CPU + AMD Radeon 7 GPU	28 iterations

constrained and an external accelerator is available to boost processing speed.

When compared to object tracking, the model may be more useful for tasks like image classification that do not require additional computations. The model also functions best when given offline preprocessed input video feeds, making it appropriate for surveillance applications involving high-resolution cameras in which the input feed could be preprocessed offline before being sent to the model. This, however, cannot be true in real-time streaming when online processing is necessary.

Furthermore, when using the Raspberry Pi as the processor, the model performs better on input feeds from handheld devices and can be a good choice when designing robotic or edge applications that use distributed and federated learning. However, the model appears to require more improvements for stand-alone object detection and object tracking tasks in order to be useful in real-time applications.

8. Limitations

The investigations were carried out with the most constrained device the library could accommodate and hence did not attempt to bring any comparison across the library or the constrained devices used. The main aim of the investigation was to evaluate the model's practical applicability in real-time and hence, the pretrained MobileNet-Ssd model was deployed from libraries. Therefore, this study does not cover any aspects of model training or testing and is only concerned with deployment-related concerns. This is particularly significant given the approach of the study, which concentrates on answering the 'what is' aspect of question rather than addressing the 'why' or 'how' aspects. As a result, the performance of the MobileNet model is of particular interest. It is to be noted that the reasons why it is so, as well as how it may be improved are answered using narrative synthesis, which lacks strong arguments as the implementation specifics of the model and frameworks are outside the scope of this work.

Because the emphasis was exclusively on one type of class, this approach falls within the area of single-object detection and does not attempt to address multi-object tracking challenges. Further, as this study focused on analyzing an object detection model, any other single object tracking approaches that do not rely on object detection to perform tracking were not explored. Hence, this study limits itself to employing the tracking-by-detection methodology and does not attempt to compare its results with any other state-of-the-art single-object tracking approaches.

Another tough area to enhance the model in, is detection accuracy. This is feasible when the object

that is being tracked is not in the list of objects that the model has been trained to recognize. Typically, the class list from the well-known MS-COCO (Lin et al., 2014) dataset or the Imagenet (Russakovsky et al., 2015) dataset makes up this list of classes. In such a case, either some transfer learning must be applied to the model to train it on a custom dataset that can contain the desired class, or new models constructed on the MobileNet architecture must be developed and trained on the custom dataset. However, this investigation solely emphasizes the deployment aspects of the model, and any such potential improvements to the model must be separately evaluated and validated and hence, considered outside the scope of this investigation.

9. Conclusion and future scope

The MobileNet-Ssd is a well-known object detection model that is widely recognized for its lightweight applications today. This work attempted to employ the model to detect and follow a human by building a robot powered by a Raspberry Pi. The implementation demonstrated that, although being a lightweight model, the MobileNet-Ssd model alone may be suitable but not entirely satisfactory to be deployed on a Raspberry Pi and may operate better with the support of an accelerator in both cases where TensorFlow and PyTorch were utilized.

Human-following robots are one type of robot that has a wide range of applications. Other uses for such robots include but are not limited to, serving customers in malls or public spaces, caring for children and the elderly with special requirements, or following a particular kind of vehicle. However, to support these challenging tasks, there is a need for a stronger model that can continue to function in real-world scenarios. Even if the performance observed was not as envisioned, overall, the MobileNet-Ssd model provides a great starting point when designing applications with resource constraints. As a result, there is a great demand for lightweight models that can run on constrained platforms like the Raspberry Pi.

The peak-memory usage structure of the MobileNet-Ssd model proved to be extremely beneficial for constrained device deployments. Hence, this model can be a useful base model for any potential neural architectural search of models aimed for constrained device deployments.

It cannot be denied that utilizing boards like the Jetson Nano may have been a better method to utilize this model than pairing a Raspberry Pi with an accelerator. However, the ability to do independent object detection with just a Raspberry Pi would be a highly cost-effective option. A unique solution to this can be a future endeavor that has the potential to change numerous industries.

Investigating the model for multi-object detection is another higher-level problem that may be examined; nevertheless, it is recommended that it be carried out on a non-constrained device as deployments of complex models on constrained devices may appear to be a research that is still in its early stages.

Finally, it is concluded that the MobileNet-Ssd model is a good model option for constrained device deployments due to its architectural form, size, peak-memory usage and library support when compared to other available options. Since this model requires external accelerator support, future research on stand-alone object detection using Raspberry Pi should focus on developing new models similar to MobileNet-Ssd while designing the model architecture by attempting to reduce the model size and layers without affecting the peak memory usage delivered by this model.

