

FEDERAL UNIVERSITY OF TECHNOLOGY – PARANÁ

**FLÁVIO SHIGUEO MIAMOTO
JOÃO PEDRO ZANLORENSI CARDOSO**

ZCART: A SMART CART PROTOTYPE

CURITIBA

2022

**FLÁVIO SHIGUEO MIAMOTO
JOÃO PEDRO ZANLORENSI CARDOSO**

ZCART: A SMART CART PROTOTYPE

zCart: Um protótipo de Smart Cart

Bachelor Thesis presented as a partial requirement for obtaining the title of Bachelor in Electronics Engineering of the Undergraduate Program in Electronics Engineering of Federal University of Technology – Paraná

Advisor: André Eugênio Lazzaretti, Ph.D.

CURITIBA

2022



[4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/)

Esta licença permite remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es) e que licenciem as novas criações sob termos idênticos. Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

**FLÁVIO SHIGUEO MIAMOTO
JOÃO PEDRO ZANLORENSI CARDOSO**

ZCART: A SMART CART PROTOTYPE

Trabalho de Conclusão de Curso de Graduação
apresentado como requisito para obtenção do
título de Bacharel em Engenharia Eletrônica
do Curso de Bacharelado em Engenharia
Eletrônica da Universidade Tecnológica Federal
do Paraná.

Data de aprovação: 14/Dezembro/2022

André Eugênio Lazzaretti
Doutor
Universidade Tecnológica Federal do Paraná

Gustavo Benvenuto Borba
Doutor
Universidade Tecnológica Federal do Paraná

Rafael Eleodoro de Góes
Doutor
Universidade Tecnológica Federal do Paraná

**CURITIBA
2022**

We would like to offer this thesis to our families
and loved ones; to the supportive and
knowledgeable Professors and mentors we had
over the course of our studies; and to other
students and enthusiasts who pursue learning
with great purpose, honesty and courage.

ACKNOWLEDGEMENTS

We want to express our enormous gratitude to our families and loved ones for standing with us and providing us with the conditions needed to develop our studies. A huge thank you to the Federal University of Technology, for having provided us with the environment, the materials and the lessons that allowed us, students, to continue growing and developing ourselves to the highest standards.

The only thing we're allowed to believe is that
we won't regret the choice we made.

- Hajime Isayama

It's not the honors and the prizes and the fancy
outsides of life which ultimately nourish our
souls. It's the knowing that we can be trusted,
that we never have to fear the truth, that the
bedrock of our very being is good stuff.

- Fred Rogers

ABSTRACT

The recent advancements in artificial intelligence have already impacted many aspects of modern life but have yet to impact one important aspect of the consumer experience in a meaningful way: shopping in physical stores. Tech giants such as Amazon have recently deployed the so called *smart carts* to their physical retail stores, allowing customers to have an improved shopping experience, including better product information and a seamless checkout process. In that regard, this work analyses the current market and describes the development of a prototype that achieves similar functionality to the ones currently available, using the same building blocks such as computer vision and sensor data. A *Deep Learning* model was developed for product detection and deployed to a single board computer capable of running inferences at 4 FPS with an average precision higher than 80%. Finally, this work discusses the challenges and practical constraints of developing such a prototype and also presents suggestions for future work to improve the solution into a commercial product.

Keywords: artificial intelligence; deep learning; smart devices; sensors; object detection.

RESUMO

Os recentes avanços em inteligência artificial já impactaram diversos aspectos da vida moderna mas ainda não atingiram um aspecto importante da experiência dos consumidores: compras em lojas físicas. Gigantes da tecnologia como a Amazon lançaram recentemente os chamados *carrinhos inteligentes* (smart carts) em suas lojas físicas, proporcionando aos consumidores uma melhor experiência de compra, com mais informações sobre os produtos e um processo de pagamento rápido e prático. Neste sentido, o presente trabalho analisa o contexto atual do mercado e descreve o desenvolvimento de um protótipo que entrega funcionalidades similares aos produtos disponíveis no mercado utilizando as mesmas bases tecnológicas de visão computacional e sensores. Um modelo de aprendizado profundo (Deep Learning) foi desenvolvido para a detecção de produtos e implantado em um Single Board Computer capaz de executar inferências em aproximadamente 4 Quadros Por Segundo com uma precisão média acima dos 80%. Finalmente, o trabalho discute os desafios e restrições práticas do desenvolvimento do protótipo e prepara o caminho para trabalhos futuros que podem levar o desenvolvimento até um produto comercial.

Palavras-chave: inteligência artificial; aprendizado profundo; smart devices; sensores; detecção de objetos.

