Henrique Fernandes de Oliveira

Object detection and obstacle avoidance for smart walkers using computer vision

Master's dissertation in Electrical and Computer Engineering, specialization in Robotics, Control and Artificial Intelligence, supervised by Professor António Paulo Mendes Breda Dias Coimbra and Professor João Paulo Morais Ferreira and presented to the Faculty of Science and Technology of the University of Coimbra.

April 2024

UNIVERSIDADE D
COIMBRA

Object detection and obstacle avoidance for smart walkers using computer vision

Henrique Oliveira

Supervisor: A. Paulo Coimbra

Co-Supervisor: João Ferreira

Jury:

Prof. Rui Alexandre de Matos Araújo

Prof. António Paulo Mendes Breda Dias Coimbra

Prof. Mahmoud Tavakoli

Dissertation for obtaining the **Master's Degree in Electrical and Computer Engineering**

Coimbra, April 2024

Acknowledgments

Com a entrega desta dissertação concluo aquela que foi uma das fases mais desafiantes e, ao mesmo tempo, gratificantes da minha, ainda, curta vida. São muitas as pessoas a quem um simples "obrigado" não é suficiente por tudo aquilo que por mim fizeram, mas tentarei:

Gostaria de começar por agradecer ao meu orientador, Professor A. Paulo Coimbra, por ter sugerido um tema tão interessante e, juntamente com o co-orientador Professor João Ferreira, terem aceitado orientar a minha tese. Os seus conselhos, apoio e paciência foram vitais para a conclusão da minha dissertação.

Fica também um agradecimentos à Fundação para a Ciência e a Tecnologia (FCT) através do Instituto de Sistemas e Robótica de Coimbra (ISR) pela ajuda financeira a este projeto.

Não poderei deixar de agradecer àqueles que foram os meus grandes impulsionadores e apoiante durante toda a minha vida, especialmente durante as épocas mais difíceis, os meus pais. É também por vocês que termino este curso e nada do que posssa escrever demonstrará o quão agradecido estou por tudo o que por mim fizeram. Quero também deixar um agradecimento especial à minha irmã que me sempre apoiou e pelo companheirismo, especialmente desde o nosso desembarque e à minha restante família.

Agradeço ainda a todos os meus amigos, do Porto Santo, de Barcelos, de Coimbra ou até de Erasmus, que, de uma forma ou de outra, sempre me apoiaram e me incentivaram nas mais variadas loucuras, inclusivé no curso. Podia agradecer a todos especificamente mas, e correndo o risco de me esquecer e não nomear alguém importante, refiro aqueles que mais diretamente viveram comigo a experiência de Coimbra: à Inês Pereira, à Inês Santos, à Mónica Rocha, ao Nuno Mendes, à Tânia Gonçalves, à Marta Carvalho, ao Lourenço Carvalho, à Inês Costa, à Ariana Nunes e a todos os outros que no passado fizeram parte desta época tão bonita. Obrigado por todos os momentos de apoio, suporte, motivação e, ainda mais importante, de diversão. Levo, certamente, estes anos como uns dos mais bonitos da minha vida e aqueles onde mais cresci e mais aproveitei.

Por último, e com o risco de parecer egocêntrico, gostaria de agradecer a mim mesmo por nunca ter desistido e por ter sempre arranjado vontade, mesmo quando era pouca, para terminar o curso. Sei que não é um feito histórico e único mas tenho plena consciência daquilo que lutei e das falhas que tive durante este processo. Aprendi, melhorei e levo-as como um ensinamento para o futuro. 

Obrigado a todos!!! Coimbra certamente ficará na minha memória como um lugar muito especial com pessoas muito especiais e com histórias muito especiais. Voltarei sempre com grande gosto àquela que ficará marcada como a minha cidade universitária, a única que poderia tê-lo sido. "Coimbra tem [mesmo] mais encanto no momento da despedida".

Henrique Oliveira

Resumo

Com a esperança de vida a aumentar, as pessoas com dificuldades motoras necessitam cada vez mais de assistência nas suas atividades diárias. A busca por uma melhor qualidade de vida por parte destas pessoas tem provocado um aumento da procura de auxiliares de marcha, nos últimos anos.

Esta investigação teve como objetivo a incorporação de visão compuatcional e inteligência artificial num andarilho com travões, permitindo identificar e evitar situações de perigo, ajudando as pessoas com problemas de mobilidade a ter uma vida mais segura atenuando as limitações impostas pela sua condição.

Tendo em vista esse objetivo, o andarilho foi equipado com uma câmara Raspberry Pi ligada a uma placa NVIDIA Jetson Nano que corre um modelo de aprendizagem profunda. A câmara permite a monitorização contínua do ambiente em frente do andarilho e a rede neuronal processa esta informação, determinando a decisão que melhor garanta a segurança do utilizador. Com esse propósito, o modelo de aprendizagem profunda foi treinado para produzir uma de quatro opções: Free Way, Left Turn, Right Turn e Full Stop. Para treinar a rede neural, o dataset foi criado por imagens recolhidas pelo DEEC. Cada imagem foi então etiquetada e o dataset foi dividido em três subconjuntos: treino, teste e validação.

Ao integrar a visão computacional e a inteligência artificial, o andarilho tem capacidade de navegar autonomamente e responder a potenciais perigos, como paredes, obstáculos e escadas, ativando os travões, forçando o andarilho a fazer curvas ou, até mesmo, a parar.

Garantir a segurança do utilizador aumenta a sua independência e reduz os constrangimentos impostos pelas suas limitações físicas ou cognitivas, ajudando-o, assim, a ter uma vida mais segura e independente.

Palavras-chave: Andarilho inteligente, Inteligência artificial, Visão Computacional, Classificação de imagens, Pessoas com deficiência, Problemas de mobilidade

Abstract

With life expectancy growing every year, elderly and disabled people are increasingly in need of assistance in their daily life activities due to their mobility issues. The search for a better quality of life for these people has led to an increased demand for walking aids in recent years.

The aim of this research was to integrate computer vision and artificial intelligence into a walker with brakes, making it possible to detect and avoid dangerous situations, helping people with mobility problems to live safer lives by reducing the limitations imposed by their condition.

In this regard, the walker has been equipped with a Raspberry Pi camera connected to an NVIDIA Jetson Nano board running a deep learning model. The camera continuously monitors the environment in front of the walker and the neural network processes this information and outputs the decision that best ensures the user's safety. For that purpose, the deep learning model was trained to output one of four options: Free Way, Left Turn, Right Turn and Full Stop. In order to train the neural network, a dataset was collected around by the DEEC. Each frame was then labelled, and the dataset was divided into three subsets: training, testing, and validation.

By integrating computer vision and artificial intelligence, the walker is able to navigate autonomously and respond to potential hazards such as walls, obstacles and stairs by activating the brakes, forcing the walker to turn or even stop.

Ensuring the user's safety increases their independence and reduces the constraints imposed by their physical or cognitive limitations, thus helping them to lead a safer and more independent life.

Keywords: Smart walker, Artificial Intelligence, Computer Vision, Image classification, Disabled people, Mobility problems

"Humans are not disabled. A person can never be broken. Our built environment, our technologies, are broken and disabled. We, the people, need not accept our limitations, but can transcend disability through technological innovation."

Hugh Herr

Contents

List of Acronyms

AI	Artificial Intelligence
CNN	Convolutional Neural Network
DEEC	Department of Electrical and Computer Engineering
DRL	Deep Reinforcement Learning
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
FN	False Negative
FP	False Positive
FTDI	Future Technology Devices International
IDE	Integrated Development Environment
ML	Machine Learning
ONNX	Open Neural Network Exchange
RAM	Random-Access Memory
RNN	Recurrent Neural Networks
SAE	Society of Automobile Engineers
SDK	Software Development Kit
TF	TensorFlow
TN	True Negative

TP	True Positive
TFLite-micro	TensorFlow Lite micro
UC	University of Coimbra
USB	Universal Serial Bus
WHO	World Health Organization

List of Figures

List of Tables

Listings

1 Introduction

Over the last few years, technology has been evolving exponentially through the most varied fields bringing multiple advantages, ranging from making our daily life more manageable (through the simplest things) to performing high-precision medical surgeries. However, one of the most fulfilling uses of technology is when it helps bring people together, providing support to disabled people to promote equity and minimise the difficulties imposed by their health condition.

Walkers started appearing in the early 1950s, with the first patent granted in 1953 to William Cribbes Robb, for a device called a "walking aid".

Since 1950, walkers have evolved and have now entered the era of intelligent walkers. These have new features and can assist their users in a whole new way, not only people with mobility problems but people with cognitive disorders also.

Although mobility device users represent only a small minority of the population with disabilities, their importance transcends their numbers. Some surveys show that the proportion of people using mobility devices increases sharply with age [13] as shown in figure **1.1**.

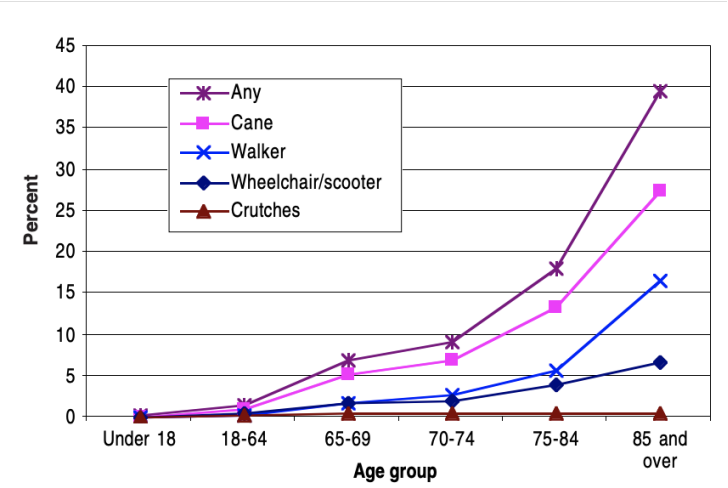


Figure 1.1: Proportion of population using mobility devices, by age and device [1].

1.1 Motivation

A rapidly ageing population brings with it a greater need for mobility aids such as walkers. However, regular walkers cannot help with all the specific problems of the entire population, which makes smart walkers a better and more viable option, as they can provide personalised and context-aware support, thereby improving the quality of life of users. Smart walkers, for example, can better assist people with cognitive or visual impairments to walk more independently by ensuring their safety by avoiding collisions or dangerous situations, such as walls, obstacles or stairs.

Despite the progress made in recent years, there is still a long way to go to improve the functionality, usability and safety of smart walkers.

Smart walkers require the integration of different disciplines such as engineering, computer science, human-computer interaction and healthcare. Writing a dissertation on this topic can be an opportunity to explore and apply knowledge from different fields.

This dissertation builds on the work of Pierdevară [14] by focusing specifically on the design and development of smart walkers, with the aim of addressing key limitations and advancing the state of the art in this field. For that purpose, an ESP32-CAM board was only used to acquire the dataset, and a Raspberry Pi camera connected to an NVIDIA Jetson Nano board was installed on the walker and programmed to monitor the front view while running a deep learning model that, based on the view ahead, outputs the

appropriate command.

With the contribution of this dissertation, smart walkers can be a meaningful and rewarding experience, with the potential to make a real-world impact and advance the state of the art in this important field.

1.2 Goals

This dissertation aims to investigate the design, development and evaluation of a smart walker with improved usability and safety features. The objective is to reduce potential risks and ensure user safety by implementing measures to prevent dangerous situations.

To achieve these goals, the following key objectives have been determined:

- Equip the walker with vision by installing a camera on the front to capture streaming images.
- Develop a deep learning model capable of classifying the images captured by the walker's camera and determining the appropriate action to avoid obstacles.
- Integrate the deep learning model into a board capable of real-time actions, providing the walker with the intelligence to decide between four outputs (Free Way, Left Turn, Right Turn and Full Stop)
- If no objects potentially endangering the walker and its user's safety are detected along its path, the deep learning model should not impose any restrictions.
- Objects located at the bottom of the frame are considered safety hazards. Appropriate action should be taken to address them. An object on the bottom left will cause the walker to navigate to the right, while an object on the bottom right will cause the walker to navigate to the left.
- In case the walker is unable to safely navigate around the obstacle, the model will classify the situation accordingly, prompting the system to bring the walker to a complete stop to ensure the user's safety.

1.3 Dissertation structure

This dissertation is organised into five main chapters.

1^{st} chapter - Introduction:

Introduces the research, addressing overall issues, justifying the study's necessity, and outlining general and specific objectives.

2^{nd} chapter - State of art:

The current bibliography of autonomous walking assistance, from conventional to smart models, is explored. The application of deep learning in autonomous driving is studied. Final considerations identify gaps in the existing literature.

3^{rd} chapter - Methods and tools:

Details crucial methodologies and tools about the used hardware (ESP32-CAM's and NVIDIA Jetson Nano) and the artificial intelligence model.

4^{th} chapter - Developed Work:

Covers key project aspects, detailing the assembly process, the dataset acquisition, methodology for model training, testing, and results. Deployment steps are outlined, concluding with an in-depth analysis of model effectiveness in practical scenarios.

5^{th} chapter - Conclusion:

A comparison is made between the goals and the results achieved, including a discussion of any difficulties encountered and their solutions. Additionally, some suggestions for future work are provided.

2 State of Art

This chapter analyses the existing bibliography on walkers and their variations, from standard walkers to smart walkers, highlighting their importance in the lives of people with limited mobility. It also discusses the ongoing evolution of technology aimed at improving the comfort and quality of life for these individuals.

An investigation into autonomous driving was conducted, demonstrating the evolution of the technology in relation to driving. It is shown that the technology can reduce road accidents by assisting the driver. A classification of levels of autonomous driving is presented, ranging from level 0 to level 5, each with its own set of characteristics. Level 0 represents the driver having full control of the vehicle, while level 5 represents the vehicle performing all driving tasks under all conditions. The study also explores deep learning in driving and the common technologies used in it.

At the end of the chapter, gaps in the bibliography that this dissertation aims to cover are presented.

2.1 Walkers

From year to year, the ageing of the population has been increasing sharply, whereby is estimated a growth of 1 in 5 by 2050 [15]. As a consequence, walkers have evolved to try to meet the needs of their users. There are various types of this device, each of them with small differences but all with the same basic design and goal, to help and support people with mobility difficulties.

The basic design of a walker always meets the following requirements: it takes an incomplete cuboid shape, medically known as *Zimmer frame*, with a height around the waist and length slightly wider than the user. The objective of this famous shape is to

transfer 64% of the user's weight into the arms, relieving some pressure from the lower body. This design has the advantage of having more stability and a greater sense of security on the part of the user and those around him [15] raising the level of activity and Independence.

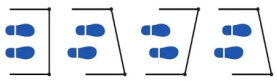
2.1.1 Standard and articulated Walkers

A standard walker (**fig: 2.1a**), often referred to as a basic walker, is a traditional mobility aid designed to offer stability and support while walking. It features a simple framework with four legs and two handles, providing the user with a reliable way to maintain their balance. These walkers are commonly used by people with limited mobility or those recovering from injuries, offering assistance during ambulation, especially indoors and on even surfaces.

An articulated walker (**fig: 2.1b**) is a device that has the same features as the standard one plus the capability of articulation, i.e., to move each pair of legs on each side separately from the other two like shown in figure **2.1c**. It provides continuous support when keeping up with the leg motion. It is more adequate for people with balance problems allowing the user to still have a natural walk.

(a) Standard walker [16].

(b)



(c)

(d) Articulated walker [17].

Figure 2.1: Fixed walkers.

2.1.2 Walkers with wheels

A walker with wheels builds upon the concept of a standard walker by integrating front wheels. This feature enhances manoeuvrability, allowing users to move more freely while

still receiving the support of the walker. The swivelling front wheels and lockable rear legs create a balanced combination of stability and ease of movement. These walkers are ideal for individuals seeking enhanced mobility and the ability to navigate various environments with reduced effort.

There are multiple wheeled walkers with different number of wheels:

(a) 2 wheels walker [18].

(b) 3 wheels walker [19].

(c) 4 wheels walker [20].

Figure 2.2: Wheel walkers.

Each of these walkers is more appropriate for different mobility problems and different environments:

A 2 wheels walker (**fig: 2.2a**) still offers the stabilisation of a standard walker without the need of bearing the walker. It can help people avoid the tiredness of dragging the device while still assuring support, for that, it is suitable for people with less strength in their upper body [15].

The 3 wheels (**fig: 2.2b**) walker assures continuous balance support while being lighter and easier to move around than a walker with more wheels.

When talking about the 4 wheels walker (**fig: 2.2c**), still provides continuous support without the need of lifting the device. It is more appropriate for people with unsteady feet but that do not need to support all their weight on the walker [15]. People with balance or cognitive problems are not suitable for this kind of walker. We can often see a seat option with this walker.

2.1.3 Smart Walker

In the last few years, intelligent walkers have been a big bet for improvement as a walking aid. *Valadao et al.* define smart walkers as "walkers that rely on electronics and

control systems, besides the mechanical structure" [21]. These more advanced walkers are equipped with multiple sensors that enable them to gather real-time data about their surroundings, user movements, and environmental conditions that help the device make decisions in order to preserve the user's safety and improve mobility.

According to Martins *et al.* [22], the development of a walker has to take into account that every disability is different so its design should be specific and attend to each user's needs. The evolution of technology allows us to integrate a range of features on the walkers empowering them with more security and making their' manoeuvrability easier and more comfortable.