Acknowledgment

The authors would also like to thank Mr. Vishwas G. Kini, Research Scholar, MIT, Manipal, for his assistance in capturing video data for the experiments.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the Manipal Academy of Higher Education Dr. T.M.A Pai Research Scholarship under Research Registration No: 200900143-2021.

About the authors

Vidya Kamath is currently pursuing her PhD in the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India. She obtained her B.E. degree from Visvesvaraya Technological University, Belagavi, Karnataka, India (2011); and her MTech degree from Kurukshetra University, Haryana, India (2014). She has six years of teaching experience and three years of research experience. She has published papers in international journals and conferences. Her major research interests include computer vision, deep learning and the

Internet of Things. Other areas of interest include artificial intelligence, computer graphics, embedded application development and web designing.

Renuka A obtained her BE and MTech degrees from Mysore University and PhD from NITK Surathkal, India. Currently, she is working as a professor in the Department of Computer Science and Engineering at Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India. Her areas of interest include cryptography and network security, cryptanalysis, image processing and pattern recognition, machine learning, deep learning, 4G LTE systems and mobile ad-hoc networks. She has published several papers in international journals and conferences. She is also the reviewer for articles submitted to various reputed journals and conferences.

ORCID

Vidya Kamath http://orcid.org/0000-0002-7396-886X
Renuka A http://orcid.org/0000-0001-6511-9780

Data availability statement

The data, metadata and codes used in this research work will be made available upon reasonable request to the corresponding author's email-id.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., & Ghemawat, S. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org/

Abbaszadeh Shahri, A., Asheghi, R., & Khorsand Zak, M. (2021). A hybridized intelligence model to improve the predictability level of strength index parameters of rocks. *Neural Computing and Applications*, *33*(8), 3841–3854. https://doi.org/10.1007/s00521-020-05223-9

Abbaszadeh Shahri, A., Shan, C., & Larsson, S. (2022). A novel approach to uncertainty quantification in groundwater table modeling by automated predictive deep learning. *Natural Resources Research*, *31*(3), 1351–1373. https://doi.org/10.1007/s11053-022-10051-w

Algabri, R., & Choi, M.-T. (2021). Target recovery for robust deep learning-based person following in mobile robots: Online trajectory prediction. *Applied Sciences*, *11*(9) , 4165. https://doi.org/10.3390/app11094165

Alzubi, J., Jain, R., Nagrath, P., Satapathy, S., Taneja, S., & Gupta, P. (2021). Deep image captioning using an ensemble of CNN and LSTM based deep neural networks. *Journal of Intelligent & Fuzzy Systems*, *40*(4), 5761–5769. https://doi.org/10.3233/JIFS-189415

Alzubi, O., Alzubi, J., Al-Zoubi, A., Hassonah, M., & Kose, U. (2022). An efficient malware detection approach with feature weighting based on Harris Hawks optimization. *Cluster Computing*, *25*(4), 2369–2387. https://doi.org/10.1007/s10586-021-03459-1

Antoniou, A. (2021). A pragmatic approach to the ontology of models. *Synthese*, *199*(3-4), 6645–6664. https://doi.org/10.1007/s11229-021-03085-9

Asheghi, R., Hosseini, S., Saneie, M., & Shahri, A. (2020). Updating the neural network sediment load models using different sensitivity analysis methods: A regional application. *Journal of Hydroinformatics*, *22*(3), 562–577. https://doi.org/10.2166/hydro.2020.098

Chen, S., Ren, Y., Friedrich, D., Yu, Z., & Yu, J. (2020). Sensitivity analysis to reduce duplicated features in ANN training for district heat demand prediction. *Energy and AI*, *2*, 100028. https://doi.org/10.1016/j.egyai.2020.100028

Chita-Tegmark, M., & Scheutz, M. (2021). Assistive robots for the social management of health: A framework for robot design and human–robot interaction research. *International Journal of Social Robotics*, *13*(2), 197–217. https://doi.org/10.1007/s12369-020-00634-z

Dipu, M., Hossain, S., Arafat, Y., & Rafiq, F. (2021). Real-time driver drowsiness detection using deep learning. *International Journal of Advanced Computer Science and Applications*, *12*(7), 844–850. https://doi.org/10.14569/IJACSA.2021.0120794

Ghaderi, A., Abbaszadeh Shahri, A., & Larsson, S. (2022). A visualized hybrid intelligent model to delineate Swedish fine-grained soil layers using clay sensitivity. *CATENA*, *214*, 106289.https://doi.org/10.1016/j.catena.2022.106289