LIST OF FIGURES

Figure 1 – Amazon Echo 4th Generation smart speaker promotional material . . .	16
Figure 2 – Diagram showing the steps of an interaction with an Alexa Skill	17
Figure 3 – US Smart Speaker Penetration from 2017 to 2022	17
Figure 4 – Localized promotional material for the Echo Show 10 targeting Brazilian customers	18
Figure 5 – Top five customer pain points in retail stores	19
Figure 6 – Caper Cart at a retail store	20
Figure 7 – Caper Cart user interface	20
Figure 8 – Caper Cart 3 promotional material	21
Figure 9 – Amazon Dash Cart	22
Figure 10 – Smart Cart Nextop® deployed to a Brazilian supermarket	23
Figure 11 – Smart Cart Nextop® user interface with a payment terminal	23
Figure 12 – Smart Cart Nextop® promotional material targetting supermarket owners	24
Figure 13 – The Perceptron Neuron	26
Figure 14 – Sigmoid Function	27
Figure 15 – Three Layered Neural Network	28
Figure 16 – Bounding Boxes	30
Figure 17 – Activation Functions in Object Detection	31
Figure 18 – NMS Applied to Object Detection	32
Figure 19 – TPU block diagram	33
Figure 20 – Coral Dev Board Micro	33
Figure 21 – Typical Strain Gauge construction	34
Figure 22 – Wheatstone Bridge circuit using four strain gauges	34
Figure 23 – HBM Z6 commercial load cell	35
Figure 24 – High level architecture of zCart	37
Figure 25 – End-to-end sequence diagram for a product addition	38
Figure 26 – User App Interface display a single product	39
Figure 27 – User App and Cart Service interactions	40
Figure 28 – Cart Service architecture	40
Figure 29 – Product Recognizer Components	42

Figure 30 – HX711 Pinout for the SOP-16L package	43
Figure 31 – HX711 breakout board and load cell	43
Figure 32 – Typical HX711 circuit	44
Figure 33 – Assembly used during development, including the Raspberry Pi, HX711 and load cell assembly	45
Figure 34 – Microsoft LifeCam Cinema Webcam used	45
Figure 35 – Thread communication for the Product Recognizer Appplication	47
Figure 36 – Illustration of the frame diff'ing scheme	48
Figure 37 – Activity diagram for the Product Recognizer thread	49
Figure 38 – Collecting Data in Edge Impulse	51
Figure 39 – Labelling Queue in Edge Impulse	51
Figure 40 – Google Colab - Overview of Colaboratory Features	53
Figure 41 – YOLOv5 vs EfficientDet - Performance Comparison	54
Figure 42 – Image Preprocessing Strategies	55
Figure 43 – Overall mechanical assembly including the LCD and Camera	57
Figure 44 – Top view of the assembly showing the false bottom	57
Figure 45 – False bottom structure with the load cell and Raspberry Pi board in be- tween	58
Figure 46 – Single-Label Classification Example 1	59
Figure 47 – Single-Label Classification Example 1	59
Figure 48 – Multi-Label Classification Example 1	60
Figure 49 – Multi-Label Classification Example 1	60
Figure 50 – Loss Chart for the EfficientDet-D0 Network	62
Figure 51 – Loss Chart for the EfficientDet-D1 Network	62
Figure 52 – Loss Chart for the EfficientDet-D2 Network	63
Figure 53 – Frame from camera feed with FPS count	63
Figure 54 – Power consumption measurement using a wattmeter	64
Figure 55 – Product listing and subtotal	73
Figure 56 – Item addition notification	73
Figure 57 – Item removal notification	74
Figure 58 – Pre-checkout confirmation popup	74
Figure 59 – Post checkout screen	75

Figure 60 – Frontal view of the prototype	77
Figure 61 – Top view displaying the products in the cart	77
Figure 62 – Frontal view of the prototype displaying the video camera feed	78
Figure 63 – Frontal view of the prototype displaying the video camera feed	78
Figure 64 – Popup confirmation before finishing the purchase	79
Figure 65 – Checkout confirmation screen	79
Figure 66 – Product addition notification	80
Figure 67 – Product removal notification	80

LIST OF TABLES

Table 1 – Cart Service API endpoints	41
Table 2 – Image Sets Created for Training, Validation and Testing	56
Table 3 – Models Trained	56
Table 4 – Test Evaluation Metrics for the Different Strategies and Models Applied for Inference	61
Table 5 – Inference Performance Metrics with and without the TPU	63
Table 6 – Items used on the prototype and their approximate retail cost in Brazil as of October 2022	65

LIST OF SOURCE CODE

Source Code 1 – Example response for the GET /cart/:cartId endpoint using JSON	41
Source Code 2 – Loop of the Main Thread. Some details were omitted for the sake of brevity	47
Source Code 3 – Product Recognizer thread logic	50
Source Code 4 – Bounding boxes coordinates file exported from Edge Impulse . .	52
Source Code 5 – CSV format for specifying the Train, Test and Validation image sets for training models using the TensorFlow's Model Maker API for Object Detection	55

LIST OF ABBREVIATIONS AND ACRONYMS

Pseudo-Acronyms

ABRAS	Associação Brasileira de Supermercados
ADC	Analog-to-Digital Converter
ANN	Artificial Neural Network
AP	Average Precision
API	Application Programming Interface
CEO	Chief Executive Officer
CNN	Convolutional Neural Networks
CSV	Comma Separated Values
DNN	Deep Neural Networks
DSA	Domain Specific Architecture
FPS	Frames Per Second
GDP	Gross Domestic Product
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IoU	Intersection over Union
JSON	JavaScript Object Notation
NMS	Non-Maximum Suppression
PoC	Proof-of-Concept
RFC	Request For Comments
SBC	Single Board Computer
SGD	Stochastic Gradient Descent
SQL	Structured Query Language
TCP	Transmission Control Protocol
TPU	Tensor Processing Unit