Martins *et al.* [22] propose that smart walkers should be designed to provide assistance to their users at various levels, such as:

- **Physical support:** Like any other type of walker, providing this type of support to the user is essential, such as balance. This support can be separated into two types, passive and active:

  -Passive: through mechanical and/or structural enhancement, the device can provide the necessary balance and support that the user needs. A UPWalker (**fig: 2.3**) is a great example of a walker with ergonomic handles and a sturdy frame that encourages the user to stand up providing support while promoting a better posture.

Figure 2.3: UPWalker [2].

-Active: with the help of the sensors installed on the walker, data on the surroundings is recorded and, allows the system to provide external and controlled help to the user making the march easier and avoiding dangerous situations. One example of active physical support is when the walker encounters a declined path and activates its brakes avoiding it to slip or when the walker applies energy to its wheels to help the user in the presence of an inclined path. The *Camino* Walker [3] (**fig: 2.4**) is a good example of an active support walker. It looks like a traditional walker but can power or brake it's wheels when there is need of support, giving more confidence to the user.

Figure 2.4: Camino Walker [3].

- **Sensory assistance:** Since the walker is used by people with mobility limitations, sudden and unexpected changes can be problematic. An early detection of obstacles is essential while the control system warns the user by sounds and vibrations alerts or by operating directly on the device's actuators (for example, by changing momentarily the path of the walker). This kind of assistance is more suitable for users with visual problems or to help in environments with multiple obstacles. Annicchiarico *et al.* [4] (**fig: 2.5**) presented a new smart walker called *i-Walker*. This device is endowed with sensory assistance when communicating with the user. The walker has a submissive behaviour, i.e., when it evaluates the surroundings and detects some obstacle, it attempts to infer a safer path for the user. Monitoring is done to evaluate if the user resists or accepts the system's suggestion. If he resists, a new attempt is done until the user agrees with the motion.

Figure 2.5: i-Walker [4].

- **Cognitive assistance:** Walkers can also be endowed with cognitive features, such as location. This type of walker is normally used by users with cognitive problems related to memory and orientation. Some of these walkers are programmed to guide people on a pre-defined path. It is also common to have the device communicating with the user through voice commands, a visual interface or by receiving directions and commands from the person.

  DALi project (**fig: 2.6**) [5] aimed to create a device that could give the needed cognitive support to people with limitations, such as memory and orientation loss. The product of this project was called *Cognitive Walker* (C-Walker) and is capable of guiding the user through a pre-determined route, detecting anomalies along the way, observing and predicting human paths on the surroundings, adapting the course in order to avoid obstacles and other intelligent features that keep the human in safety.

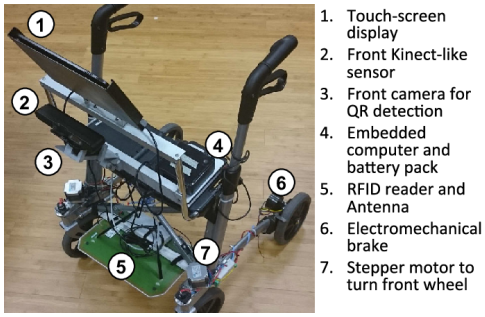


Figure 2.6: C-Walker [5].

- **Health monitoring:** Since we are dealing with people with fragile health, monitoring their health while giving updates to their caretaker, might be very useful.

This information is used to keep a medical history of the patient and used later to get better treatment for his limitations.

The *Camino* walker (**fig: 2.4**) [3] can also provide monitoring of health to its user. Using artificial intelligence, it tracks the walk with 22 different metrics, helping keep track of the user's march progress.

- **Advanced human-machine interface:** A simple and functional interface is the final key when designing a smart walker. Is through this element that the machine and the user/caretaker will communicate, for example, the walker will give warnings and updates on the surrounding information and the person will inform the machine about the path to follow. Since a big target of the user of the walker are the elder people, a clean and not too complicated interface is essential, for it to be simple to learn and to avoid frustration when using it.

  The C-Walker (**fig: 2.6**) [5] is also equipped with a rich set of interfaces that are used to communicate with the user to suggest new paths or even give warnings, using visual, acoustic or haptic signals. Active interfaces are also used, such as electromechanical brakes and motorized wheels.

Smart walkers can then provide a range of assistance, from enhancing physical support to incorporating sensory feedback, aiding cognitive functions, monitoring health and offering advanced user interfaces. These devices are suitable to help people with various limitations, such as mobility, cognitive or visual.

2.2 Autonomous driving & Deep learning

In recent years, autonomous driving has gained more and more enthusiasts and fame, which has boosted its growth. Renowned car manufacturers such as Audi, BMW, Ford, Google, General Motors, Tesla, Volkswagen and Volvo have all been engaged in intensive research to improve safety through autonomous vehicle technology. These systems employ a sophisticated combination of sensors, cameras, and radar to gather and analyse data from the vehicle's surroundings. Subsequently, they process this information to generate a decision.

2.2.1 Autonomous driving

Data from the World Health Organization (WHO), published in the *Global status report on road safety 2023* [23], shows that although the number of road traffic deaths has fallen, it is still a cause for concern, killing 1.19 million annually, making it the 12th more common cause of death in general and the leading cause of death in the 5-29 age group. The main cause of these accidents is undeniably human error, including speeding, drunk driving, and driver distraction. This underlines the importance of integrating new vehicle technologies such as rear-view cameras, adaptive headlights, and forward collision warning systems. These advances play a crucial role in reducing accidents by assisting the driver and helping to prevent potential collisions.

The major goal of autonomous driving is to prioritise user safety, a task that becomes increasingly complex given the vast amount of data these systems must handle. Basic autonomous driving systems include a number of essential technologies, including sensing, localisation, perception, decision-making, seamless interaction and data storage. These integrated components work together to enhance safety and drive the evolution of autonomous driving capabilities.

According to the Society of Automobile Engineers (SAE), automation should be classified into 6 levels, from 0 to 5, being 0 no driving automation and 5 full driving automation [24].

The following image resumes the classification of each level of automation:

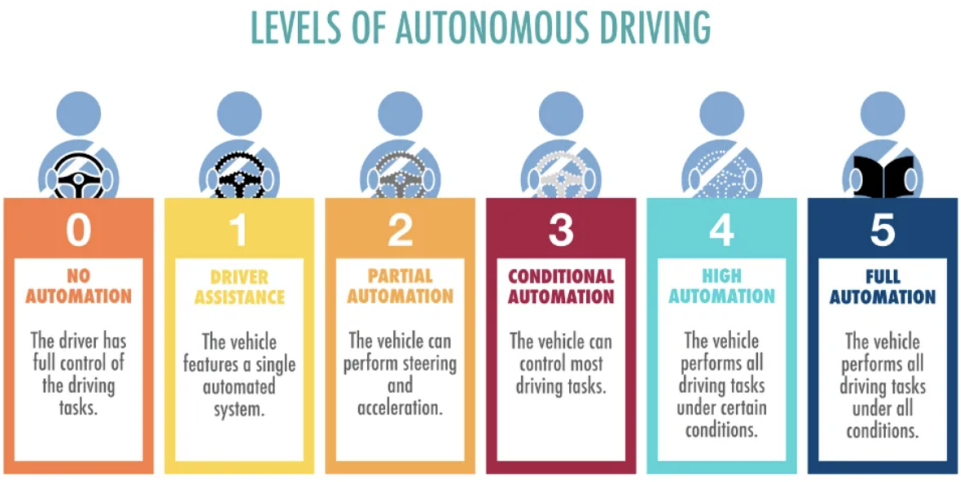


Figure 2.7: Levels of autonomous driving according to SAE[6].

- *SAE Level 0:* **No automation** - The human driver maintains full control over all

elements of the dynamic driving task, even when aided by warning or intervention systems.

- *SAE Level 1:* **Driver assistance** - A driver assistance system operates in specific driving modes, taking charge of either steering or acceleration/deceleration based on the driving environment, while expecting the human driver to handle all other aspects of the dynamic driving task.

- *SAE Level 2:* **Partial automation** - In certain driving modes, one or more driver assistance systems manage both steering and acceleration/deceleration based on environmental data, with the expectation that the human driver will manage all other aspects of the dynamic driving task.

- *SAE Level 3:* **Conditional automation** - An automated driving system performs all elements of the dynamic driving task in specific driving modes, anticipating that the human driver will intervene as needed.

- *SAE Level 4:* **High automation** - In designated driving modes, an automated driving system assumes full responsibility for all aspects of the dynamic driving task, regardless of the human driver's response to intervention requests.

- *SAE Level 5:* **Full automation** - Under any roadway or environmental condition manageable by a human driver, an automated driving system consistently executes all aspects of the dynamic driving task.

2.2.2 Deep learning

Deep Learning is the core of autonomous vehicles making it a decision-maker system. These systems gather data from different on-board sources, such as cameras, radars, LiDARs, ultrasonic sensors, GPS units and/or inertial sensors and process them making a driving decision on low latency.

Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Deep Reinforcement Learning (DRL) are the most common deep learning networks used in autonomous driving. This work will focus on *Convolutional Neural Networks*.

Convolutional Neural Networks, also known as CNN or ConvNet, are specialised neural networks primarily designed for processing spatial data, particularly images. Their

distinguishing feature is the convolution operation, a mathematical process that blends two functions to generate a third function that represents how the shape of one function is changed by the other. CNNs operate by taking an input image and transforming it into a simplified format that is easier to process while retaining crucial features. These networks have the ability to autonomously learn and extract complex features from the dataset they are trained on, effectively encoding the feature space.

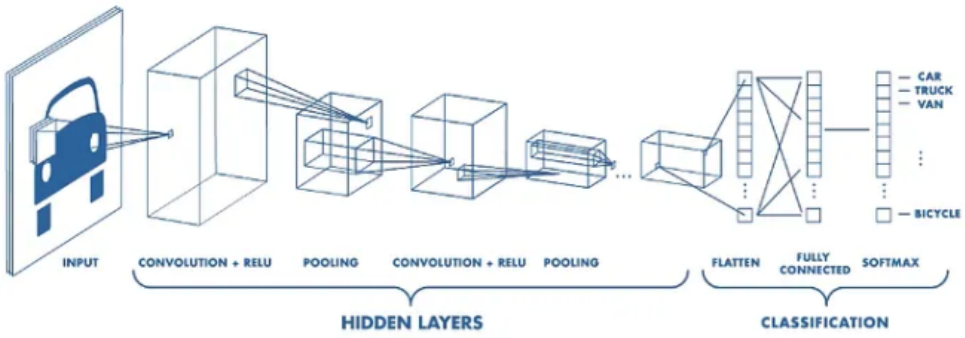


Figure 2.8: Architecture of a CNN [7].

Figure **2.8** shows the basic architecture of a CNN. The model receives an image as input, with the final layer being the decision layer, a softmax layer responsible for computing the probabilities associated with all potential outputs and selecting the one with the highest probability value.

The key point for deep learning systems is the amount of data available for training. Tesla and Waymo, two of the biggest companies leading the development of autonomous driving, have different approaches concerning the collection of data. Tesla uses its extensive user base to gather real-world data on vehicle performance and potential improvements. Conversely, Waymo employs robust computer simulations to refine its systems and applies what it learns from these simulations to a smaller real-world fleet. In recent years, prompted by growing interest in autonomous vehicles, many driving datasets have been publicly released and documented. This accessibility allows independent researchers to effectively train their models.

Deep learning-based perception, particularly using convolutional neural networks (CNNs), has become the dominant standard for object detection and recognition, with significant successes in competitions such as the ImageNet Large Scale Visual Recognition Challenge. The advances in deep learning for object detection and recognition represent a remarkable trajectory that will shape several fields in the coming years. Deep learning is expected to

play a key role in revolutionising industries such as autonomous vehicles.

2.3 Final Considerations

The state-of-the-art review has provided insights into the evolution of walkers, ranging from standard designs to advanced smart models. As the global population ages, the demand for innovative mobility aids continues to grow, emphasising the need for solutions that prioritise user safety, autonomy, and functionality.

While the existing literature has extensively explored various aspects of smart walkers, including stability, manoeuvrability, and user interface design, there are notable gaps that warrant further attention. One significant gap pertains to the limited focus on collision avoidance capabilities in smart walkers. While features such as stability and user interface have received considerable attention, the integration of advanced collision avoidance technologies, such as cameras and deep learning algorithms, remains relatively underexplored.

Furthermore, comprehensive studies that consider the user experience and usability of collision avoidance systems in smart walkers are lacking. It is crucial to understand user preferences, feedback, and interactions with such systems to ensure their effectiveness and acceptance in real-world settings.

By addressing these gaps in the literature, future research effords can contribute significantly to advancing the field of assistive technology and improve the safety, autonomy, and usability of smart walkers for individuals with mobility limitations.

3 Methods and tools

This chapter covers the methods and tools used to develop and implement the smart walker. Specifications about the hardware and details about their use, as well as information on the software, including the machine learning model and framework choice and used techniques are provided.

3.1 Hardware

In this dissertation, the ESP32-CAM from Espressif was used as the primary device for dataset acquisition. Subsequently, for real-world implementation, a Raspberry Pi camera was connected to the Jetson Nano from NVIDIA. The hardware platforms were chosen for their ability to capture and process visual data in controlled and dynamic environments.

3.1.1 ESP32-CAM

The ESP32-CAM board is widely used in a variety of applications, such as home automation, surveillance, robotics, and IoT projects. Its main advantages include its low cost, compact size, and versatility.

In terms of camera performance, the ESP32-CAM features an OV2640 camera module with a resolution of up to 2 megapixels. It can capture images and video at various resolutions and frame rates and supports features such as automatic gain control, white balance, and exposure.

Although, its small size brings some inconveniences, such as the fact that, this board, does not have a built-in programmer. In fact, to program this microcontroller, the use of an Future Technology Devices International (FTDI) programmer is needed to connect

it to the computer and upload code. The ESP32-CAM can be programmed using the Arduino Integrated Development Environment (IDE) or other programming languages such as MicroPython. It also has a range of libraries and examples available, making it easy to get started.

PinOut

ESP32-CAM board comes with 3 ground (**GND**) pins, 2 power pins: **5V** and **3.3V**, and 10 **General Purpose Input/Output (GPIO)** pins, as we can see on the PinOut scheme in figure **3.1**. GPIO's are uncommitted digital signal pins which may be used as inputs or outputs, or both. In other words, GPIO pins are found on many microcontrollers, including the ESP32-CAM, capable of interfacing a wide range of products, such as sensors, displays, and other microcontrollers. These are powerful and flexible pins that are totally programmable to perform a variety of functions, depending on the needs of the application. Although, their flexibility brings some limitations, such as their maximum voltage and current ratings, and their ability to handle high-frequency signals.

A 4th power pin coloured in yellow can be noticed, **3.3V/5V**. This is a power-out pin, intended to power any device with 3.3 or 5V.

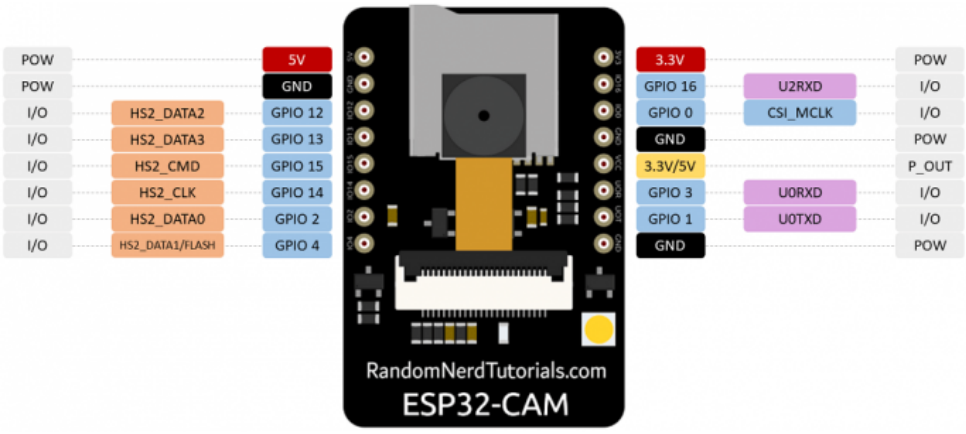


Figure 3.1: ESP32-CAM PinOut [8].

GPIO 1 and **GPIO 3** are the serial pins (TX and RX, respectively). These are the pins used to communicate with the board and upload the code. We can also use them to connect some other output or sensor, but we will not be able to open the Serial Monitor.

Connecting the **GPIO 0** pin to **GND** puts the ESP32-CAM in programming mode. To run the code we just need to disconnect it and press the RESET button.

ESP32-CAM has a couple of integrated LED that we can use to test the connection. Next to the RESET button, there is a red LED that is connected internally to the pin **GPIO 33**. There is also a very bright integrated LED on the front of the board that can work as a flashlight. This LED is internally connected to pin **GPIO 4**.

Overall, the ESP32-CAM is a powerful and affordable microcontroller board that offers a wide range of features and applications. Its popularity and active community support make it a great choice for developers and hobbyists alike.

Integrated Development Environment

There are multiple options to program the ESP32-CAM. Since this microcontroller is Arduino programmable, it is possible to use the Arduino IDE to communicate with it, although there is also the option to use Platform IO, a cross-platform, cross-architecture, multi-framework professional IDE tool for embedded system and software engineers who write embedded applications, the installation of the corresponding extension for *Visual Studio Code* is a viable option to communicate with the board.

Despite the countless available options, the Arduino IDE has been selected for use in this dissertation due to its extensive collection of examples, robust support, and familiarity with it.