Gupta, M., Kumar, S., Behera, L., & Subramanian, V. K. (2017). A novel vision-based tracking algorithm for a human-following mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(7), 1415–1427. https://doi.org/10.1109/TSMC.2016.2616343

Han, D., & Peng, Y. (2020). Human-following of mobile robots based on object tracking and depth vision. In *2020 3rd International Conference on Mechatronics, Robotics and Automation, ICMRA 2020* (pp. 105–109). https://doi.org/10.1109/ICMRA51221.2020.9398366

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *Mobilenets: Efficient convolutional neural networks for mobile vision applications*. https://arxiv.org/pdf/1704.04861.pdf

Islam, M., Hong, J., & Sattar, J. (2019). Person-following by autonomous robots: A categorical overview. *The International Journal of Robotics Research*, *38*(14), 1581–1618. https://doi.org/10.1177/0278364919881683

Jiang, L., Nie, W., Zhu, J., Gao, X., & Lei, B. (2022). Lightweight object detection network model suitable for indoor mobile robots. *Journal of Mechanical Science and Technology*, *36*(2), 907–920. https://doi.org/10.1007/s12206-022-0138-2

Kamath, V., & Renuka, A. (2023). Deep learning based object detection for resource constrained devices: Systematic review, future trends and challenges ahead. *Neurocomputing*, *531*, 34–60. https://doi.org/10.1016/j.neucom.2023.02.006

Karaman, O., Alhudhaif, A., & Polat, K. (2021). Development of smart camera systems based on artificial intelligence network for social distance detection to fight against covid-19. *Applied Soft Computing*, *110*, 107610. https://doi.org/10.1016/j.asoc.2021.107610

Kastner, L., Fatloun, B., Shen, Z., Gawrisch, D., & Lambrecht, J. (2022). Human-following and -guiding in crowded environments using semantic deep-reinforcement-learning for mobile service robots. In *Proceedings - IEEE International Conference on Robotics and Automation* (pp. 833–839). https://doi.org/10.1109/ICRA46639.2022.9812111

Khare, S., Palakkal, S., Hari Krishnan, T., Seo, C., Kim, Y., Yun, S., & Parameswaran, S. (2020). Real-time driver drowsiness detection using deep learning and heterogeneous computing on embedded system. In N. Nain, S. K. Vipparthi, and B. Raman (Eds.), *Communications in computer and information science, 1148 CCIS* (pp. 86–97). Springer Singapore. https://doi.org/10.1007/978-981-15-4018-9_8

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars (Eds.), *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), 8693 LNCS(PART 5)* (pp. 740–755). Springer International Publishing. https://doi.org/10.1007/978-3-319-10602-1_48

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. (2016). Ssd: Single shot multibox detector. In Leibe, Bastian and Matas, Jiri and Sebe, Nicu and Welling, Max (Eds.), *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), 9905 LNCS* (pp. 21–37). Springer International Publishing. https://doi.org/10.1007/978-3-319-46448-0_2

Malik, H., Fatema, N., & Alzubi, J. (2021). *Ai and machine learning paradigms for health monitoring system*. Springer. https://doi.org/10.1007/978-981-33-4412-9

Merrill, G. H. (2011). Ontology, ontologies, and science. *Topoi*, *30*(1), 71–83. https://doi.org/10.1007/s11245-011-9091-x

Movassagh, A., Alzubi, J., Gheisari, M., Rahimi, M., Mohan, S., Abbasi, A., & Nabipour, N. (2023). Artificial neural networks training algorithm integrating invasive weed optimization with differential evolutionary model. *Journal of Ambient Intelligence and Humanized Computing*, *14*(5), 6017–6025. https://doi.org/10.1007/s12652-020-02623-6

Murthy, C., Hashmi, M., & Keskar, A. (2021). Optimized MobileNet + SSD: A real-time pedestrian detection on a low-end edge device. *International Journal of Multimedia Information Retrieval*, *10*(3), 171–184. https://doi.org/10.1007/s13735-021-00212-7

Ou, S., Park, H., & Lee, J. (2019). Implementation of an obstacle recognition system for the blind. *Applied Sciences*, *10*(1), 282. https://doi.org/10.3390/app10010282

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32, pp. 8024–8035). Curran Associates. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Rosebrock, A. (2015, December). *Increasing raspberry pi fps with python and opencv*. https://pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Safadinho, D., Ramos, J., Ribeiro, R., Filipe, V., Barroso, J., & Pereira, A. (2020). UAV landing using computer vision techniques for human detection. *Sensors*, *20*(3), 613. https://doi.org/10.3390/s20030613