CONTENTS

1	INTRODUCTION	16
1.1	Motivation	16
1.2	Current scenario	19
1.2.1	Caper Cart	19
1.2.2	Amazon Dash Cart	21
1.2.3	Nextop	22
1.3	Objectives	25
1.3.1	Main objective	25
1.3.2	Specific objectives	25
2	BACKGROUND	26
2.1	Neural Networks	26
2.1.1	Neurons	26
2.1.2	Learning Algorithms	27
2.2	Deep Learning	28
2.2.1	Transfer Learning	29
2.3	Object Detection	29
2.4	Tensor Processing Unit	32
2.5	Strain Gauge	33
2.6	Load Cell	35
3	DEVELOPMENT	36
3.1	Design	36
3.1.1	Architectural Guidelines	38
3.2	User App	39
3.3	Cart Service	40
3.4	Product Recognizer	42
3.4.1	Weight Sensor	42
3.4.2	Camera	45
3.4.3	Application	46
3.4.4	Product Recognizer Thread	47
3.4.5	Object Detection Model	49

3.5	Mechanical Assembly	56
4	RESULTS	59
4.1	Model Accuracy	59
4.2	Power Consumption	64
4.3	Cost	65
4.4	Challenges and future work	66
4.4.1	Extending the model for new products	66
4.4.2	Deploying updates	66
4.4.3	Batteries and charging	66
4.4.4	Loss Prevention	67
4.4.5	Improving accuracy and reliability	67
5	CONCLUSIONS	68
	BIBLIOGRAPHY	69
	APPENDIX A USER APP SCREENSHOTS	73
	APPENDIX B PROTOTYPE PHOTOS	77

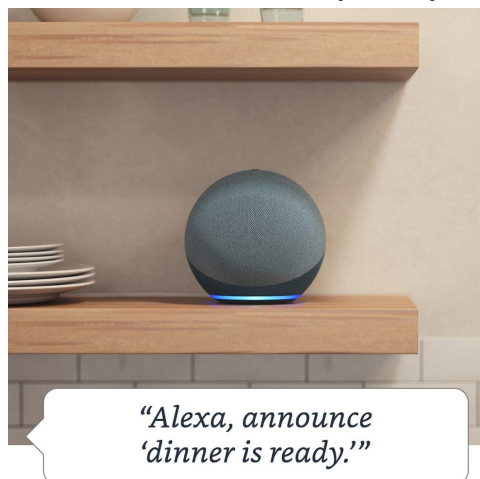
1 INTRODUCTION

1.1 Motivation

With the advancement of high speed mobile networks and smartphone penetration, customer demands are on an ever increasing trajectory for more personalized and digital experiences. Companies worldwide are fighting for customer attention in the digital era by developing products and services that bring state-of-the-art technologies to the masses in the so called smart devices and systems (SHAFIQUE *et al.*, 2020).

As an example of such advancements, smart speakers such as the Amazon Echo (GAO *et al.*, 2018), shown on Figure 1, include the latest and greatest in terms of Natural Language Processing and Deep Learning (YOUNG *et al.*, 2018), allowing customers to interact with the product in an conversational manner that was considered to be science fiction material until a couple of years ago.

Figure 1 – Amazon Echo 4th Gen Smart Speaker promotional material



Source: Amazon (2022).

This device is a gem! When I'm busy in the kitchen, for example, and can't get to a computer to find info or music to play, Alexa would be there to listen and do what I ask.

Customer review from Gao *et al.* (2018)

Alongside the devices themselves, entirely new markets have emerged such as the third-party software extensions called *Alexa Skills* (AMAZON, 2022b).

These skills function much like mobile phone apps, extending and enhancing the functionality of the device and can be sold to end users.

Developers can then easily leverage the highly advanced machine learning models through Application Programming Interface (API) and focus exclusively on their business logic, as shown on Figure 2.

This can be valuable as Machine Learning models are considered to be expensive to develop and maintain, with some sources mentioning a minimum expenditure of US\$ 60.000 over a five year period considering both tasks (PHDATA, 2021).

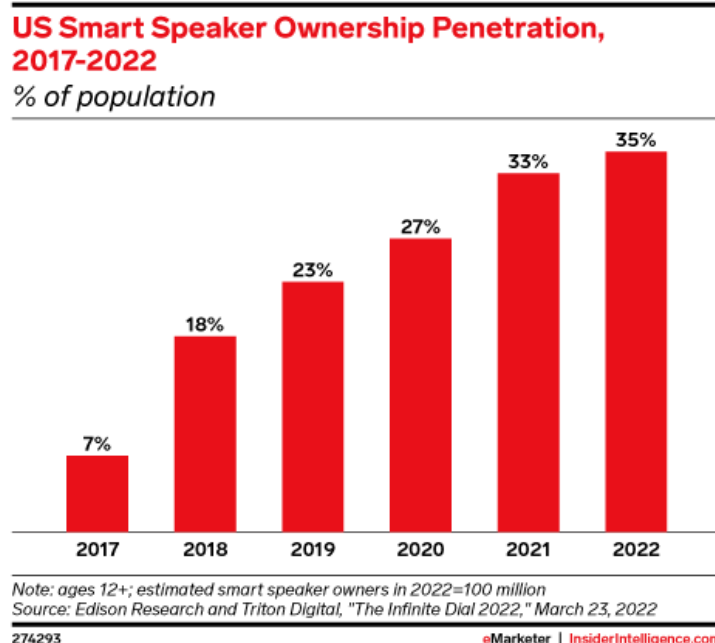
Figure 2 – Diagram showing the steps of an interaction with an Alexa Skill



Source: Amazon (2022b).