Programming

As explained earlier, this particular ESP32 CAM module (OV2640) does not have a built-in Universal Serial Bus (USB) programmer. Connection and power are facilitated by the use of an FTDI programmer. The corresponding schematic diagram of the required connections is shown in figure **3.2**.

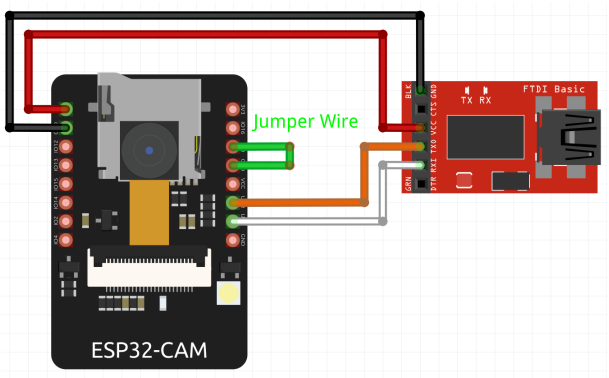


Figure 3.2: ESP32-CAM connections schematic [9].

Table 3.1: ESP32-CAM connections

ESP32-CAM	FTDI
GND	GND
5V	VCC (5V)
U0R	TX
U0T	RX
GPIO 0	GND

It is important to notice that, in order to have the microcontroller in programming mode, the pin **GPIO 0** and **GND** need to be connected with a jumper. As soon as the code is uploaded to the board, we disconnect this pin and press the RESET button to run the code the new code.

3.1.2 NVIDIA Jetson Nano

Jetson Nano Developer kit-B01 is a small, powerful computer developed by NVIDIA specifically designed to elevate low-budget Artificial Intelligence (AI) projects [25]. This computer is powered by a small Graphics Processing Unit (GPU) capable of running multiple neural networks in parallel for applications like image classification, object detection, segmentation and speech processing which makes it an ideal hardware for this dissertation.

Specifications

The main technical specifications of the Jetson Nano Developer kit-B01 are described in the following table:

Table 3.2: Jetson Nano specifications

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Connectivity	Gigabit Ethernet, M.2 Key E

Getting started

The Jetson Nano Developer kit-B01 has multiple ports for the most variable uses, as shown in figure 3.3.

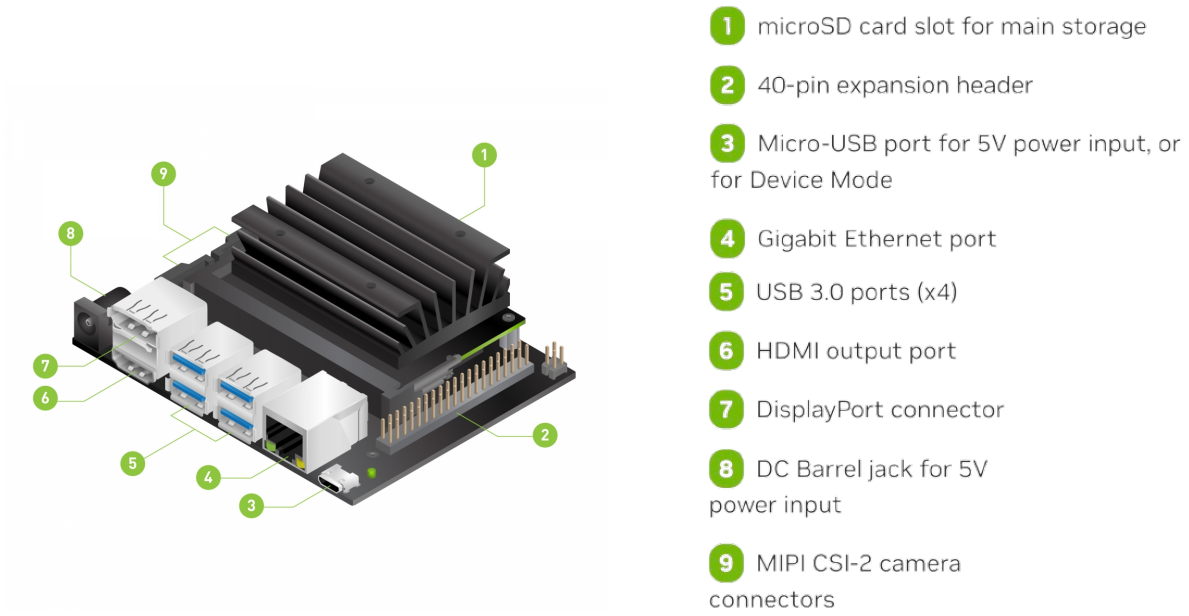


Figure 3.3: Jetson Nano connections [10].

To get started with the Jetson Nano, it is needed to power it with 5V ⎓ 2A supply, connect a monitor through the HDMI port, a keyboard and a mouse using the USB ports and the ethernet cable.

This single-board computer uses the system image on the microSD card as a boot device, the minimum storage recommended for the card it's 32 GB. Using another computer with an internet connection, the image of the system should be downloaded from the NVIDIA official page [26] and written to the SD card.

Despite being highly capable, the Jetson Nano remains an embedded device, which means it is likely to be slower than a laptop. Training full deep learning models on it is usually not advisable as it will take a long time. It is advisable to train the model with data on a desktop or a powerful internet server optimized for that task, as was done in this work, and then transfer the model to the Jetson Nano for prediction purposes.

Raspberry Pi Camera 2

The Raspberry Pi Camera 2, featuring 8MP resolution and available in both visible light and infrared versions, seamlessly integrates with the Jetson Nano. This streamlined compatibility allows for the effortless capture of photos or videos to support models running on this robust platform.

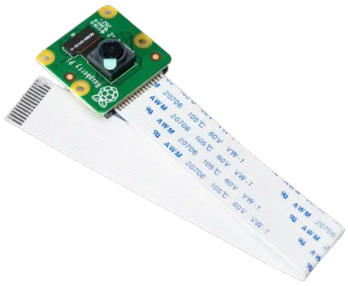


Figure 3.4: Raspberry Pi Camera Module 2 [11].

The Jetson Nano supports capturing video feeds and static images via a variety of interfaces. The Raspberry Pi Camera camera has to be attached to the pins represented by the number 9 on the figure 3.3. The camera stream can be accessed in real-time by connecting the camera to the Jetson Nano and passing the camera identifier. For instance, in the case of the Raspberry Pi Camera, the identifier is **csi://0**. A table containing the identifiers of compatible cameras can be found on NVIDIA's official GitHub repository for Camera Streaming and Multimedia [27].

3.2 Artificial Intelligence Model

The core of this dissertation is the machine learning model trained specifically for this project. The following subsections cover the tools (IDE and framework), the chosen

architectures and the technique used during training. A description of each output and the metrics used to evaluate the performance of the model are also explained.

3.2.1 Google Collab

Google Colab [28] (short for "Colaboratory") from *Google Research* is a hosted Jupyter Notebook service that requires no setup to use. Colab allows anyone with a Google account to write and run Python code through the browser and is particularly well suited to machine learning.

The use of Google Colab is appropriate for this work, as it provides free access to computing resources, including GPUs. A code to train CNNs that would take several minutes or even hours to run on our computer can easily be run in the browser using the resources of Google's GPUs.

3.2.2 Convolutional Neural Network

Various CNN architectures serve different purposes. This dissertation employs the fundamental architecture of AlexNet [12].

Alexnet was created by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor at the University of Toronto. This neural network is similar to LeNet, one of the first successful CNNs.

AlexNet became famous after winning the ImageNet Large Scale Visual Recognition Challenge (ILSRC) in 2012, defeating its opponents by speed and efficiency. It consists of eight layers (five convolutional and three fully connected):

1. **Convolution:** kernel 11x11, stride 4
2. **MaxPool:** kernel 3x3, stride 2
3. **Convolution:** kernel 5x5, pad 2
4. **MaxPool:** kernel 3x3, stride 2
5. **Convolution:** kernel 3x3, pad 1

6. **Convolution:** kernel 3x3, pad 1

7. **Convolution:** kernel 3x3, pad 1

8. **MaxPool:** kernel 3x3, stride 2

The structure of AlexNet is represented in the figure **3.5**, presented in the original paper on the network created by Alex Krizhevsky.

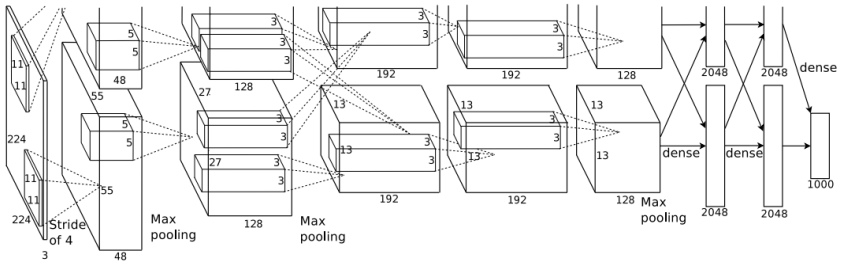


Figure 3.5: Architecture of AlexNet [12].

3.2.3 Transfer Learning

Traditionally, training a deep learning model from scratch demands a substantial amount of labelled data, which might be impractical or unavailable in certain scenarios. A technique called **transfer learning** addresses this limitation by leveraging knowledge gained from another pre-trained model on large datasets and applying it to a specific task with a smaller dataset.

The fundamental idea behind transfer learning is to exploit the features learned by a model on a source task and adapt or transfer this knowledge to a target task. This process is especially beneficial when the target task shares some underlying patterns or features with the source task. By using a pre-trained model as a starting point, the model can kickstart its learning process with a foundation of generalized knowledge. This can lead to quicker convergence and improved performance, even when dealing with limited data.

Transfer learning comes in various forms, such as fine-tuning, where a pre-trained model is adjusted for the target task, and feature extraction, where the knowledge from the pre-trained model is used to extract relevant features for the new task. The versatility of transfer learning makes it a valuable tool in scenarios where data scarcity is a challenge,

allowing practitioners to make the most of available resources and still achieve robust and effective models.

In this particular instance, transfer learning is being employed by utilizing a pre-trained Alexnet model that has been initially trained on the extensive ImageNet dataset. The objective is to re-train this model with the dataset specifically collected for the new task. To align the model with the new task's requirements, a crucial adjustment involves modifying the number of output classes. In the original Alexnet, designed for ImageNet's 1000 classes, this adjustment entails customizing the model to output the requisite 4 classes relevant to this specific application. This tailored fine-tuning ensures that the model adapts its learned features to the nuances of the new dataset, enhancing its capability to accurately classify and address the distinctive characteristics of this targeted task.

The neural network architecture after changing the number of outputs from the original 1000 classes to 4 is shown in appendix **B**, created with Netron [29].

3.2.4 Output

For the model to enable the smart walker to make the decisions of continuing to move, stopping one side, or applying brakes on both sides, a multi-class classification output is needed. The output will induce an action on the brakes, depending on the scenario it encounters, avoiding the collision by turning left or right or even by braking. The relation between the decision made by the model and the action applied on the brakes is described in the following table:

Table 3.3: Smart Walker Action Classification

Class	Output	Meaning	Action
0	Free Way	No obstacle detected	No action
1	Left Turn	Obstacle detected on the right	left brake activated
2	Right Turn	Obstacle detected on the left	right brake activated
3	Full Stop	Dangerous or no escape situations	wheels lock

Section **4.1.3** provides a more comprehensive explanation of each class and the labelling process.

3.2.5 Machine Learning Framework

Machine Learning Frameworks are interfaces that make more agile the process of creating, training, testing and deploying a model. Multiple frameworks help create Machine Learning models, such as PyTorch [30] and TensorFlow [31]. The choice of which framework to use is mainly personal, although several key points should be taken into consideration:

- **Personal Case:** What problems are addressed? What type of data is being used? What kind of software/hardware is used? These are questions that may help when choosing the framework to work with.
- **Language of Programming:** The programming language should be taken into consideration also. Different frameworks may have different supports for each programming language. Choosing a framework that is compatible with the language in use is crucial.
- **Personal Preference:** Lastly, the usage preference should be taken into consideration at the time of choosing the framework to work with. The familiarity with it, and the experience of the user may help make a final decision.

For familiarisation purposes, this work used PyTorch, a widely acclaimed deep learning framework developed by MetaAI. PyTorch offers dynamic computation graphs, easy debugging, and extensive support for GPU acceleration.

ONNX, which stands for Open Neural Network Exchange, is a standard format used to encode deep learning models. It functions as a universal language for AI models, allowing for easy transfer and execution across various frameworks and hardware platforms. This compatibility is particularly useful for devices like the Jetson Nano, a compact computer designed for running AI applications at the edge.

ONNX enables easy integration of AI models into Jetson Nano projects. Various libraries and tools are available to facilitate the deployment of ONNX models.

3.2.6 Metrics

In order to assess how well the ML model is predicting, various metrics are calculated and different conclusions can be drawn from these to help make changes to the training

to achieve better results.

Each inference of a classification model can be categorised according to 4 concepts:

True Positive (TP): Instance correctly predicted as part of the class;

True Negative (TN): Instance correctly predicted as not part of the class;

False Positive (FP): Instance wrongly predicted as part of the class;

False Negative (FN): Instance wrongly predicted as not part of the class;

Although a confusion matrix is not considered a performance metric, it can give a lot of information about the performance of the model and is very useful to see where the model tends to confuse itself. It shows a relation between label and prediction, with the rows representing the ground truths and the columns the predictions. The correct predictions are on the main diagonal and the incorrect predictions are in all the other cells.

Figure **3.6** is a general example of a confusion matrix.

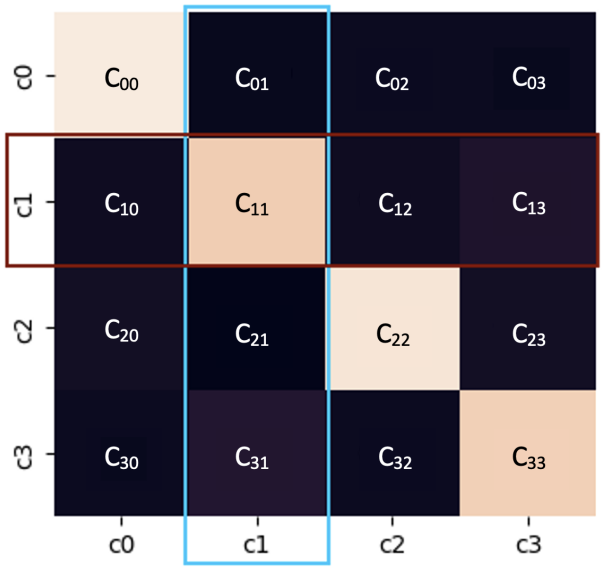


Figure 3.6: Example of a confusion matrix.

Given class 1, the row, marked with a red rectangle, represents all the predictions for inputs with label 1. The blue column represents all the times the Machine Learning (ML) model predicted an inference as 1, regardless of the ground truth.

When studying the confusion matrix, taking the values of TP's is as simple as looking at the value on the cell of the desired class in the main diagonal. In the case of the class 1:

$$TP_1 = C_{11} \tag{3.1}$$

In a multi-class problem like this, reading the values of TN, FP and FN is not as straightforward as in a binary problem. The value of the FP's is the sum of all the values on the column of the desired class, except the main diagonal:

$$FP_1 = C_{01} + C_{21} + C_{31} \tag{3.2}$$

The value of the FN's is the sum of all the values on the row of the desired class, except the main diagonal:

$$FN_1 = C_{10} + C_{12} + C_{13} \tag{3.3}$$

The value of the TN's is the sum of all the values not in the column or the row of the desired class:

$$TN_1 = C_{00} + C_{02} + C_{03} + C_{20} + C_{22} + C_{23} + C_{30} + C_{32} + C_{33} \tag{3.4}$$

The confusion matrix is a good place to start calculating the metric, as all the predictions are well represented visually.

The major metrics used to evaluate the model are Accuracy, Precision, Recall, and F1-score.

Accuracy:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.5}$$

Accuracy is the most common and simplest metric for evaluating a model. It shows the percentage of correct instances out of the total number of predictions, i.e. the ratio of correct classifications to the total number of predictions.

Having a low accuracy may indicate unbalanced or low representative data. Expanding the dataset may be a good way to improve this metric.

Precision:

$$Precision = \frac{TP}{TP + FP} \tag{3.6}$$

Precision is a metric that shows, from all the inferences predicted as part of the class in focus, how many are correct. A low precision score indicates a high number of false positives, which may result from an imbalanced class or untuned model hyperparameters.

Recall:

$$Recall = \frac{TP}{TP + FN} \tag{3.7}$$

Recall is similar to precision. It measures, out of all instances that truly belong to the class in focus, how many were correctly predicted as such. Having a low recall means the model failed a lot in identifying instances as part of the class, i.e. high value of FN.

F1 Score:

Finally, the F1 score is the harmonic mean between precision and recall.

$$F_1 = \frac{2 * precison * recall}{precison + recall} \tag{3.8}$$

Often F1 score is used instead of accuracy as it is not affected by unbalanced data and gives more realistic results about the accuracy of the models to get correct predictions for each class.

A low F1 score indicates that the model is struggling to balance both precision and recall. It could be due to either a high number of false positives (resulting in low precision) or a high number of false negatives (resulting in low recall), or a combination of both. This suggests that the model's performance is not ideal, as it fails to accurately balance both precision and recall.

4 Developed work

This chapter covers the implementation of the smart walker. It explains the process of obtaining the custom dataset, including the camera assembly, the code developed to take and save the photo using the ESP32-CAM, and the labelling process.