Saini, J. (2021). *Ai robot - object detection with tensorflow lite on raspberry pi—live-stream results on browser*. https://github.com/jiteshsaini/robotics-level-4/tree/main/earthrover/object_detection

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520). https://doi.org/10.1109/CVPR.2018.00474

Sanjay Kumar, K., Subramani, G., Thangavel, S., & Parameswaran, L. (2021). A mobile-based framework for detecting objects using ssd-mobilenet in indoor environment. In Peter, J. Dinesh, Fernandes, Steven L. & Alavi, Amir H. (Eds.), *Advances in intelligent systems and computing* (Vol. 1167, pp. 65–76). Springer Singapore. https://doi.org/10.1007/978-981-15-5285-4_6

Shafique, M., Theocharides, T., Reddy, V., & Murmann, B. (2021). Tinyml: Current progress, research challenges, and future roadmap. In *Proceedings - Design Automation Conference* (Vol. 2021–December, pp. 1303–1306). IEEE. https://doi.org/10.1109/DAC18074.2021.9586232

Shakeel, M., Bajwa, N., Anwaar, A., Sohail, A., Khan, A., & Ur Rashid, H. (2019). Detecting driver drowsiness in real time through deep learning based object detection. In Rojas, Ignacio, Joya, Gonzalo, & Catala, Andreu (Eds.), *Advances in Computational Intelligence,* Lecture Notes in Computer Science LNTCS 11506. (pp. 283–296). Springer International Publishing. https://doi.org/10.1007/978-3-030-20521-8_24

Singh, A., Kalaichelvi, V., & Karthikeyan, R. (2022). A survey on vision guided robotic systems with intelligent control strategies for autonomous tasks. *Cogent Engineering*, *9*(1), 2050020. https://doi.org/10.1080/23311916.2022.2050020

Yuan, J., Cai, J., Zhang, X., Sun, Q., Sun, F., & Zhu, W. (2021). Fusing skeleton recognition with face-TLD for human following of mobile service robots. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *51*(5), 2963–2979. https://doi.org/10.1109/TSMC.2019.2921974

Zhang, J., Xu, J., Zhu, L., Zhang, K., Liu, T., Wang, D., & Wang, X. (2020). An improved MobileNet-SSD algorithm for automatic defect detection on vehicle body paint. *Multimedia Tools and Applications*, *79*(31-32), 23367–23385. https://doi.org/10.1007/s11042-020-09152-6

Zhou, W., Dickenson, P., Cai, H., & Li, B. (2022). Human following for mobile robots. In Liu, Honghai and Yin, Zhouping and Liu, Lianqing and Jiang, Li and Gu, Guoying and Wu, Xinyu and Ren, Weihong (Eds.), *Intelligent Robotics and Applications Lecture notes in computer science, 13455 LNAI* (pp. 660–668). Springer International Publishing. https://doi.org/10.1007/978-3-031-13844-7_61

Appendix A

Table A1. Device specifications.

	Device_1	Device_2	Device_3
Device name	Raspberry Pi 3 Model B+	Raspberry Pi 4 Model B	ASUS TUF Gaming Laptop
Operating system	Raspbian Debian-Buster	Raspbian Debian-Buster	Microsoft Windows 11–64 bit
RAM	1GB LPDDR2 SDRAM	4GB LPDDR4-3200 SDRAM	16GB DDR4 3200 MHz
Memory	2GB	4GB	512GB
CPU	1.4 GHz 64-bit Quad Core ARM Cortex-A53	1.5 GHz 64-bit Quad Core ARM Cortex-A72 (v8)	2.9 GHz–4.2 GHz AMD Ryzen 7 Octa Core 4800H
GPU	300MHz Broadcom BCM2837B0 VideoCore IV	500MHz Broadcom BCM2835 VideoCore VI	144Hz 4GB GDDR6 NVIDIA GeForce RTX 3050
	Device_4	Device_5	Device_6
Device name	HP Desktop	Vivo Smartphone	Samsung Galaxy A7 Tablet
Operating system	Microsoft Windows 11–64 bit	Funtouch 9 (Android 9 Pie)	Android 10
RAM	16GB DDR4 3200 MHz	3GB eMMC 5.1	3GB eMMC 5.1
Memory	250GB	64GB	32GB
CPU	2.5 GHz 11th Gen Intel Core i7-11700	2.0 GHz Octa Core MediaTek MT6762 Helio P22	Octa Core (4 × 2.0 GHz Kryo 260 Gold & 4 × 1.8 GHz Kryo 260 Silver)
GPU	780MHz 2GB GDDR5 AMD Radeon R7 430	650MHz PowerVR GE8320	750MHz Adreno 610