More impressively, such technological advancements have been able to reach a considerable amount of households in a short period of time in developed countries like the United States, as shown on Figure 3.

Figure 3 – US Smart Speaker Penetration from 2017 to 2022



Source: Intelligence (2022).

On developing countries such as Brazil, these innovations tend to have delayed arrivals due to historical economic barriers but the potential customer base has attracted big tech companies like Amazon, which are able to shorten the arrival delay with their economic power.

In Figure 4, we can see an example of a smart speaker advertisement from Amazon for the Brazilian customer base.

Figure 4 – Localized promotional material for the Echo Show 10 targeting Brazilian customers



Source: Amazon (2022).

According to the research company IDC Brasil, the Brazilian home automation market – in which smart speakers are included – would have reached US\$ 291 million on 2021, an impressive figure that might explain the attractiveness of our market (BRASIL, 2021).

But even with all of these innovations impacting customer behaviors day by day, one important aspect of consumer life still has not had any significant changes in the last couple of years: **shopping on physical stores**.

According to Associação Brasileira de Supermercados (ABRAS) - the Brazilian Supermarket Association - the Brazilian grocery retail sector has reached an impressive total revenue of **R\$ 611 billion** in 2021 - roughly US\$ 117 billion on October 2022 conversion rates - making up 7,03% of the national Gross Domestic Product (GDP). About **28 million** customers visit one of the more than **92.000** stores countrywide on a daily basis (ABRAS, 2022).

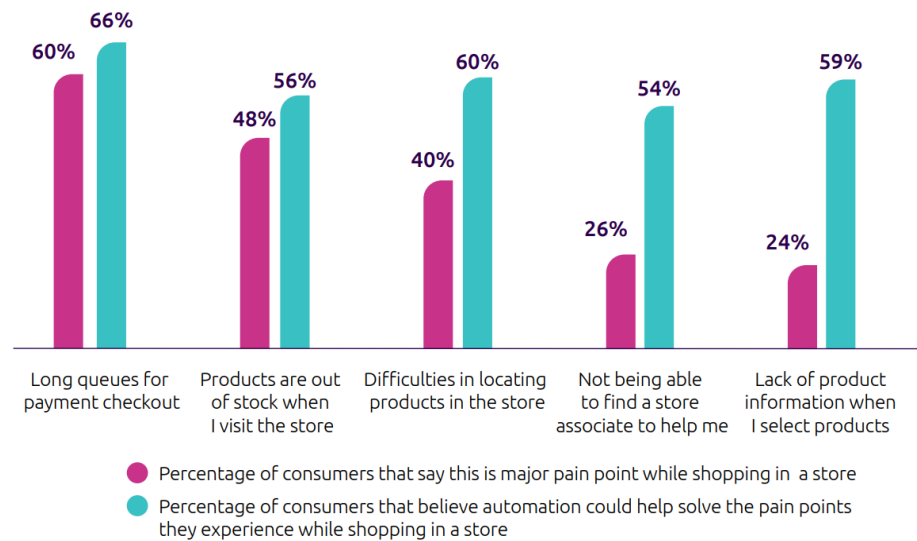
Despite all the technological advancement seen over the last few years and the economic relevance of such sector, retail grocery shopping still exhibits the same limitations found a decade ago. In a survey conducted in 2019, Capgemini has found out five key pain problems related to physical stores in general (CAPGEMINI, 2020):

1. Long queues for payment checkout
2. Out of stock products
3. Difficulties in locating products in the store
4. Not being able to find a store associate to help
5. Lack of product information when I select products

The survey results can be seen in more detail in Figure 5.

Figure 5 – Top five customer pain points in retail stores

Top five consumer pain points in stores and whether consumers believe automation can help solve them



Source: Capgemini Research Institute, Automation in Retail Stores Research, Consumer Survey, October 2019, N=5,110 consumers.

Source: (CAPGEMINI, 2020).

More interestingly, the survey points out that at least half of the survey respondents believe that all of the five pain points can be solved through **automation**. Even in light of the recent pandemic scenario, innovations that increase automation such as e-commerce platforms had their adoption increased in 2020 but 2021 showed a trend of consumers shifting back to their pre-pandemic behavior, favoring physical retail stores (KANTAR, 2022).

It is in this scenario of customer pain and enormous market potential that this thesis will explore a technological solution to improve customer experience and increase sales, namely the **smart shopping cart**.

1.2 Current scenario

In this next section, we'll explore some of the existing solutions and the user experience provided by them.

1.2.1 Caper Cart

Developed by the Caper¹ company, the Caper Cart was the world's first AI-powered smart cart (CAPER, 2020)

¹ <https://caper.ai>

The first version was launched in 2017 and offered grocer's the great advantage of not requiring any infrastructure overhaul for deployment. In Figure 6, we can see the cart deployed at American retail store.

Figure 6 – Caper Cart at a retail store



Source: Caper (2020).

For end users, it offered visual product recognition and a payment terminal, allowing them to avoid the dreaded queues by the end of their shopping session. Additionally, customers were able to search products, get discounts and locate items more easily with the help of the interactive user interface provided by the cart that can be seen on Figure 7.