It presents the system architecture and explains the image pre-processing procedure, as well as the training and testing of the model on the computer. Concrete results and metrics are studied and discussed.

The chapter describes the unsuccessful attempt made with the ESP32-CAM microcontroller and the solution found using the NVIDIA Jetson Nano. It includes testing the model on the Jetson Nano while streaming the camera image.

The figure **4.1** is a schematic of the entire system. The Raspberry PI camera will take a photo and input it to the model on the NVIDIA Jetson Nano. The model will perform an inference and output the class with the biggest probability as described in section **3.2.4**. However, it's important to note that the connection to the brakes was not addressed in this dissertation, this aspect is expected to be addressed in future work.

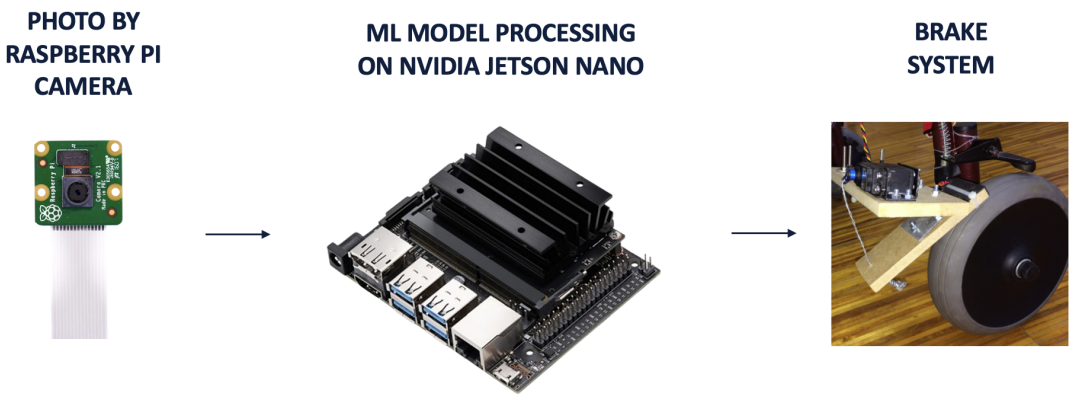


Figure 4.1: System main components.

4.1 Dataset acquisition

There is an undeniable lack of a dataset for autonomous driving indoors, for that reason, there was the need to create a specific dataset.

To collect the dataset, an ESP32-CAM was attached to the walker to capture images and store them on an SD card. These images were then labelled and categorised based on different scenarios. Once the complete dataset was compiled, it was divided into training, test and validation sets. Before being used to train the model, the images were pre-processed.

4.1.1 Camera assembly

To begin the data acquisition process, the ESP32-CAM was mounted on the walker's basket at a 30° angle and positioned 52.5 centimetres above the ground. In this configuration, the camera obtained a field of view that covered the ground from 40 centimetres in front of the walker up to 1.4 metres, providing a visibility range of one metre, as shown in the figure **4.2**. The smaller base of the trapezoid formed by the field of view is 60 cm.

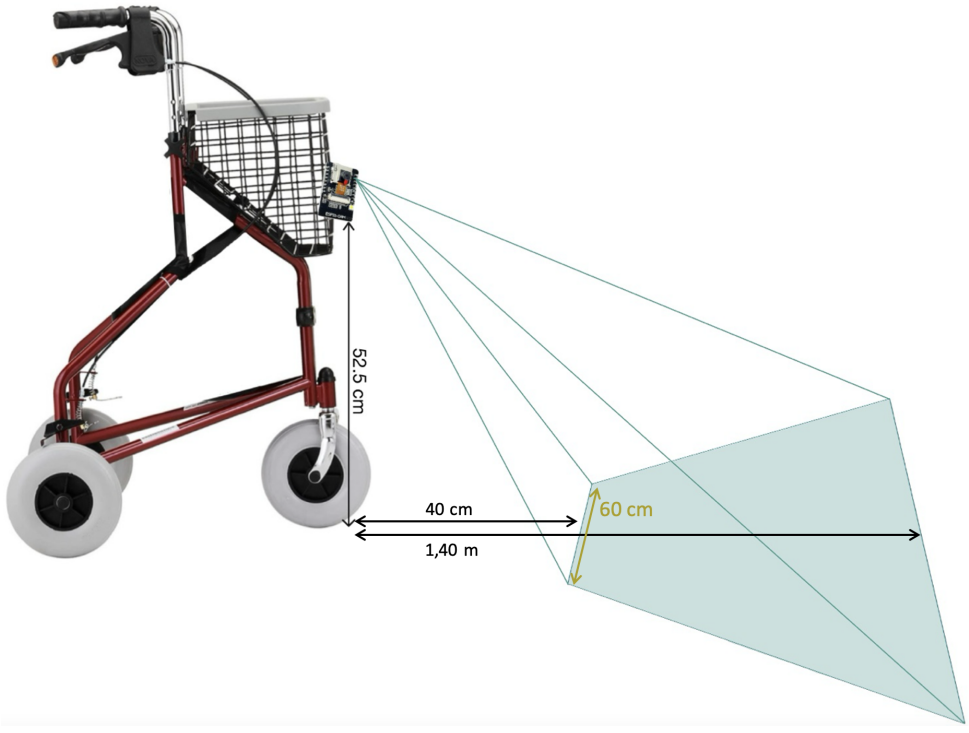


Figure 4.2: ESP32-CAM vision field.

The camera requires a permanent power source, which in this scenario is a power bank connected via the FTDI.

4.1.2 Image acquisition

To generate the dataset necessary for this specific project, a dedicated code was written to capture photos using the ESP32-CAM and subsequently save them onto the SD card attached. A detailed examination of the main sections of the Arduino code is provided below.

The following section of the code shows how to take a picture using the ESP32-CAM.

A pointer named **'fb'** of type **'camera_fb_t'** is initialised to NULL. After, using the ESP32 Camera library function **'esp_camera_fb_get()'**, a photo is taken and a pointer is returned to a **'camera_fb_t'** structure, representing the frame buffer of the captured image. The pointer is assigned to the **'fb'** variable.

A check is then made to ensure that an image has been correctly captured, otherwise an error message is printed.

The **'esp_camera_fb_return(fb)'** line is used to release or return the memory associated with the frame buffer (fb) after it has been used.

```
1  camera_fb_t * fb = NULL;
2    // Take Picture with Camera
3    fb = esp_camera_fb_get();
4    if(!fb) {
5      Serial.println("Camera capture failed");
6      return;
7    }
8
9    {...}
10
11   esp_camera_fb_return(fb);
```

Listing 4.1: Image capture

The following code is used to save the image taken with the code above to the SD card installed on the board.

The ESP32-CAM is endowed with Electrically Erasable Programmable Read-Only

Memory (EEPROM), i.e., non-volatile data, that can only be erased or changed by programming. In this case, it's being used to store the number of images taken, which is later used to save the name of the photo.

The EEPROM is initialised and the value of the first EEPROM address is read and incremented by 1. This value is stored in the variable **'pictureNumber'** and the path to save the new picture is created (**'String path = "/picture" + String(pictureNumber) +".jpg"'**).

The code then opens a file on the SD card in write mode (**'fs.open'**). If the file is opened successfully, it writes the image data from the frame buffer to the file, i.e., saves the photo, prints a success message to the serial console, updates the image number stored in EEPROM, commits the changes, and closes the file.

If there is a problem opening the file, an error message is printed. This sequence ensures that each image captured is stored on the SD card with a unique filename and that the image number is updated for the next capture.

```
1  // initialize EEPROM with predefined size
2    EEPROM.begin(EEPROM_SIZE);
3
4    pictureNumber = EEPROM.read(0) + 1;
5
6    // Path where new picture will be saved in SD Card
7    String path = "/picture" + String(pictureNumber) +".jpg";
8
9    //Initialising the filesystem object with the SD card
10   fs::FS &fs = SD_MMC;
11   Serial.printf("Picture file name: %s\n", path.c_str());
12
13   File file = fs.open(path.c_str(), FILE_WRITE);
14   if(!file){
15     Serial.println("Failed to open file in writing mode");
16   }
17   else {
18     file.write(fb->buf, fb->len); // payload (image), payload length
19     Serial.printf("Saved file to path: %s\n", path.c_str());
20     EEPROM.write(0, pictureNumber);
```

```
21      EEPROM.commit();
22    }
23    file.close();
```

Listing 4.2: Image saving

The complete code for capturing photos with the ESP32-CAM is available in the appendix **A** of this dissertation.

All of these functionalities are declared within the setup() function, indicating that the board captures a single picture each time it is initialised and does not persistently take an indefinite number of photos. Initiating the capture of a single photo simply requires resetting the microcontroller, by either pressing the reset button or disconnecting and reconnecting its power supply.

A customized dataset of 898 images was created from frames taken while walking around in the Department of Electrical and Computer Engineering (DEEC) at University of Coimbra (UC).

4.1.3 Labeling

As highlighted in section **3.2.4**, the classification task on this work involves multiple classes, specifically four in total.

The figure **4.3** represents the schematic of the walker with its 3 wheels and its dimensions. Additionally, the field of view is shown. All dimensions are presented in real scale.

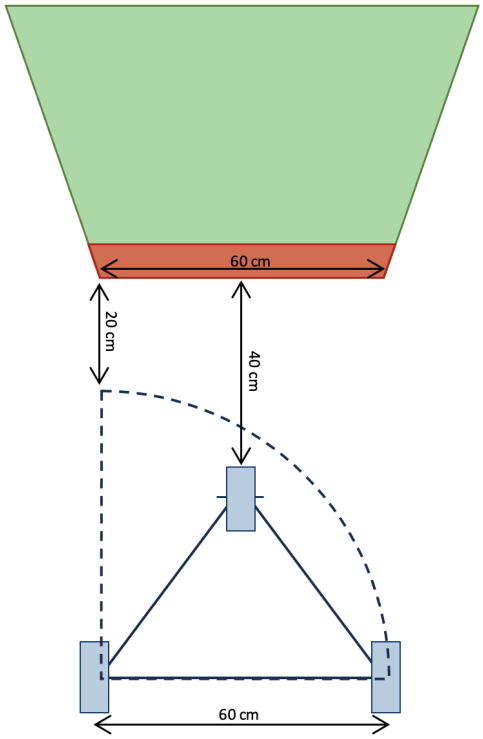


Figure 4.3: Manoeuvrability of the walker.

It is possible to notice that the bottom dimension of the trapezoid formed by the field of view matches the distance between the back wheels of the walker. This indicates that if an obstacle appears in the bottom frame, even in the corners, it will impact the walker's path. Similarly, if there is free space in the middle of the frame but obstacles on both sides, there will not be sufficient space for the walker to pass.

When examining the manoeuvrability, by applying brakes to one of the wheels to induce a turn, the walker can successfully avoid an obstacle located at the bottom of the frame with, at least, a safety distance of 20 cm.

The smaller red trapezoid in the figure **4.3** represents the lower portion of the field of view that triggers an action in the walker to avoid an obstacle. It has a height of 14 cm, equivalent to 10% of the total view distance of the camera.

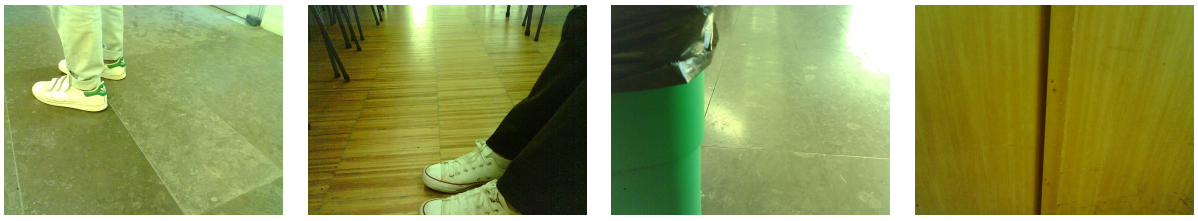
After evaluating the manoeuvrability of the walker, the classification process of the dataset was done manually. To ensure consistency and uniformity during the labelling, the following points were determined:

- The obstacles were assumed to be static. Although the walker is still capable of

functioning when encountering moving obstacles, it will have a smaller margin of free space to intervene effectively when dealing with moving obstacles.

- An object that is not at the bottom of the frame, i.e. on the red trapezoid, does not interfere with the walker's path;
- Only objects on the bottom of the frame, on the red portion, induce an action to the walker;
- An object on the right bottom of the frame will make the walker turn left in order to avoid collision;
- An object on the left bottom of the frame, will make the walker turn right to avoid a collision;
- The walker will attempt to avoid the obstacle by turning left or right, resorting to full braking only if avoidance is impossible.
- An unavoidable obstacle or specific scenarios, such as stairs, will make the walker make a full stop;

The following set of images shows an example of each class:



(a) Free Way (b) Left Turn (c) Right Turn (d) Full Stop

Figure 4.4: Example of each output class.

In figure **4.4a**, the absence of obstacles in the lower part of the frame is evident, resulting in a classification of 0: Free Way. No brake will not be engaged, allowing the walker to move freely.

In the second image (figure **4.4b**), the presence of obstacles, notably the feet at the bottom right, represents an obstacle to the walker's path. Consequently, the image is classified as 1: Left Turn. This classification causes the left brake to be activated so that the walker turns left to avoid the obstacle without collision.

The figure **4.4c** unfolds a scenario with an obstacle positioned at the left bottom of the frame. In response to the need to get around the obstruction by the right, the image is appropriately classified as 2: Right Turn. As a result, the right brake will engage, enabling the walker to avoid the obstacle.

In the case shown in figure **4.4d** the walker faces a door that is already too close. Because turning will not avoid a collision, the only solution is to classify it as 3: Full Stop, making the walker activate both brakes.

After labelling the entire dataset, the distribution per class achieved is depicted in figure **4.5**. As represented, the dataset is almost evenly distributed along the 4 classes (Free Way, Left Turn, Right Turn, Full Stop).

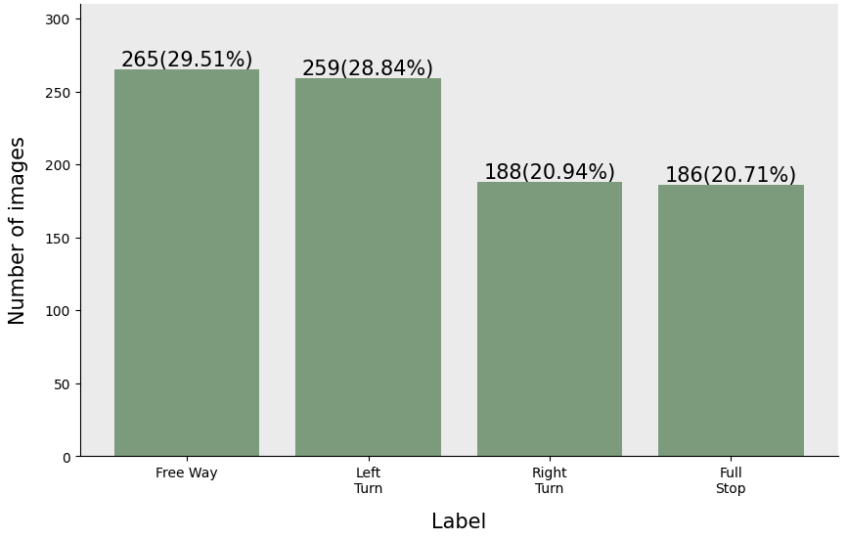


Figure 4.5: Images per class.

The dataset for the 'Full Stop' class represents specific scenarios, such as stairs and walls that are too close, making it a class with very specific features.

4.1.4 Data split

The dataset was then split into 3 sets, training, validation and testing. The original goal was to split the dataset using 70% for training, 15% for validation and 15% for testing. Due to adjustments made to the dataset, such as the removal of images that did not accurately represent the class or the addition of new images, the final distribution has slightly changed, as shown in the following graph:

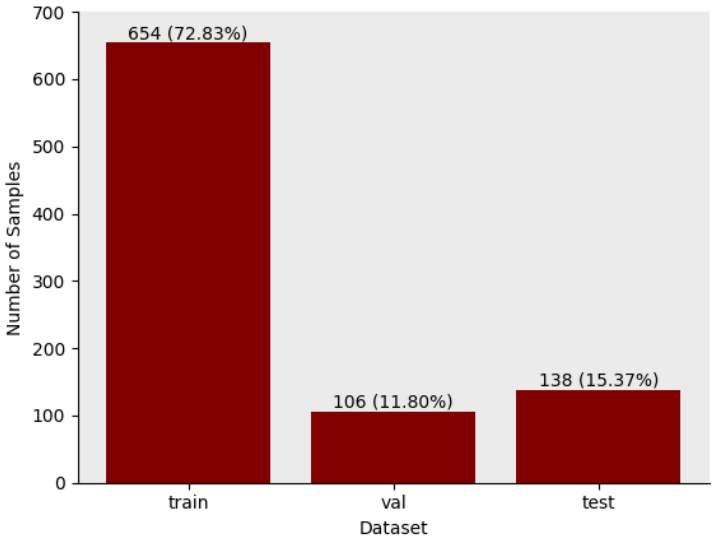


Figure 4.6: Dataset split.

This way, the data used for each stage of the process is isolated from other stages, reducing the risk of overfitting the model, where it may fail to generalize to unseen data.

4.1.5 Data pre-processing

Before feeding input images into the model, a preprocessing step is essential to ensure that all images meet the model's correct format.