Figure 7 – Caper Cart user interface

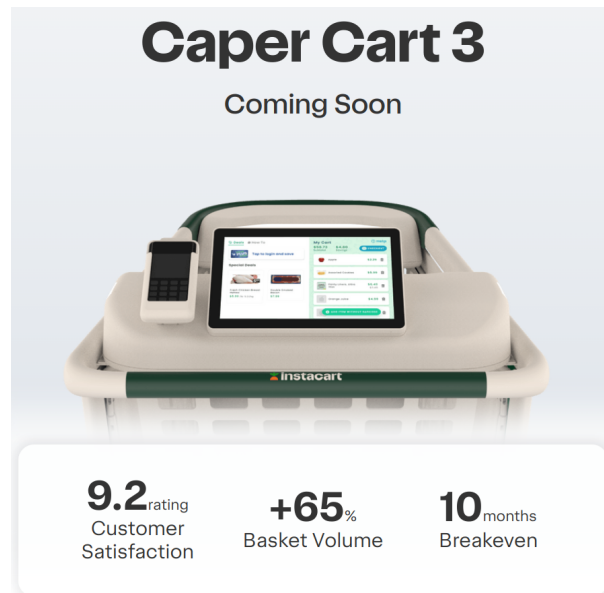


Source: Caper (2020).

Although Caper does not publicize the cost of each cart, it is estimated that each unit costs between **US\$ 5,000 and 10,000** (POST, 2021).

Acquired by Instacart² in 2021 for US\$ 350 million, Caper is developing in 2022 the third version of its Smart Shopping Cart, advertising an increase of **65% in the basket volume** and a **10 month** break even period, as shown on Figure 8.

Figure 8 – Caper Cart 3 promotional material



Source: Caper (2022).

1.2.2 Amazon Dash Cart

Available at the Amazon Fresh³ retail chain, the Amazon Dash Cart is the company's first smart cart available to end users and is shown in Figure 9.

According to Amazon, it uses computer vision and sensors to allow customers to simply add items to their cart like they usually would. The cart accounts all the items present in the cart, displaying a list which includes their prices and subtotal. By the end of their item selection, customers can check-out automatically without having to go through queues, solving the biggest customer pain point pointed out by Capgemini (2020).

Looking to make grocery trips quicker? With the Amazon Dash Cart you can skip the checkout line and roll out to your car when you are done.

The Dash cart uses a combination of computer vision algorithms and sensor fusion to help identify items placed in the cart - simply grab an item, scan it on one of the Dash Cart cameras, and place it in the cart like you normally would.

Amazon (2022)

² <https://instacart.com>

³ <https://www.amazon.com/fmc/m/30003175?almBrandId=QW1hem9uIEZyZXNo>

Figure 9 – Amazon Dash Cart



Source: Amazon (2022).

In addition to the item accounting capabilities, the user interface provided by the Cart also allows customers to search for the location of items in the store and see more information about them, improving the customer experience.

One of its particularities is that it requires the download and usage of an mobile phone app for using the cart, something not required by Caper's Cart.

As of October 2022, the Amazon Dash Cart is exclusively available at the Amazon Fresh chain and therefore no commercial information regarding cost per unit is available.

1.2.3 Nextop

Founded in 1997, Nextop⁴ is a Brazilian company that develops products with a focus on the grocery stores market, with an emphasis on loss prevention.

Their smart cart offering, shown in Figure 10, is the first deployed in Brazil and Latin America according to the company and was initially rolled out to the Enxuto supermarket chain in 2022 (TOTAL, 2022).

⁴ <https://nextop.com.br>

Figure 10 – Smart Cart Nextop® deployed to a Brazilian supermarket



Source: Nextop (2022).

In contrast to the carts developed by Amazon and Caper, Nextop's cart requires an additional step of scanning the product using the integrated barcode reader, shown in Figure 11, before adding it to the cart. With that, the Nextop advertises for a *triple validation* system, using the cameras, sensors and the barcode scanner to prevent losses (NEXTOP, 2022).

Figure 11 – Smart Cart Nextop® user interface with a payment terminal



Source: Total (2022).

Although loss prevention is an important selling point for the Brazilian market, the usage of the barcode scanner creates, in our opinion, a worse customer experience, becoming a *mobile*

checkout station. Also, the product does not include additional features such as product location search and item details.

Figure 12 shows an advertisement, in Portuguese, that lists the following benefits:

- Innovative supermarkets sell 20% more
- Loss prevention
- Triple validation
- Freedom and agility
- New self checkout using artificial intelligence
- No queues

Figure 12 – Smart Cart Nextop® promotional material targeting supermarket owners. It advertises for improved sales, loss prevention and reduced queues.



Source: Nextop (2022).

Offering a solution to the main end user pain point of having to go through long queues, the product also advertises increased sales as a result of the innovative approach and also allows a deeper understanding of the customer journey by collecting analytical data (TOTAL, 2022).

We are offering our customers an innovative and unique shopping experience within Enxuto

With the smart cart, we broke through this barrier and managed to monitor the entire customer's purchase circuit in the physical store. We have moved from the identified ticket era to the end-to-end identified journey

Bruno Bragancini Junior, Chief Executive Officer (CEO) of the Enxuto Group from Total (2022)

According to Nextop's CEO, Juliano Camargo, the company has already invested **R\$ 8,5 millions** - about US\$ 1,63 million on October 2022 - and **4 and half years** of research and development.

Each Smart Cart is estimated to cost around **R\$ 120,000** or around US\$ 23,020 on October 2022 (TOTAL, 2022).

1.3 Objectives

After presenting the problem domain and the current market scenario, in this section we discuss the objectives of this work.

1.3.1 Main objective

Develop a prototype of a smart shopping cart that utilizes computer vision and sensor data for product recognition.

1.3.2 Specific objectives

- Build a mechanical assembly for the prototype
- Develop an interactive user interface for the prototype
- Collect a product dataset for training a deep learning model
- Train a Deep Learning model capable of detecting target products
- Learn the practical challenges of developing a Deep Learning based product
- Understand the economic viability of such a project in the Brazilian context

2 BACKGROUND

In this section, we'll define in greater detail the theory behind some most relevant techniques used in the development of the prototype.

This section can be skipped for readers with familiarity on the topics discussed.

2.1 Neural Networks

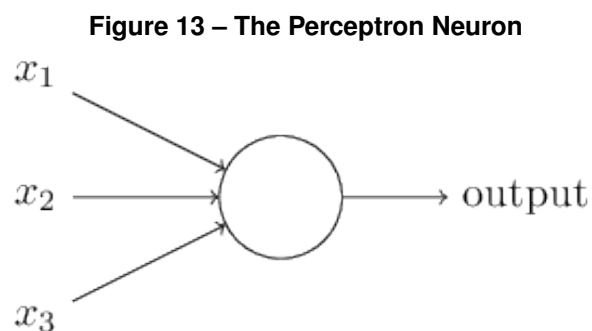
Neural Networks, or Artificial Neural Network (ANN), are networks built out of interconnected decisional Neurons that aim to replicate the behavior of the human brain, enabling computational systems to cluster data and to make predictions (IBM, 2020b).

A Neuron is similar to a Digital Logic Gate, which is capable of producing different outputs depending on the input signals that are sent to it. A Neuron has weights, which are the coefficients that are used for calculating the outputs; and biases, which are the boundaries that represent how prone is an output to fit into a specific category of output.

The main difference between a digital logic gate and a Neuron is that, because neurons are parameterized with weights and biases, we can apply *learning algorithms* to tune (or train) Networks of Neurons using real world or synthetic data (NIELSEN, 2015).

2.1.1 Neurons

One of the most popular types of Neurons is the Perceptron, introduced by Rosenblatt (1958) and is shown in Figure 13. A perceptron takes one or more binary inputs and produces a single binary output.

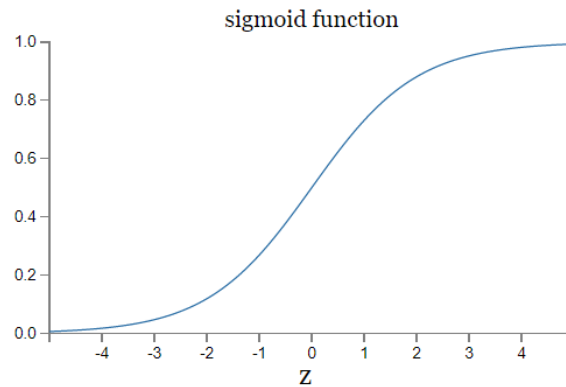


Source: Nielsen (2015).

A Neuron does not necessarily have binary inputs and outputs. In fact, contemporary systems commonly use a different type of Neuron known as the Sigmoid Neuron, which take real numbers as inputs and also produces continuous outputs within the boundaries of the sigmoid curve, shown in Figure 14, instead of discrete zeroes and ones.

This is particularly useful for learning, since in Perceptrons small differences in the weights or biases could yield to different outputs without a full transparency on how close the output would have been to a different one (NIELSEN, 2015).

Figure 14 – Sigmoid Function



Source: Nielsen (2015).

2.1.2 Learning Algorithms

There are three main types of learning algorithms: Supervised, Unsupervised and Reinforcement. Supervised learning is employed when you know what are your expected outputs and use this information to feed (train) your models such that they can start doing that on their own; Unsupervised learning is applied when you do not know what are the expected labels in your data or you do not have them available; and Reinforcement learning is used when algorithms need to replicate specific behaviors depending on the feedback that is provided to them (COURSERA, 2022).

Supervised learning is typically employed for regression and classification tasks; Unsupervised Learning is typically used for clustering raw data into different buckets without necessarily knowing how to do it or having the labelled data available; and Reinforcement learning is often applied in control systems. This work will primarily focus on Supervised Learning, since our proposed project uses Object Detection and is trained based on labeled data samples.