The PyTorch library has a function named '**transforms.Compose**' within 'torchvision.transforms'. This function enables the combination of multiple image transformation operations into a single callable object. The object can then be applied sequentially to input images. It takes a list of transformation functions or classes as input and returns a composed transformation object that applies these transformations in the order they are listed. The following listing shows the "**preprocess_image**" function created to preprocess the images before feeding them to the model:

```
from torchvision import transforms
from PIL import Image

preprocess_image = transforms.Compose([
  transforms.Resize(256),
  transforms.CenterCrop(224),
  transforms.ToTensor(),
```

```
    transforms.Normalize(
    mean = [0.485, 0.456, 0.406],
    std = [0.229, 0.224, 0.225])
])

[...]

img = Image.open(path)
images.append(preprocess_image(img.convert('RGB')))
```

Listing 4.3: Python preprocessing.

When training and testing a model, consistency is important. A model should always be run with inputs of the same features. The values used on listing **4.3** match the needs of Alexnet described on the official paper [12] and are commonly used on the pre-processing steps for models trained on the ImageNet dataset. These values were originally introduced in the PyTorch documentation and are based on an extensive statistical analysis of the ImageNet dataset, ensuring that the preprocessing steps adequately capture the typical distribution of pixel values in natural images[32].

In this pre-processing, an image starts by being converted to RGB, resized to a square shape with dimensions 256x256 pixels and then center-cropped to obtain square images with dimensions 224x224 pixels.

Subsequently, the images are converted to tensors. This transformation converts the input image, which is typically represented as a NumPy array with pixel values in the range [0, 255], into a PyTorch tensor where the pixel values are normalized to the range [0, 1] by dividing each pixel value by 255.

After converting to tensors, there is a second normalization process. This normalization involves subtracting the mean and dividing by the standard deviation of the RGB color channels, ensuring that each channel has a mean of approximately 0 and a standard deviation of approximately 1.

It is imperative to apply the same normalization procedure to both the training and test datasets to ensure uniformity in processing.

4.2 Model training

As mentioned in the section **3.2.3**, transfer learning is a technique that allows the use of a pre-trained neural network to modify its output and retrain it with a new dataset. This approach saves time and enables the training of a new model with a smaller dataset.

To modify the original 1000 classes from the Alexnet to 4 classes, the function in listing **4.4** was developed. The function accepts the pre-trained neural network as an input and, in the subsequent steps, lines 2 and 3, iterates through all layers, effectively freezing their parameters and rendering the model non-trainable.

Following this, in lines 6 and 7, the function extracts the size of the last layer and establishes a full connection to a new decision layer, a fresh layer configured with the desired number of outputs for the model.

In the subsequent lines, 9 to 12, the function attempts to alter the model's classifier attribute by substituting the last layer with a new linear layer having an output size of 4. If the classifier attribute is absent, it falls back to modifying the 'fc'. This approach ensures adaptability, catering to variations in the model's structure while maintaining a consistent modification procedure.

```
1  def change_out(model):
2    for param in model.parameters():
3      param.requires_grad = False
4
5    # Size of the last layer to fully connect to the decision layer
6    last_layer = [key for key in model.state_dict().keys()][-2]
7    last_layer_size = model.state_dict()[last_layer].shape[-1]
8
9    try:
10     model.classifier = nn.Sequential(list(model.classifier)[:-1],
       nn.Linear(last_layer_size, 4))
11   except Exception:
12     model.fc = nn.Linear(last_layer_size, 4)
13   return model
```

Listing 4.4: Changing the output to 4.

4.3 Model testing and results

To evaluate the probabilities of each class for a given image, the following section of code was developed.

The program randomly selects an image from the test set and performs an inference, generating an array of probabilities for each class using the softmax layer. The resulting probabilities are then plotted alongside the image on a bar chart.

```
for i in np.random.choice(range(test_images.shape[0]), replace=
    False, size=1):
  predictions, accuracy = perform(i)
  predictions = softmax(predictions[0])
  max_value_index = np.argmax(predictions)

  plt.figure(figsize=(6, 3))
  plt.subplot(1, 2, 1)
  plt.title(f"\nLabel: {label[test_labels[i]]}")
  plt.imshow(test_images_origin[i])
  plt.subplots_adjust(hspace=0.5)

  plt.subplot(1,2,2)
  colors = ['gray' if i != max_value_index else 'blue' for i in
    range(4)]
  plt.bar(['Free Way', 'Left\nTurn', 'Right\nTurn', 'Full\nStop']
    , predictions, color = colors)
  plt.show()

```

Listing 4.5: Model Testing

The probabilities displayed on the charts are converted from raw scores to probabilities using the softmax function, the decision layer, to determine the final output based on the highest probability.

The image below (figure **4.7**) shows a scenario captured by the walker's camera. In

response to this input, the model accurately predicts the output as "Free Way", aligning with expectations. The chart displays the probabilities assigned to each output, showing the model's high confidence in its predictions.

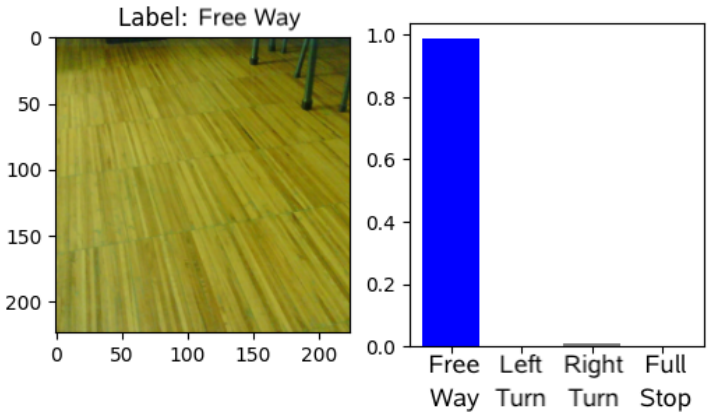


Figure 4.7: Free Way - output probabilities.

In the absence of any obstacles directly in front of the walker, the model outputs the response "Free Way", exhibiting an exceptionally high probability level, nearing 100%. Even if low, the model still assigned a small probability to the output "Right Turn".

In the scenario displayed in figure **4.8**, the camera captured the presence of a table leg emerging on the right side. Without intervention, the walker is at risk of colliding with it. The model's post-analysis determined that executing a left turn would successfully avert the potential collision.

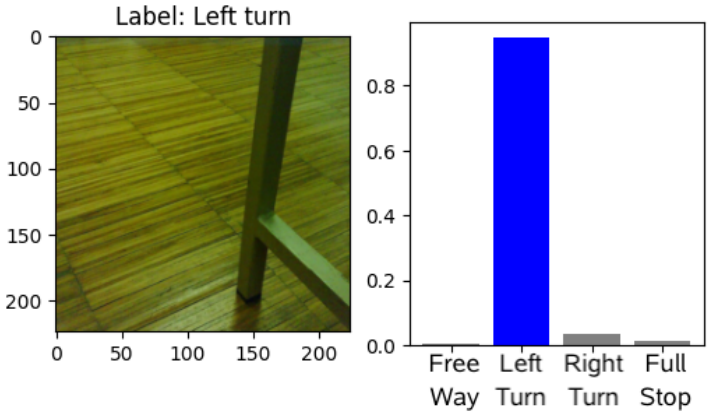


Figure 4.8: Left Turn - output probabilities.

The high probability for "Turn Left" suggests that the model has learned that turning

left is an appropriate action in the presence of an obstacle on the right side (the table leg). Even though the model has a high confidence in its final answer, it still has low probabilities for the other outputs. The low probabilities for "Free Way" and "Turn Right" are consistent with the model's understanding that in this context, going straight or turning right is less appropriate due to the obstacle on the left, which should be avoided. The low probability for "Full Stop" suggests that the model does not consider the immediate stop solution very much. Rather, given the possibility of avoiding the obstacle, the most appropriate action is to turn left.

On another occasion, upon entering a bathroom, the camera captures an image reflecting a wall on the left side (figure **4.9**). In response, the model instinctively seeks to avoid it by triggering the right brake and initiating a right turn to navigate around the obstacle.

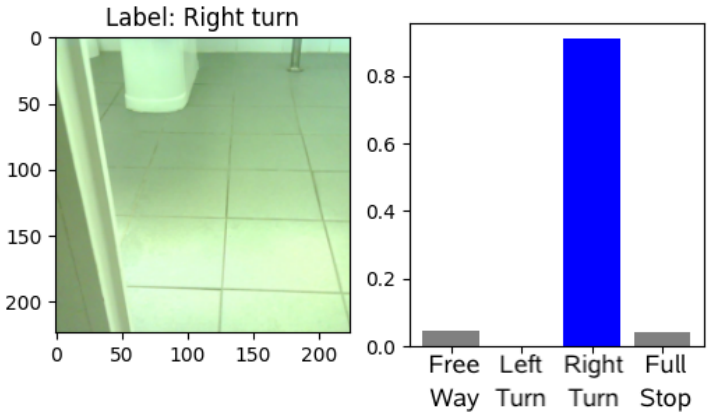


Figure 4.9: Right Turn - output probabilities.

After analysing the prediction graph, it can be seen that the model exhibits a small consideration for proceeding with no interference or applying both brakes, while entirely excluding the possibility of executing a left turn. By examining the model behaviour and analysing the output probabilities, the results can be analysed objectively.

- **Free Way:** By predicting a non-zero probability even when there's an obstacle on the left side of the frame, it suggests that the model might be considering the broader context of the scene. It could be interpreting the scene as having a navigable path on the right, and therefore, it suggests moving freely.
- **Full Stop:** A non-zero probability in the presence of an obstacle that is still avoidable, indicates that the model recognizes the obstacle and suggests stopping. This

aligns with the model's training objective to associate obstacles with the action of stopping.

- **Left Turn:** The low probability (or even 0%) when there is an obstacle covering the whole left side, implies that the model is learning to associate the presence of an obstacle on the left with a lower likelihood of turning left. This could be a reasonable behaviour if, during training, the model encountered similar scenarios where turning left was less likely due to the obstacle.

One of the scenarios for which the model has been specifically trained is the detection of stairs and the immediate application of both brakes to prevent the fall of the walker and its user.

As shown in Figure **4.10**, the detection of a staircase in the camera's field of view causes the model to output a full stop action with a high degree of certainty.

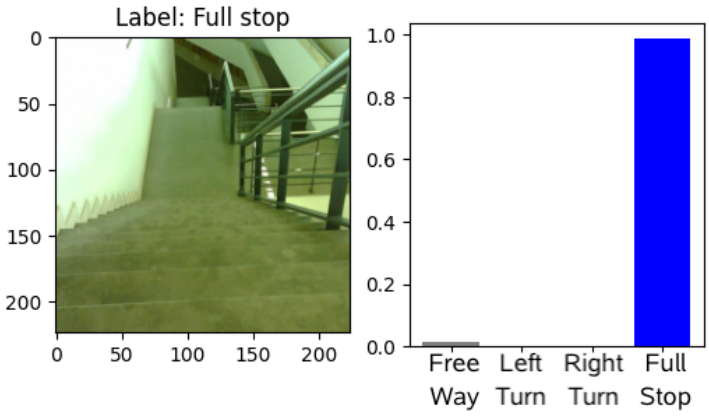


Figure 4.10: Full Stop - output probabilities.

Another interesting observation can be made from one of these examples, shown in the following figure:

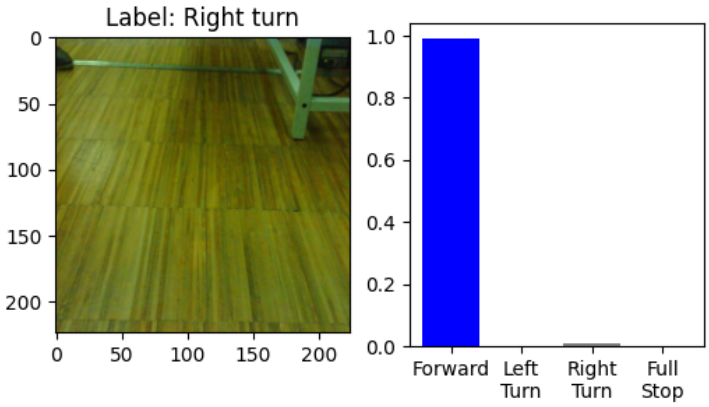


Figure 4.11: Mislabelled image - output probabilities.

At first glance, it is noticeable that the model misclassified the image labelled as "Right Turn" as "Free Way." However, upon closer examination, it becomes clear that a mislabelling occurred initially. No visible obstacle obstructs the walker's path, indicating that the image should indeed be classified as "Free Way," as the model correctly predicted with a high level of certainty. This example highlights the model's ability to recognise patterns and correct human errors, demonstrating its learning capabilities and consistency.

After identifying and correcting the mislabel, the model was retrained.

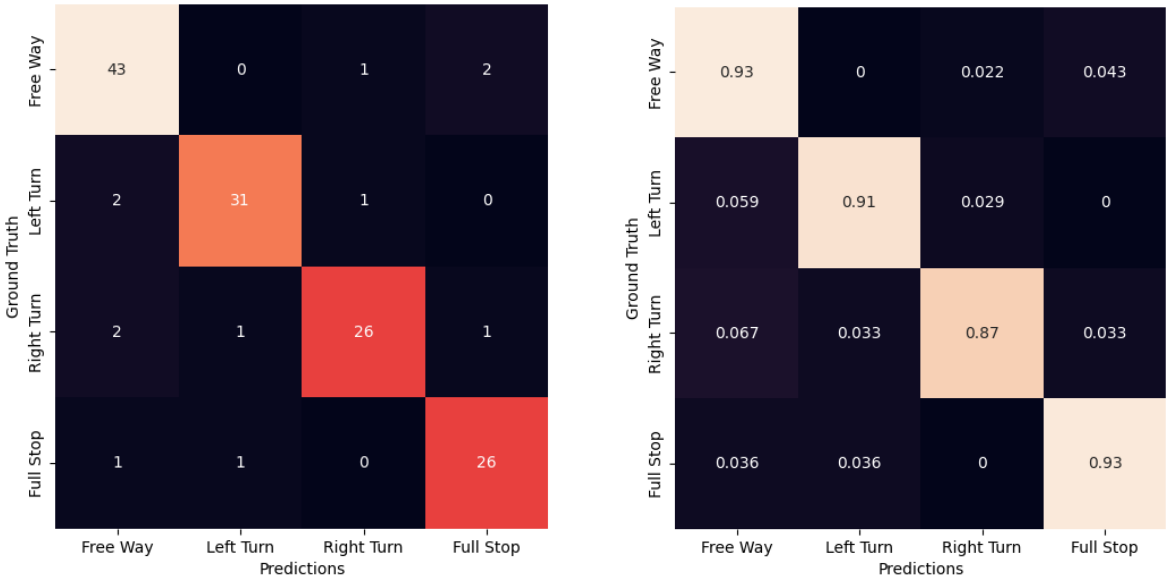
In order to evaluate how well the predictions were being made, a couple of metrics were determined, such as accuracy, precision, recall and F1 score. For a better understanding of the neural network behaviour, a confusion matrix and 12 images of the test dataset with the truth label and the predicted label were analysed.

The obtained **confusion matrices** are represented in figure **4.12**, being the ground truth represented by the rows and the predictions by the columns. The confusion matrix in figure **4.12a** shows the precise number of each prediction compared to its label.

Because the dataset is unbalanced, the representation with the specific value may be misleading. Thus, it is wise to also analyse a confusion matrix with normalised data, represented in figure **4.12b**.

In total, 138 images were analysed with the following distribution by class:

Free Way: 46; **Left Turn:** 34; **Right Turn:** 30; **Full Stop:** 28;



(a) Non-normalised data.

(b) Normalised data.

Figure 4.12: Confusion matrix.

Finally, after largely training and testing the model, good overall accuracy was achieved, around 91.30%.

To determine the overall accuracy of the model, it is important to consider its general definition: is the ratio of correct classifications (sum of diagonal elements in the confusion matrix) to the total number of predictions:

$$Acc = \frac{Correct\ predictions}{Total\ predictions} = \frac{43 + 31 + 26 + 26}{138} = \frac{126}{138} = 91.30\% \tag{4.1}$$

The individual metrics achieved by each class are represented in the following table:

	TP	TN	FP	FN	Accuracy	Precision	Recall	F1 score
Free Way	43	87	5	3	94.20%	89.58%	93.48%	91.49%
Left Turn	31	102	2	3	96.38%	93.94%	91.18%	92.54%
Right Turn	26	106	2	4	95.65%	92.86%	86.67%	89.66%
Full Stop	26	107	3	2	96.38%	89.66%	92.86%	91.23%

Table 4.1: Test metrics

From the metrics on the table **4.1**, some conclusions may be taken:

Figure **4.12b** shows that the model predicts more correctly the classes "Free Way" and "Full Stop" than the other classes, with 93% of true positives, followed by 91% of the "Left Turn" class. It also shows that the model is less accurate in classifying "Right Turn" images correctly classifying only 87% of the total images labelled as such. This lower rate may be due to the limited amount of data available for this particular class.

Accuracy:

The accuracy for the "Free Way" class is the lowest among all the classes. When looking at the class "Left Turn", an accuracy of 96.38% is achieved, indicating that 96.38% of the predictions for this class are correct in relation to the total number. For the "Right Turn" class, the value was 95.65%, indicating that 95.65% of the predictions made by the model for this class were correct.

It is worth noting that the model's overall accuracy is lower than the individual accuracies of the classes. This may be due to the limitations of using this metric to evaluate a model's performance, particularly when dealing with unbalanced data. Accuracy is a simple metric that only considers the number of correct predictions divided by the total number of predictions, without taking into account the severity of misclassifications.

Precision:

The precision is calculated as the ratio of TP to the sum of TP and FP. The value for the class "Free Way" indicates that around 89.58% of the inferences predicted as "Free Way" are correctly predicted. For the "Left Turn" class, the precision is 93.94%, meaning that 93.94% of the instances classified as part of this class were correctly identified. The "Right Turn" class demonstrates a strong value of 92.86%, indicating high certainty in its positive predictions. About the "Right Turn" class, 89.66% of instances predicted as belonging to this class were correct.

Recall:

Recall is calculated as the ratio of TP to the sum of TP and FN. Similar to the previous metric, the recall determines that from all the images labelled as "Free Way", 93.48% were correctly classified as such. In the case of the "Left Turn", the recall is 91.18%, indicating that 91.18% of "Left Turn" class instances were correctly identified

by the model. Additionally, it exhibits a recall of 86.67% for the "Right Turn" class, showcasing the model's ability to capture a significant portion of true positives. About the "Full Stop" class, it shows that 92.86% of all instances in the class were classified correctly.

F1 Score:

The F1 score metric can be more accurate when evaluating the performance of a model than accuracy itself. By calculating the harmonic mean of precision and recall, it is possible to estimate how precise a network is even with imbalanced data. At first glance, when analysing only the accuracy of each class, it would be logical to think that the model would classify more correctly images labelled as "Right Turn". However, the F1 score indicates that this class has the lowest score of all. This discrepancy is attributed to the dataset's imbalance, with fewer images available for testing in this class. As a result, each inference within this class carries more weight in determining the final accuracy, potentially inflating its apparent accuracy compared to its actual performance.

Although the dataset from the class "Full Stop" has almost the same number of images as the class "Right Turn", its higher F1 score may be due to its distinguishability dataset (more distinctive features) compared to the other classes.

It can be concluded that the 'Free Way' and 'Left Turn' classes have better results. This is likely due to the larger dataset for these classes, which allowed the model to be better trained and more familiar with their examples, resulting in more accurate classification.

Figure **4.13** demonstrates 12 examples of images from the test dataset. The prediction and ground truth are displayed at the top of each image, revealing the model's overall ability in prediction. However, upon closer examination, some notable observations can be made.

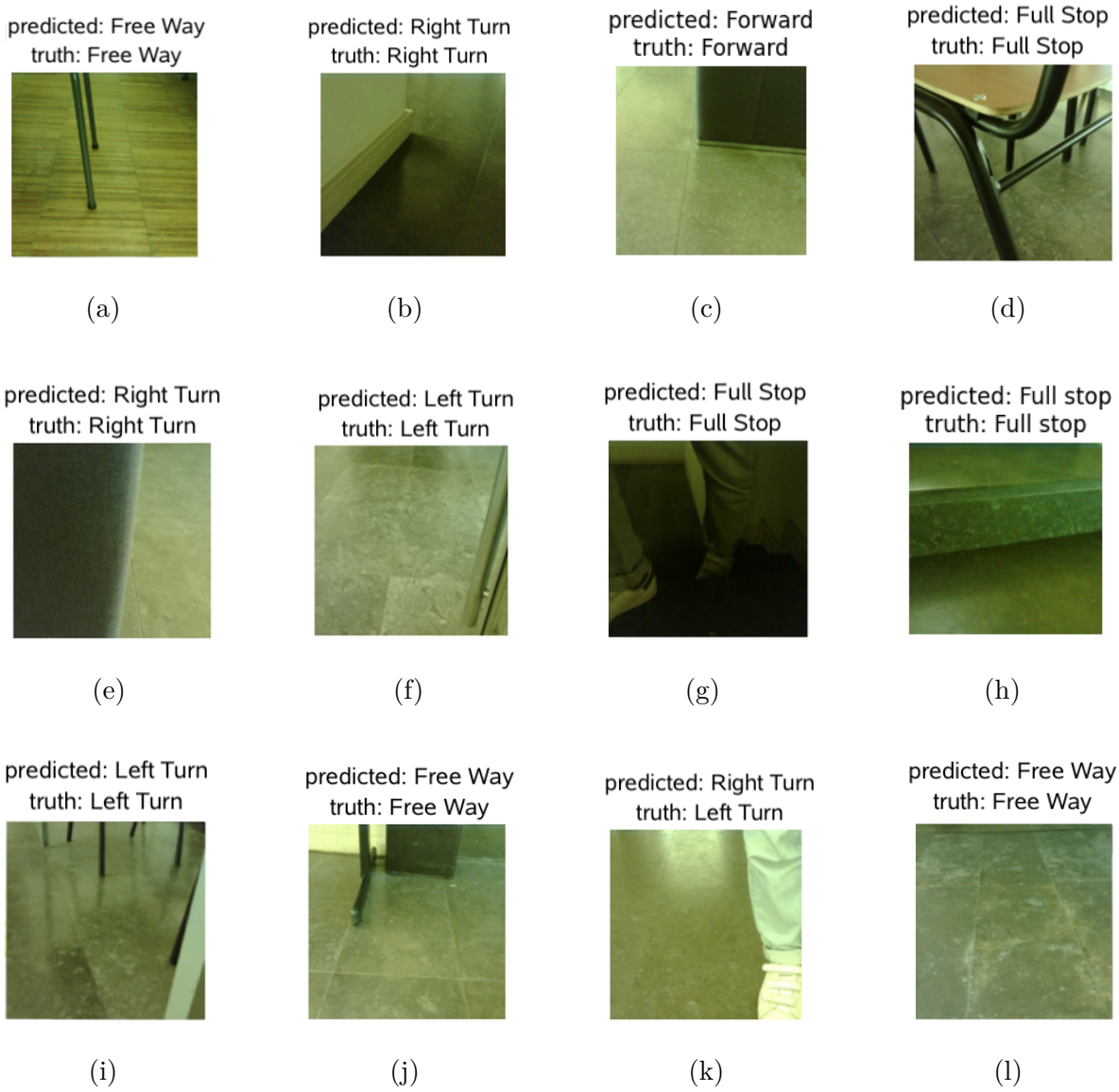


Figure 4.13: Comparison between predicted and ground truth labels for tested images.

Although the model achieved good accuracy, it still makes some mistakes when predicting certain scenarios. In the example in figure **4.13k**, the model predicts a "Right Turn" on an image labelled as "Left Turn".

As can be seen from the confusion matrix in figure **4.12**, the model correctly predicts 91% of the images labelled as "Left Turn". However, it's worth noting that the model occasionally misclassifies "Left Turn" images as "Right Turn" as can be seen in this example. When looking at the probabilities of each output for this inference, the following chart is achieved:

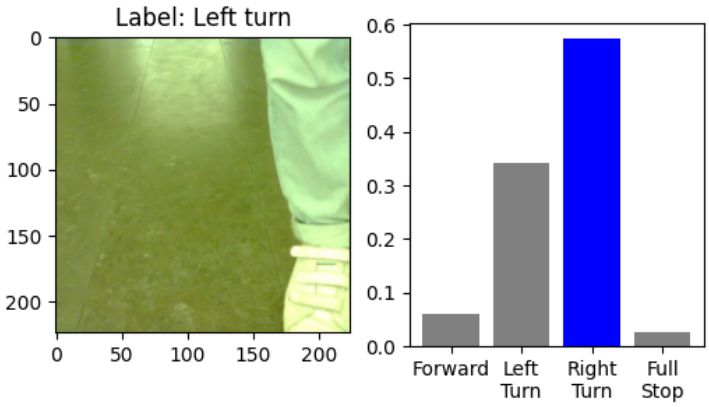


Figure 4.14: Wrong prediction - output probabilities.

As can be seen from the chart in figure **4.14**, although the model has misclassified the image, its confidence in the predictions is lower than in inferences with correct predictions, around 57%. This response, even if wrong, shows that the model has less confidence in this prediction having some hesitation and considering other classes.

4.4 Exporting the model to ONNX

Once the model has been completely trained, it should be exported to ONNX. The interconnectivity of ONNX enables easy integration of the trained model into the Jetson Nano. The following code was used in this conversion:

```
1  torch.save(model.state_dict(), 'model.pth')
2
3  input_tensor = torch.randn(1, 3, 224, 224, device=torch.device("
       cuda" if torch.cuda.is_available() else "cpu"))
4
5  torch.onnx.export(model,
6                    input_tensor,
7                    "model.onnx",
8                    verbose=True,
9                    input_names=['input'],
10                   output_names=['output'])
11
```

```
serializable_dict = {}
for key, value in best_acc.items():
  if isinstance(value, np.ndarray):
      serializable_dict[key] = value.tolist()
  else:
      serializable_dict[key] = value

# Write the serializable dictionary to the file
with open("acc_file.txt", "w") as file:
  json.dump(serializable_dict, file, ensure_ascii=False)
```

Listing 4.6: Export to ONXX.

The sample of code **4.6**, starts by saving the state dictionary of the Pytorch model, i.e., the weights and bias after the training, to a file named "model.pth". This allows saving the trained model so that it is possible to load it later for inference or further training without needing to retrain it from scratch.

To convert a model to ONXX, using "torch.onnx.export" the parameters are:

- **model:** model to convert;
- **input_tensor:** a tensor with the desired size of the inputs;
- **"model.onnx":** the name to use on the converted model;
- **verbose:** a flag (normally set to True) that determines whether additional information about the export process should be printed to the console or not;
- **input_names** and **output_names:** the name of the input and output layers;

The parameter **input_tensor** represents a tensor that specifies the desired input size for the model, enabling the converter to adjust accordingly. In this work, the "input_tensor" has the dimensions (1, 3, 224, 224) because of the size of the input images.

Afterwards, the script converts the **best_acc** dictionary into a format suitable for serialization, converting data structures or objects into a format that can be easily stored or transmitted and reconstructed later.

Finally, the serialized dictionary (**serializable_dict**) is written to a text file named "acc_file.txt" using the JSON format.

4.5 ESP32-CAM for Image Classification

The initial goal of this project was to use the microcontroller ESP32-CAM to capture frames and then process them through a neural network to determine and execute appropriate actions.

As said before, the ESP32-CAM is an affordable low-power board equipped with an ESP32 microcontroller, an integrated OV2640 camera and a microSD slot.

As mentioned in the previous section 3.2.5 on frame choice, the process of selecting a framework begins with an assessment of individual requirements. For deploying a deep learning model on a microcontroller, an optimal framework choice is TensorFlow Lite for microcontrollers, commonly known as TensorFlow Lite micro (TFLite-micro) [31]. TFLite-micro brings the power of machine learning to the microcontrollers, enabling the execution of deep learning models directly on these devices. With support for TFLite-micro on ESP32-CAM, it arises as a particularly suitable option for this specific application.

TensorFlow (TF) has an integrated tool that takes a model built with a TF core and converts it to a smaller, more efficient ML model format (optimized FlatBuffer format).

When using a pre-trained model on the ESP32-CAM, it becomes necessary to convert the trained model into a format compatible with the microcontroller, typically hexadecimal code. Since the ESP does not have a file system, the file needs to be exported to a data array to access the weights. This converted model must be imported into the microcontroller's memory so that it can be used for inference tasks directly on the device. This process allows the ESP32-CAM to perform machine learning tasks autonomously, eliminating the need for external server or cloud-based processing.

To convert the model to hexadecimal code, an approach is outlined in the following listing:

```
1 from everywhereml.code_generators.tensorflow import convert_model
2
3 # Converting the model to hex code
```

```
c_header = convert_model(model, X, y, model_name='mobile_model_c')
print(c_header)
```

Listing 4.7: TensorFlow Lite conversion

The code provided starts by importing the 'convert_model' function from a module named 'code_generators.tensorflow' within a package called 'everywhereml', responsible for handling the conversion of the TensorFlow model. The function 'convert_model' is called with the following parameters:

- **model:** The TensorFlow model to convert, already trained;
- **X:** The input data;
- **y:** The label data;
- **model_name:** The name to be attributed to the converted model;

The function will return a string containing the converted model in a C header file format.

The model is now fully encoded in hexadecimal code and is ready to be imported into the microcontroller.

While some models can run on the ESP32-CAM, it's important to notice that it remains an inexpensive microcontroller with inherent limitations. Attributes such as Random-Access Memory (RAM) can significantly limit the use of neural networks on these microcontroller boards, particularly for projects requiring image processing or video streaming. The process of capturing a frame, pre-processing it, feeding it into the neural network, running the model, and producing a decision can consume significant memory resources. Consequently, the RAM requirements of certain large, unoptimized models may exceed the capacity of the ESP32-CAM, making it unable to allocate sufficient memory to execute the neural network.

When attempting to convert the current model into a hex code file, a file containing 10 182 512 bytes, describing the network, was generated. This file is too large for the ESP32-CAM, which has limited RAM and flash memory. Additionally, the ESP32-

CAM may require memory for other tasks and libraries. If the model consumes excessive memory, it could result in insufficient resources for essential operations.

While this board remains suitable for low-cost IoT applications, it has been demonstrated that the model used in this work requires additional memory to function without issues.

4.6 Testing on Jetson Nano

The Jetson Nano is equipped with TensorRT [7], a Software Development Kit (SDK) for high-performance deep learning inference. TensorRT offers the possibility to run optimized networks on GPU from C++ or Python and PyTorch for training deep learning models.

4.6.1 NVIDIA container set-up

NVIDIA provides a pre-built Docker container for image processing [10] which can be automatically downloaded with a **$ git clone** command that includes a lot of pre-installed packages, such as PyTorch and deep neural network models, making really easy the fist steps on Jetson nano.

Assuming that everything has been properly cloned from GitHub, changing the current working directory to the 'jetson-inference' folder is the next step.

Any files saved inside the container will be lost when it is shut down. To avoid it, the files should be saved on the host device (Jetson Nano) by using a directory from the host device mounted into the container environment, allowing data to persist across sessions. The following command guarantees this:

```
root@ubuntu:~/jetson-inference$ ./docker/run.sh --volume ~/Walker:/Walker
```

The --volume argument should be followed by {directory on the host}:{directory inside the container}. In this case, the folder containing all the necessary data and scripts of the project is called "Walker".

Once the container is up and running there is the need to access the folder mounted with the volume command, simply by using the **cd** command to navigate to the "Walker" folder.

Before running the model from the container, the directory should be structured correctly to ensure proper access to the necessary files by the script. A figure with the directory structure is represented in the appendix **C**.

The labels file specifying the model outputs must be listed alphabetically to enable the script to match it with the correct folders:

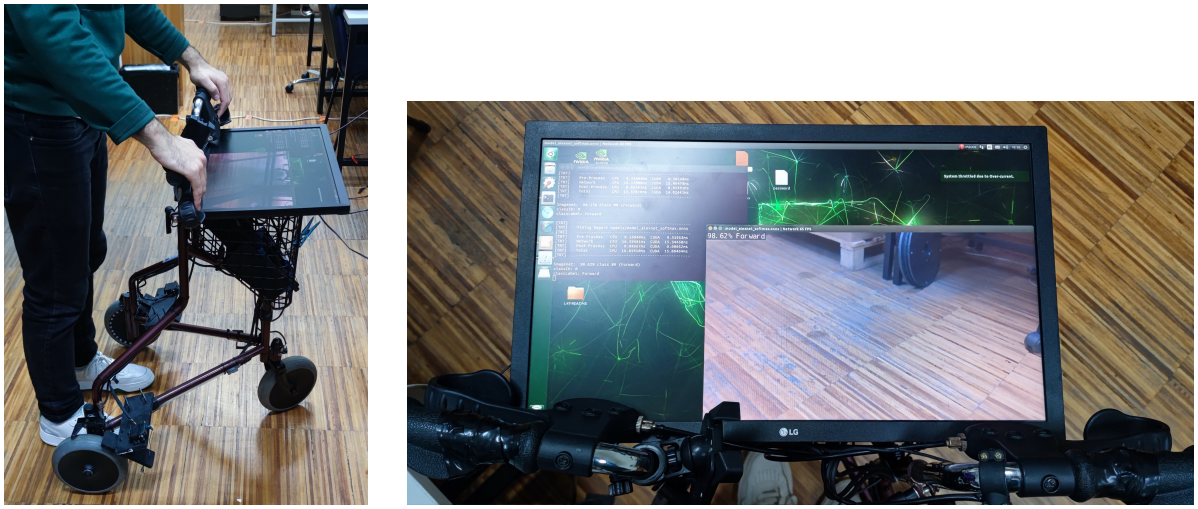
The file specifying the labels must be listed models output

Figure 4.15: labels.txt

The final layer must return a valid probability value to enable the script to interpret the output of the model. It is worth noting that pre-trained models trained on ImageNet images usually do not have a softmax decision layer, so it may need to be added. A script for this purpose can be found in the appendix **D**.

4.6.2 Testing assembly

To test the model with live streaming, the Jetson Nano connected to the Raspberry Pi camera was assembled on the walker. To visualize the live image and the output while testing, a screen was mounted on top of the walker and connected via HDMI to the Jetson Nano, as shown in the figure **4.16**.



(a) (b) User perspective.

Figure 4.16: Setup for testing.

4.6.3 Testing

Once all compatibility requirements have been confirmed and the board and camera are assembled, testing the model with a streaming camera is a simple matter of providing the camera identifier as input to the model, i.e. csi://0.

The necessary arguments for calling the **imagenet.py** script are:

- **--model:** path to custom model to load (caffemodel, uff, or onnx)
- **--input-blob:** name of the input layer (default is 'data')
- **--output-blob:** name of the output layer (default is 'prob')
- **--labels:** path to text file containing the labels for each class
- **csi://0:** Raspberry Pi camera identifier

The input and output layer names can be verified on the neural network's architectural representation in appendix **B**.