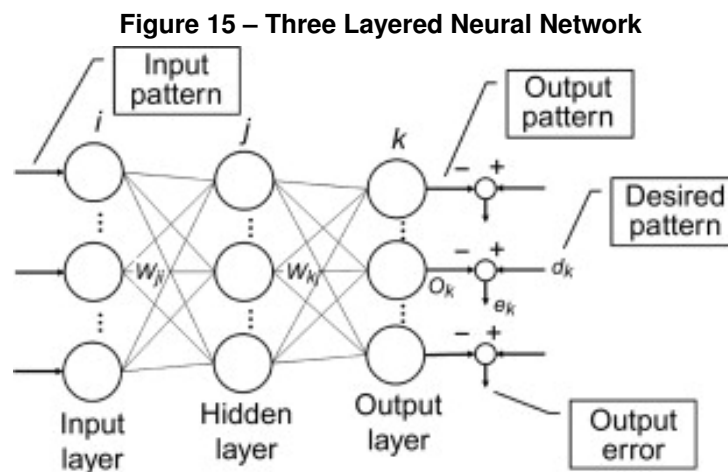
As for the learning algorithms used for training Neural Networks, the standard algorithm is the Stochastic Gradient Descent (SGD). Alike other gradient descent algorithms, the SGD aims to minimize the loss function of a given Neural Network iteratively. In simple terms, it consists of an iterative optimization algorithm defined by an objective function to minimize the error.

The key feature of the SGD is that it does that very efficiently by initializing the weights and biases randomly and then fine tuning it during training by trying to find the higher descents (or the higher derivatives of the error function), reducing the number of necessary iterations, which is particularly important considering that the amount of data that is used to train AI models has increased considerably over the last few years (PRICE *et al.*, 2020).

Finally, in terms of the way how the SGD is computed in Neural Networks, a widely adopted approach for Supervised Learning is Backpropagation. Backpropagation computes the gradients of the final layers of a Neural Network first, and the gradients of the first layers at last, and it reuses partial computations of the gradient from a layer to the other to compute each layer's gradient, making it more efficient than calculating each layer's gradient separately (MCGONAGLE *et al.*, 2022).

2.2 Deep Learning

Deep Learning defines a group of AI algorithms that use advanced learning techniques on the top of ANNs to train models that allow systems to forecast and clusterize data efficiently (IBM, 2020a). Deep Learning algorithms use Neural Networks that have three types of layers: Input Layer, Hidden Layer and Output Layer, shown in Figure 15.



Source: Araki e Omatu (2015).

The Input Layer takes the input data that will be fed into the model for training, x_i , and produces the weighting coefficients w_{ji} . Input Layers typically have the role of encoding the input data into a structure that can be processed by the subsequent hidden layers (PARANJAPE; DUBEY; GOPALAN, 2020).

The w_{ji} coefficients are then used to feed the Hidden Layer of the network, which produces the weighting coefficients w_{kj} . The Hidden Layers of a Deep Learning Neural Network are used for the heavy processing of the encoded input data by applying a series of mathematical operations that ultimately makes it possible to break down the most important features of the data and to produce comprehensive outputs to the decisional layers of the network (DEEPAI, 2022).

2.2.1 Transfer Learning

Another key technique applied by developers when training Deep Learning models is *Transfer Learning*. It consists of reusing pre-trained networks, which were tuned in other datasets – usually big and somehow related to the one that you are going to run your inferences on – for training custom models.

Transfer Learning works by freezing the weights and biases present in specific layers of a pre-trained Network – usually being the hidden layers, which are the feature extractors – when training customized models. This way, only some layers of the Network – usually the last layers, which are the deciders – effectively have to be tuned.

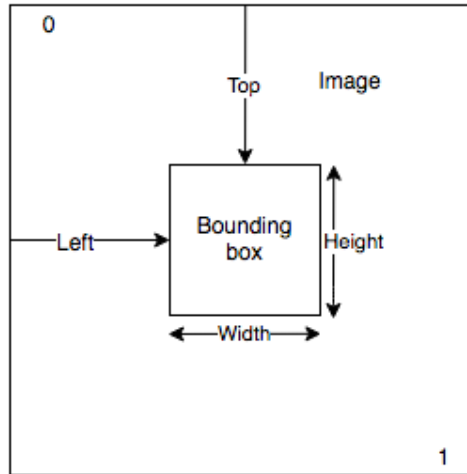
The biggest advantages of Transfer Learning are that, because it allows for training only specific layers of an architecture to get a custom model, the training process is much faster than it would be if the whole network had to be retrained; and additionally less training data is required for achieving a good performing model, since you reuse the work done on the previous training (LI, 2022).

2.3 Object Detection

Images can be processed as number matrices by computers, where each number represents a pixel's color level and each index points to a position in the image; therefore, we can use collections of images as an input to train Deep Learning algorithms and perform classification and object detection tasks on them.

Image Classification is a field of studies that is focused on labeling images as a whole; Object Detection, on the other hand, focuses not only in classifying them, but also in identifying the individual label's coordinates within each image. For the purpose of our project and considering our proposed product's specifications, we will focus on Object Detection, as it needs to be able to recognize multiple objects within an image.

To feed supervised Deep Learning algorithms to detect multiple objects in an image, we need to provide our AI Models with images that have one or more labels, the specification on what labels are there, and their coordinates. To specify the coordinates of an object within an image, we use Bounding Boxes, shown in Figure 16. The Bounding Box also describes the width and height of the object.