Additional parameters and flags are available when calling the **imagenet.py** and can be consulted on Jetson-inference official documentation [33].

Thus, the command to execute an infinite loop of inferences using live camera images as input is:

```
root@ubuntu:Walker#      imagenet.py --model=model.onnx --input_blob=input
--output_blob=output --labels=labels.txt csi://0
```

Note: To prevent freezing during training or testing the model, it may be necessary to mount a swap space for the Jetson Nano, depending on the computational resources used by the neural network. The commands to solve this problem are available in the appendix **E**.

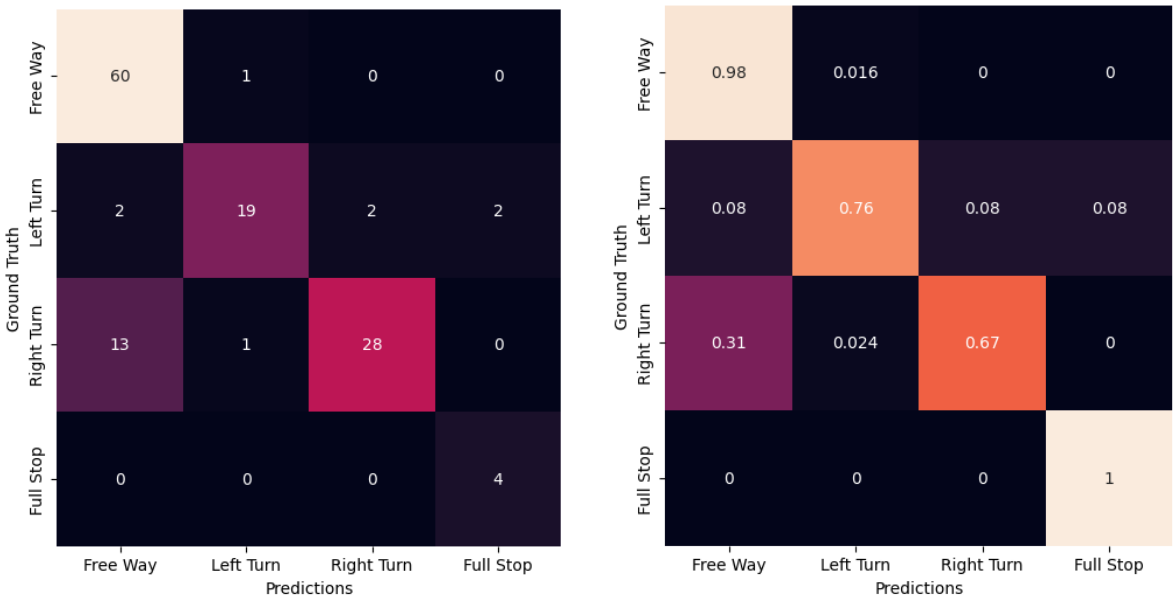
The camera is initialised and infinite inferences are made until a keyboard interruption occurs. The highest class probability and its label are displayed in the top left corner of the live image.

4.6.4 Results

To better evaluate the performance of the model in real time, a total of 132 frames were analysed. Each frame was evaluated by comparing the model's prediction with the ground truth (the human classification). In total, each class had the following number of images:

Free Way: 61; **Left Turn:** 25; **Right Turn:** 42; **Full Stop:** 4;

The confusion matrices generated are presented in figure **4.17**. The rows represent the labels and the columns represent the predictions. Figure **4.17a** shows each class's exact number of predictions. The confusion matrix in figure **4.17b** shows the data normalised.



(a) (b)

Figure 4.17: confusion matrix from the live test.

It is noticeable that the accuracy of the live test is generally lower than the test conducted on Google Colab. This is expected as there are factors in real-life scenarios that may affect the model's performance, such as the latency between taking the photo, inputting it into the model, processing it, and outputting a prediction.

As can be seen from the figure **4.17**, the model tends to better correctly predict 'Free Way' and 'Left Turn' situations as opposed to 'Right Turn'. This discrepancy could be attributed to variations in the dataset sizes during training. Additionally, the model had the highest accuracy for 'Full Stop' images, although this may be due to the limited number of images labelled as such in this test set.

Free Way

Figure **4.18** shows an image that reveals an obstacle at the top of the camera's field of view.

In this scenario, the feet are positioned on the upper half of the frame, at approximately 70 cm from the walker, allowing some space to move without obstruction (about 30 cm). Therefore, according to the points defined in section **4.1.3**, the model accurately classifies the input as 'Free Way' with a 99.97% probability.

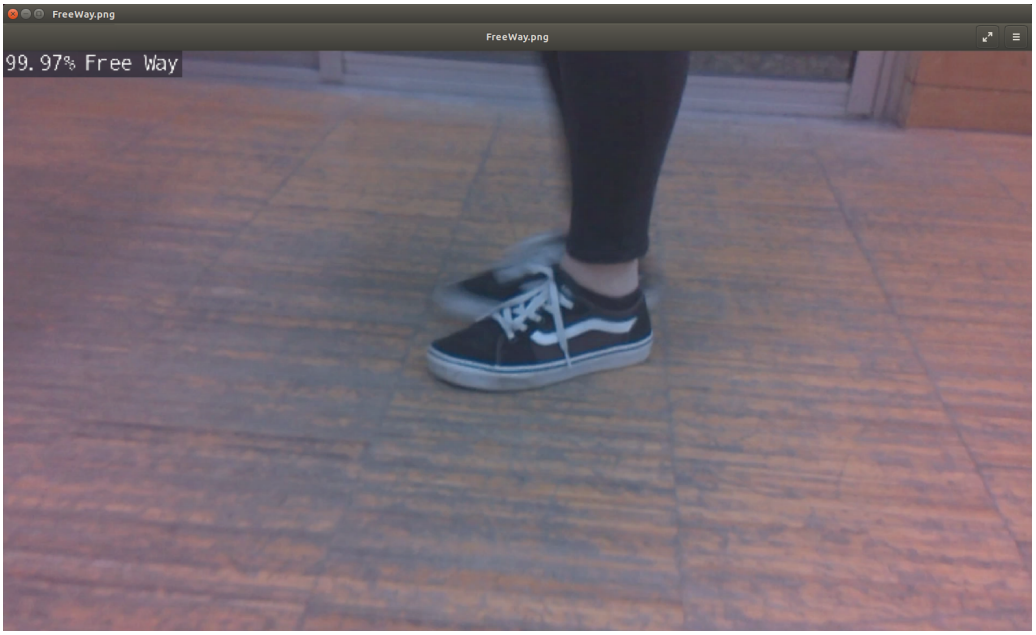


Figure 4.18: Free Way scenario on live test.

Left Turn

In the upcoming scenario, the walker encounters a wall on the right side, without intervention, it will collide with it. Therefore, according to the points defined in section **4.1.3**, the model must predict a "Left Turn" to navigate away from the obstacle, as it does with a high level of certainty.

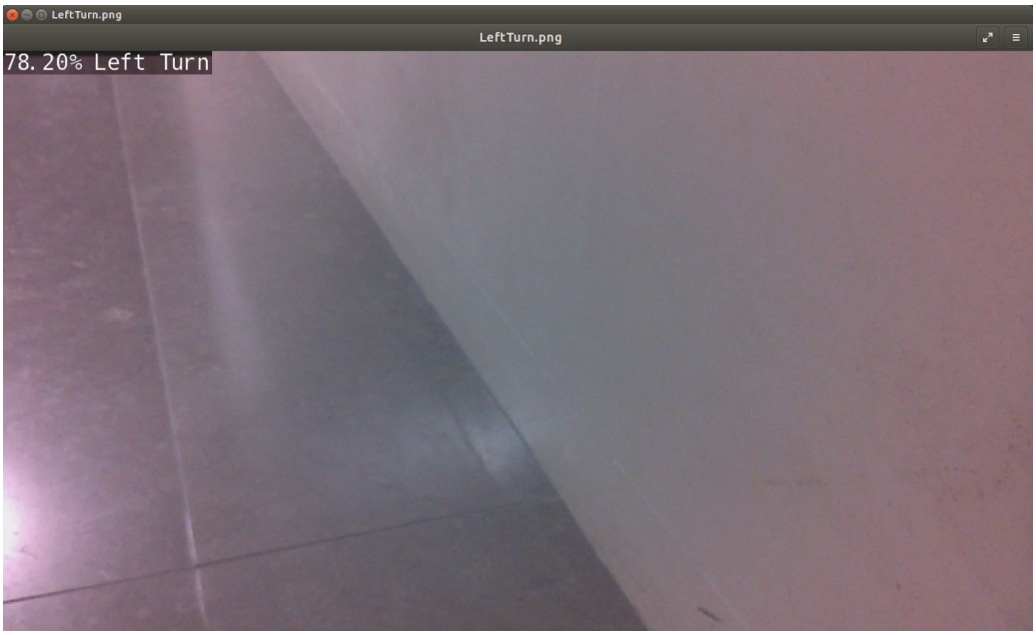


Figure 4.19: Left turn scenario on live test.

Right Turn

In the scenario represented in figure **4.20**, the walker faces an obstacle on the left side. To avoid a collision, the neural network accurately predicts the need to make a right turn, consistent with the points outlined in section **4.1.3**, with a probability of 90.98%.

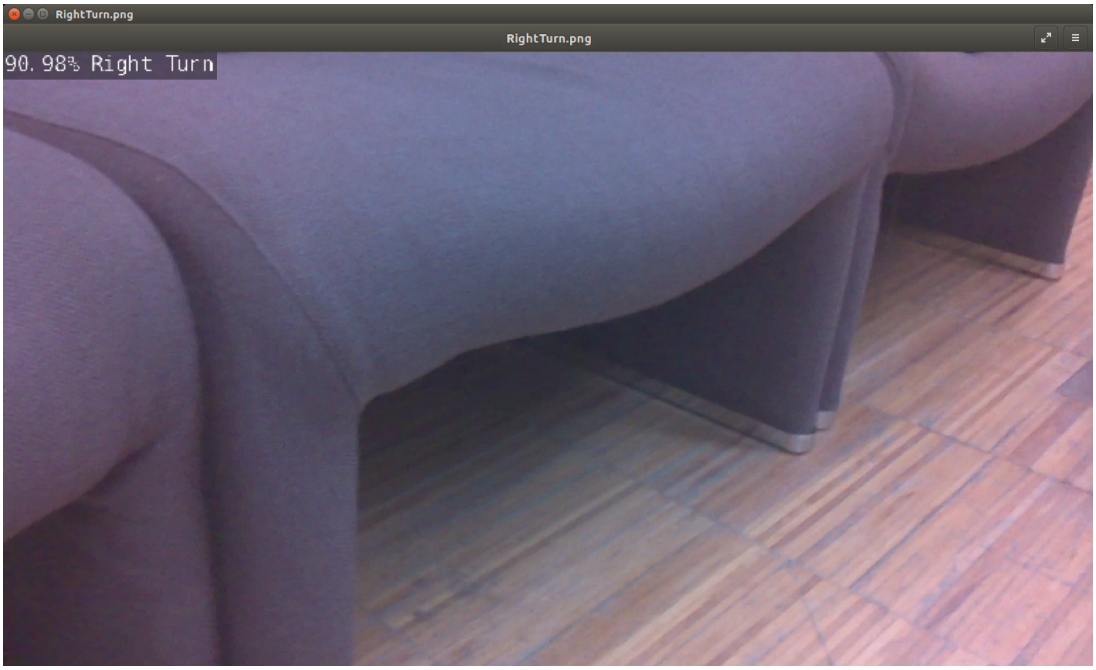


Figure 4.20: Right turn scenario on live test.

Full Stop

On the situation represented on the image **4.21** the walker encounters stairs in front of it. Because of the big and common dangers of the elderly falling from stairs, this is one of the special scenarios where the neural network was trained to predict "Full Stop". The model predicts the expected output with a probability of 72.64%, avoiding life-threatening scenarios.

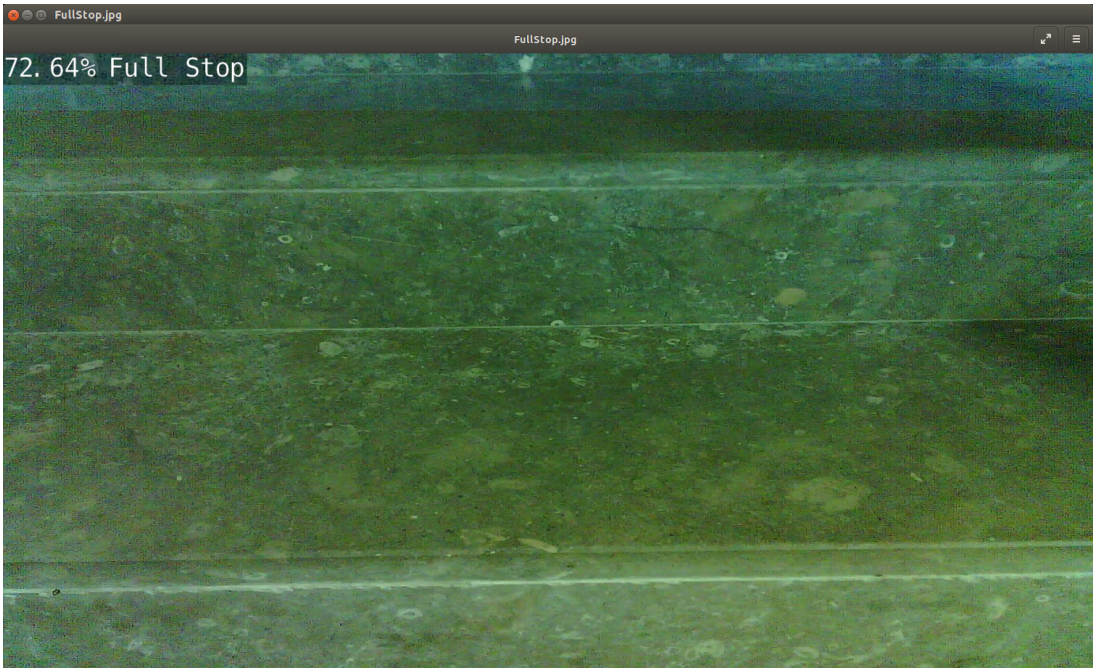


Figure 4.21: Full stop scenario on live test.

4.6.5 Wrong predictions

Although the model achieves good accuracy, like all deep learning models, there are still some incorrect predictions.

Figure **4.22** shows a situation where the model predicted an image as "Left Turn". At the first instance, it is possible to see that it is wrongly predicted, the image is labelled as "Free Way", although, it is possible to see why the model wrongly predicted it. A reflection of light is visible within the frame, which the model interprets as an obstacle situated on the right side. Consequently, in its effort to avoid a collision with the supposed obstacle, the model classifies the image as a "Left Turn".

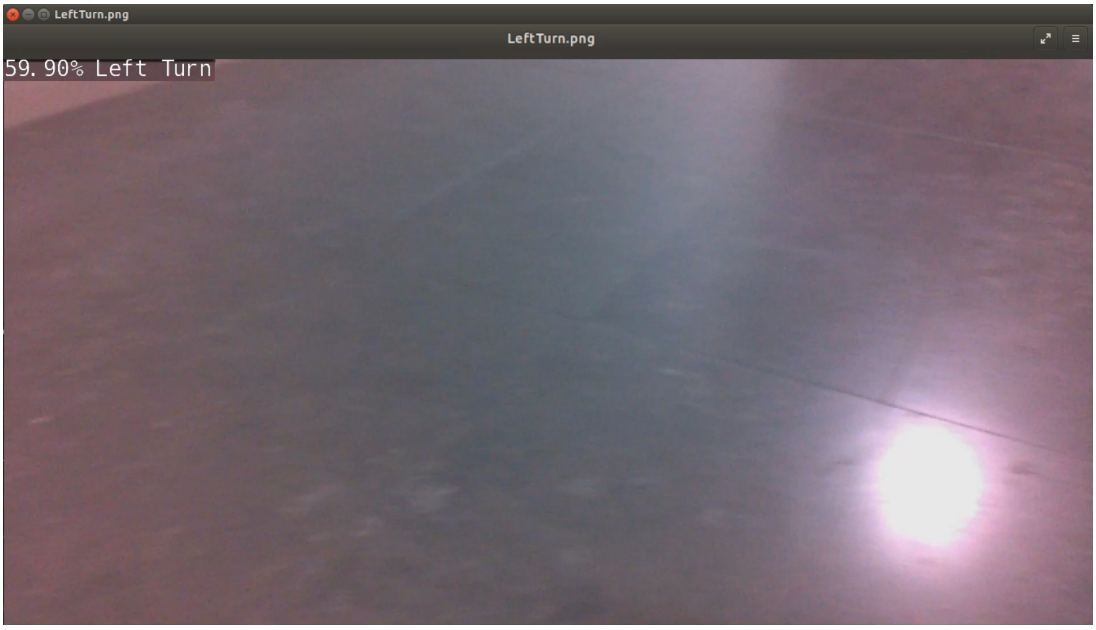


Figure 4.22: Free Way wrongly predicted as Left Turn.

To prevent further misclassifications caused by artificial light, additional data should be included in the dataset to retrain and prepare the model for this scenario. Another possible solution is to employ an alternative preprocessing technique that eliminates the reflection, thereby avoiding further incorrect predictions.

In the context of this dissertation, although one class may be more accurate than others, it is still possible that another label could also be considered correct.

In the situation shown in the figure **4.23**, the image was labelled as a "Left Turn" but the model predicted it as a "Full Stop". Although the label and the prediction do not match, the classification given by the model is not entirely incorrect and will not put the user in any dangerous situation. On the contrary, it is safe to classify it as such.

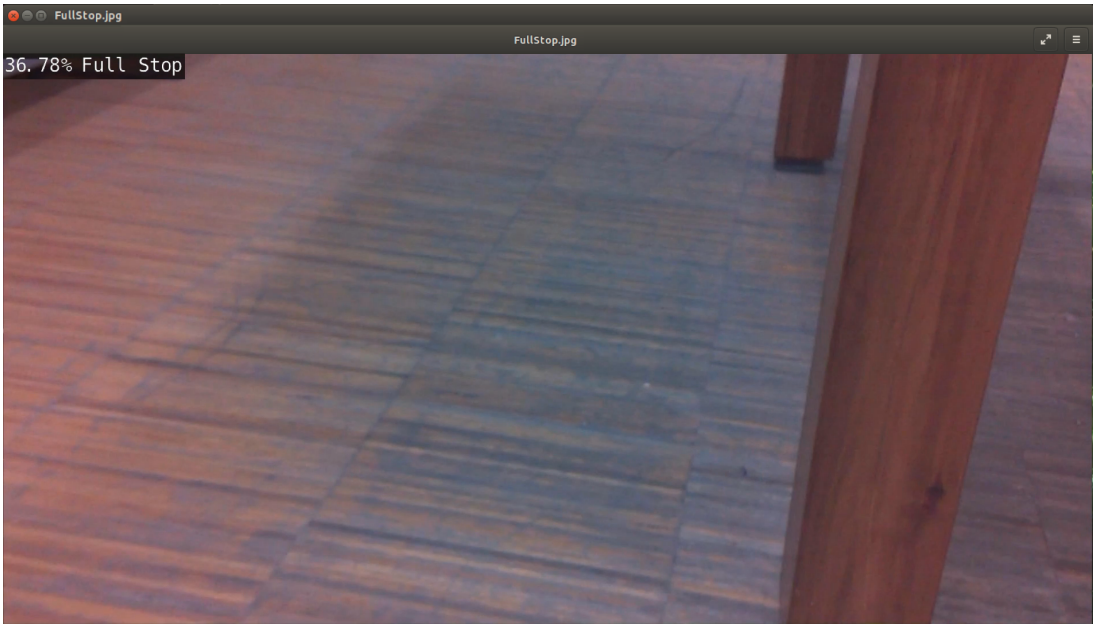


Figure 4.23: Left Turn wrongly predicted as Full Stop.

The situations presented contribute to lowering the accuracy of the model, although it is possible to understand why the images have been predicted as such.

5

Conclusion

The goals of this dissertation to develop and evaluate an object avoidance through image classification on a smart walker were achieved. A deep learning model was trained and tested, achieving good results in both the computer test and the real-time test with the walker. The model was capable of ensuring the safety of the user by either avoiding collisions or dangerous situations by going around the obstacle or performing a full stop. These results prove that the architecture used for the neural network is capable of classifying the images according to the defined goals.

The walker was endowed with a board and a camera that captures real footage from its front. The board then pre-processes the images, runs the deep learning model, and finally outputs the prediction (Free Way, Left Turn, Right Turn, Full Stop).

The board Jetson Nano by NVIDIA, together with the Raspberry Pi camera, was proven to be a reliable choice, capable of capturing images, pre-processing them, running the deep learning model, and outputting the prediction with no difficulty.

The ESP32-CAM was also tested for use in this project, but although the board is suitable for general IoT projects, it proved unsuitable for the purposes of this dissertation as it is a board with limited memory and resources.

Assistive technology and smart walkers are evolving rapidly to meet the needs of people with mobility difficulties. To fully leverage the potential of this project, the following future work is proposed:

- To improve the model's accuracy, expanding the dataset and retraining the deep learning model are necessary steps. By ensuring a more balanced dataset, better results can be achieved not only while improving overall accuracy but also enhancing

the model's robustness and reliability across diverse scenarios and inputs.

- After the model has been retrained, integrating it with the motors on the brakes automates obstacle avoidance. However, this task is challenging because the process within the NVIDIA container does not have the necessary permissions to control GPIO pins on the Jetson Nano. Therefore, a workaround is to run two processes simultaneously: one inside the container that runs the deep learning model, and another external process that manages the brakes via GPIO pins. Effective communication between these two processes is crucial. The process running the deep learning model must inform the process controlling the GPIO pins of the actions required to avoid obstacles.
- Further extensive testing should be conducted, involving users, and analysing the adjustments made on the walker.

References

[1] H. S. Kaye, T. Kang, and M. P. LaPlante, "Mobility Device Use in the United States," 2020.

[2] LifeWalker, "UPWalker Lite." [Online]. Available: https://upwalker.com/products/upwalker-lite

[3] "Meet Camino." [Online]. Available: https://caminomobility.com/pages/meet-camino

[4] R. Annicchiarico, C. Barrué, T. Benedico, F. Campana, U. Cortés, and A. Martínez-Velasco, "The i-Walker: an intelligent pedestrian mobility aid," Jan. 2008, pp. 708–712.

[5] L. Palopoli, A. Argyros, J. Birchbauer, A. Colombo, D. Fontanelli, A. Legay, A. Garulli, A. Giannitrapani, D. Macii, F. Moro, P. Nazemzadeh, P. Panteleris, R. Passerone, G. Poier, D. Prattichizzo, T. Rizano, L. Rizzon, S. Scheggi, and S. Sedwards, "Navigation assistance and guidance of older adults across complex public spaces: the DALi approach," *Intelligent Service Robotics*, vol. 8, pp. 77–92, Apr. 2015.

[6] "Autonomous Cars: Levels of Autonomous Driving Explained," May 2022. [Online]. Available: https://ackodrive.com/car-guide/autonomous-cars-and-levels-of-autonomous-driving/

[7] M. Mishra, "Convolutional Neural Networks, Explained," Sep. 2020. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[8] "ESP32-CAM AI-Thinker Pinout Guide: GPIOs Usage Explained | Random Nerd Tutorials," Mar. 2020. [Online]. Available: https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/

[9] R. Carreira, "How to Begin and Blink a LED with ESP32 CAM Board," Jan. 2022. [Online]. Available: https://www.geekering.com/categories/embedded-sytems/esp32/ricardocarreira/esp32-cam-board-how-to-begin-and-blink-a-led/

[10] "NVIDIA TensorRT." [Online]. Available: https://developer.nvidia.com/tensorrt

[11] R. P. Ltd, "Buy a Raspberry Pi Camera Module 2." [Online]. Available: https://www.raspberrypi.com/products/camera-module-v2/

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[13] M. J. A. Carinhas, A. P. V. Eusébio, L. N. M. C. d. V. d. Carvalho, e. M. C. Lopes, and R. J. V. d. A. Braga, "Cuidados à pessoa com alterações da mobilidade - posicionamentos, transferências e treino de deambulação," 2013.

[14] C. Pierdevară, "Andarilho instrumentado para reabilitação e assistência," Ph.D. dissertation, Universidade de Coimbra, Coimbra, 2022.

[15] C. Luz, T. Bush, and X. Shen, "Do Canes or Walkers Make Any Difference? NonUse and Fall Injuries," *The Gerontologist*, vol. 57, no. 2, pp. 211–218, Apr. 2017.

[16] "Standard Walkers | Hemi Walkers - Walkers for Elderly." [Online]. Available: https://www.vitalitymedical.com/standard-walker.html

[17] "Articulated Walker." [Online]. Available: https://farmacianovadamaia.pt/en/mobility/10468-articulated-walker.html

[18] "2 Wheel Folding Walker (Adult & Junior) with Wheels • Walkers • HMEBC." [Online]. Available: https://www.hmebc.com/products/2-wheel-folding-walker-adult/

[19] "Adjustable Three Wheeled Walker." [Online]. Available: https://www.welcomemobility.co.uk/adjustable-three-wheeled-walker.html

[20] "4 wheels walker." [Online]. Available: https://www.gimaitaly.com/prodotti.asp?sku=43130&dept_selected=24&dept_id=244

[21] C. T. Valadao, F. Loterio, V. Cardoso, T. Bastos, A. Frizera-Neto, and R. Carelli, "Robotics as a Tool for Physiotherapy and Rehabilitation Sessions**Authors acknowledge the financial support from FAPES, CAPES and CNPq." *IFAC-PapersOnLine*, vol. 48, no. 19, pp. 148–153, Jan. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S240589631502649X

[22] M. M. Martins, C. P. Santos, A. Frizera-Neto, and R. Ceres, "Assistive mobility devices focusing on Smart Walkers: Classification and review," *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 548–562, Apr. 2012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0921889011002181

[23] W. H. Organization, "Global status report on road safety 2023," World Health Organization, Geneva, Tech. Rep., 2023.

[24] "SAE Levels of Driving Automation™ Refined for Clarity and International Audience." [Online]. Available: https://www.sae.org/site/blog/sae-j3016-update

[25] "Jetson Nano Developer Kit." [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[26] "Get Started With Jetson Nano Developer Kit." [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit

[27] "jetson-inference/docs/aux-streaming.md at master · dusty-nv/jetson-inference." [Online]. Available: https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-streaming.md

[28] "Google Colaboratory." [Online]. Available: https://colab.research.google.com/

[29] "Netron." [Online]. Available: https://netron.app/

[30] "PyTorch." [Online]. Available: https://pytorch.org/

[31] "TensorFlow Lite for Microcontrollers." [Online]. Available: https://www.tensorflow.org/lite/microcontrollers

[32] "Transforming and augmenting images — Torchvision 0.17 documentation." [Online]. Available: https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.Normalize

[33] “Python: package jetson.inference.” [Online]. Available: https://rawgit.com/dusty-nv/jetson-inference/master/docs/html/python/jetson.inference.html#imageNet

Appendix A

Dataset acquisition code

Complete Arduino code used on the ESP32-CAM for dataset acquisition:

```
#include "esp_camera.h"
#include "Arduino.h"
#include "FS.h"                // SD Card ESP32
#include "SD_MMC.h"            // SD Card ESP32
#include "soc/soc.h"           // Disable brownour problems
#include "soc/rtc_cntl_reg.h"  // Disable brownour problems
#include "driver/rtc_io.h"
#include <EEPROM.h>            // read and write from flash memory
#include "PinDefinitions.h"

int pictureNumber = 0;
const int botao = 3;
const int led = 4;
int buttonState = 0;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(botao, INPUT);

  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

  Serial.begin(115200);

  camera_config_t config;
```

```
25    config.ledc_channel = LEDC_CHANNEL_0;
26    config.ledc_timer = LEDC_TIMER_0;
27    config.pin_d0 = Y2_GPIO_NUM;
28    config.pin_d1 = Y3_GPIO_NUM;
29    config.pin_d2 = Y4_GPIO_NUM;
30    config.pin_d3 = Y5_GPIO_NUM;
31    config.pin_d4 = Y6_GPIO_NUM;
32    config.pin_d5 = Y7_GPIO_NUM;
33    config.pin_d6 = Y8_GPIO_NUM;
34    config.pin_d7 = Y9_GPIO_NUM;
35    config.pin_xclk = XCLK_GPIO_NUM;
36    config.pin_pclk = PCLK_GPIO_NUM;
37    config.pin_vsync = VSYNC_GPIO_NUM;
38    config.pin_href = HREF_GPIO_NUM;
39    config.pin_sscb_sda = SIOD_GPIO_NUM;
40    config.pin_sscb_scl = SIOC_GPIO_NUM;
41    config.pin_pwdn = PWDN_GPIO_NUM;
42    config.pin_reset = RESET_GPIO_NUM;
43    config.xclk_freq_hz = 20000000;
44    config.pixel_format = PIXFORMAT_JPEG;
45
46    if(psramFound()){
47      config.frame_size = FRAMESIZE_UXGA; // FRAMESIZE_ + QVGA|CIF|VGA|SVGA|
       XGA|SXGA|UXGA
48      config.jpeg_quality = 10;
49      config.fb_count = 2;
50    } else {
51      config.frame_size = FRAMESIZE_SVGA;
52      config.jpeg_quality = 12;
53      config.fb_count = 1;
54    }
55
56    // Init Camera
57    esp_err_t err = esp_camera_init(&config);
58    if (err != ESP_OK) {
59      Serial.printf("Camera init failed with error 0x%x", err);
60      return;
61    }
62
```

```
63  //Serial.println("Starting SD Card");
64  if(!SD_MMC.begin()){
65    Serial.println("SD Card Mount Failed");
66    return;
67  }
68
69  uint8_t cardType = SD_MMC.cardType();
70  if(cardType == CARD_NONE){
71    Serial.println("No SD Card attached");
72    return;
73  }
74
75  camera_fb_t * fb = NULL;
76  // Take Picture with Camera
77  fb = esp_camera_fb_get();
78  if(!fb) {
79    Serial.println("Camera capture failed");
80    return;
81  }
82  // initialize EEPROM with predefined size
83  EEPROM.begin(EEPROM_SIZE);
84  pictureNumber = EEPROM.read(0) + 1;
85
86  // Path where new picture will be saved in SD Card
87  String path = "/picture" + String(pictureNumber) +".jpg";
88
89  fs::FS &fs = SD_MMC;
90  Serial.printf("Picture file name: %s\n", path.c_str());
91
92  File file = fs.open(path.c_str(), FILE_WRITE);
93  if(!file){
94    Serial.println("Failed to open file in writing mode");
95  }
96  else {
97    file.write(fb->buf, fb->len); // payload (image), payload length
98    Serial.printf("Saved file to path: %s\n", path.c_str());
99    EEPROM.write(0, pictureNumber);
100   EEPROM.commit();
101 }
```

```
102   file.close();
103   esp_camera_fb_return(fb);
104
105   // Turns off the ESP32-CAM white on-board LED (flash) connected to GPIO 4
106   pinMode(4, OUTPUT);
107   digitalWrite(4, LOW);
108   rtc_gpio_hold_en(GPIO_NUM_4);
109 }
110
111 void loop() {
112 }
```

Listing A.1: Dataset acquisition.

Appendix B

Neural Network Architecture

The CNN architecture used is represented in the figure below, created with Netron [29].

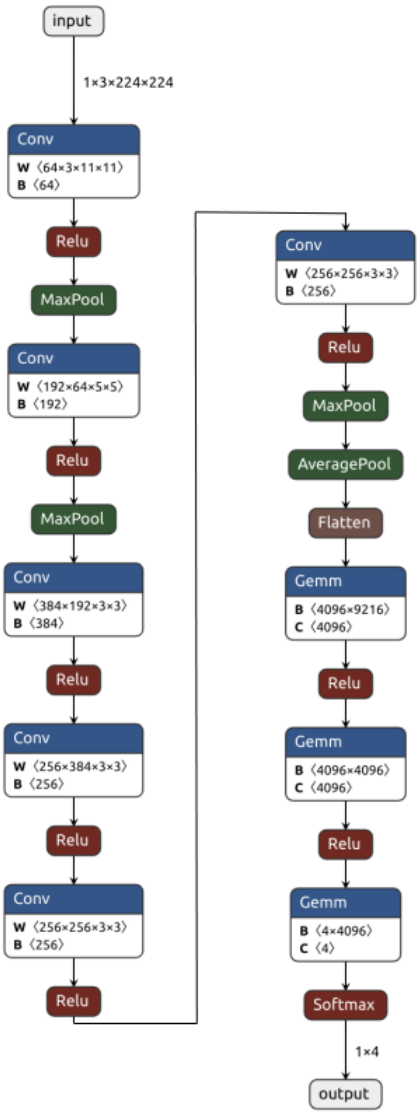


Figure B.1: Model architecture.

Appendix C

Directory structure

Mandatory structure for the model to properly access the necessary files:

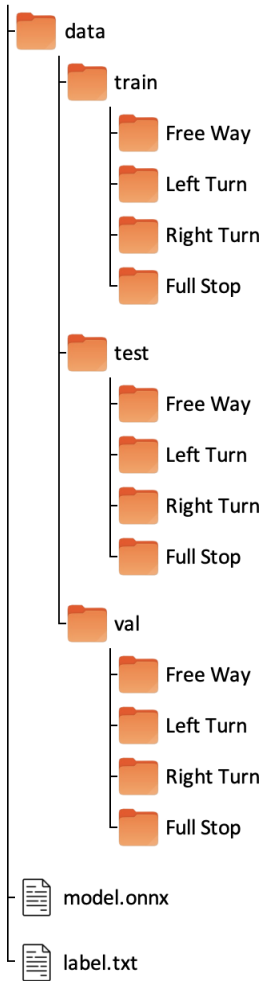


Figure C.1: Directory structure.

Appendix D

Add a softmax layer to the model

The following script ensures the presence of a final softmax layer, serving as a decision layer. This configuration allows the model to return valid probabilities after each inference, facilitating interpretation by the scripts of the container.

```
modelo = nn.Sequential(
  modelo,
  nn.Softmax(dim=1)  # Apply softmax along the dimension of
   classes
)
```

Listing D.1: Softmax Layer.

Appendix E

Mounting Swap

The command lines below create a 4GB swap space to prevent freezing during model training or testing. This swap memory will be deleted each time the Jetson Nano is rebooted.

```
sudo systemctl disable nvzramconfig
sudo fallocate -l 4G /mnt/4GB.swap
sudo mkswap /mnt/4GB.swap
sudo swapon /mnt/4GB.swap
```

The following terminal command checks the memory usage of the Jetson Nano:

```
free -h
```