Figure 16 – Bounding Boxes

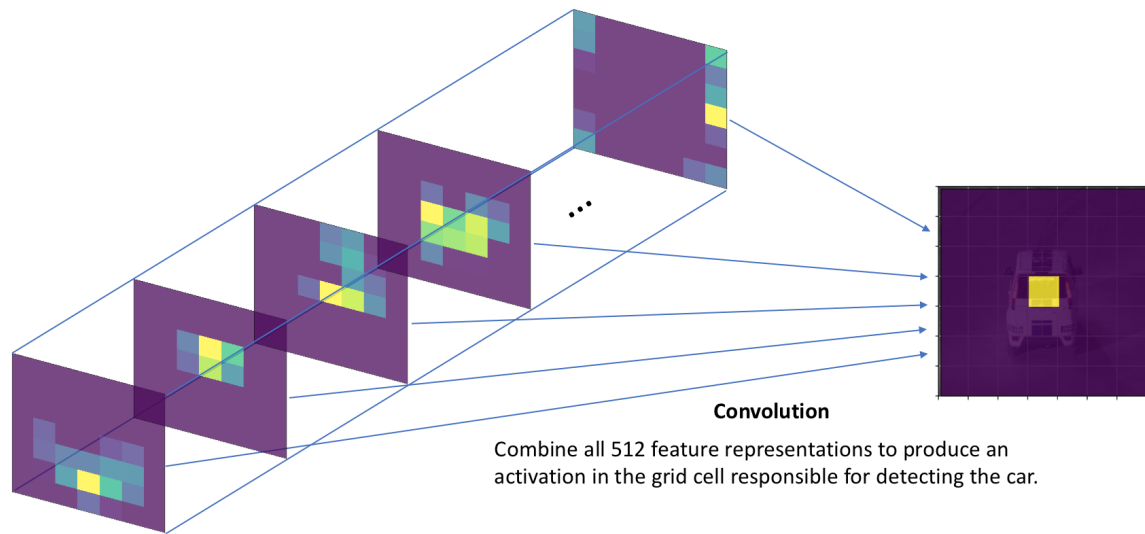
Source: Amazon (2022a).

For our models to perform feature extraction and ultimately be able to classify each label in an image and tell their positions, an important mathematical operation comes to play: the Convolution. Deep Learning algorithms that use Convolutions in their backbones are typically called Convolutional Neural Networks (CNN). These algorithms typically start by creating grid cells, which are delimiters for groups of pixels, around the raw images for determining regions of interest and breaking down concise representations of what are the elements within the image. Convolutions are then applied from the original image against those grids to filter and reduce the dimensionality of the original image and create feature maps (ZHAO *et al.*, 2019).

These feature maps then typically go through the final convolutional stages to calculate Kernels that allows for creating activation functions around the grid cells that contain each of the objects within the pictures (JORDAN, 2018); and those activation functions allow us to determine the Bounding Box coordinates, the confidence score of the inference and the labels of the image themselves.

An example of that process is shown in Figure 17.

Figure 17 – Activation Functions in Object Detection



Source: Jordan (2018).

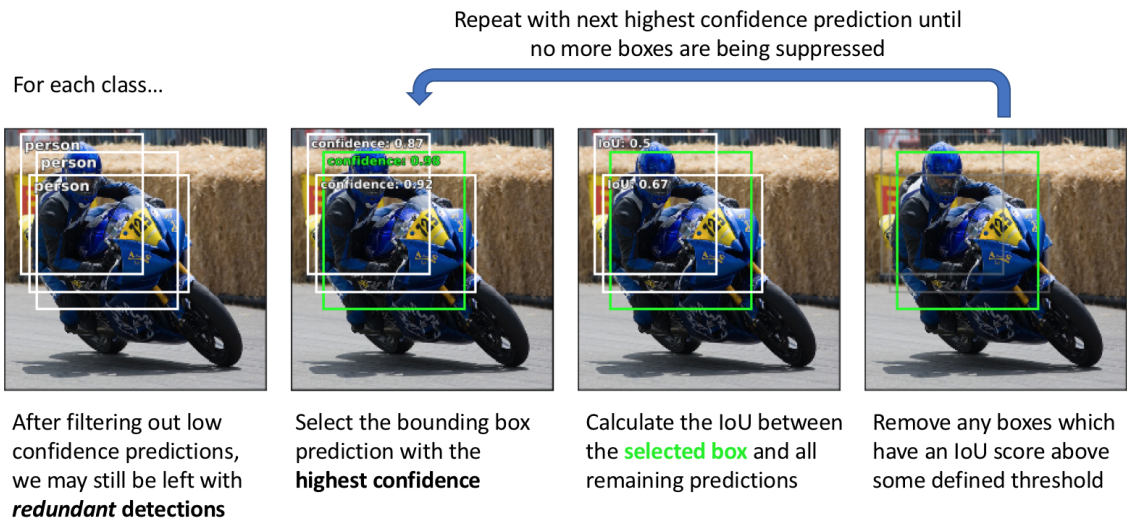
Finally, a technique known as the Non-Maximum Suppression (NMS) is commonly used to filter the regions of interest with the highest probability scores and ultimately get to a single bounding box for each prediction. An example of the technique is shown in Figure 18.

In this sense, two important metrics used for evaluating Object Detection models are the Intersection over Union (IoU), which measures the overlap between the predicted bounding box and the actual bounding box, as shown in Equation 1; and the Average Precision (AP), which measures the percentage of correct predictions made by the object detection model, as shown in Equation 2, where **True Positives** indicates the number of correct predictions and **False Positives** measures the number of incorrect predictions made by the model.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (1)$$

$$AP = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (2)$$

Figure 18 – NMS Applied to Object Detection

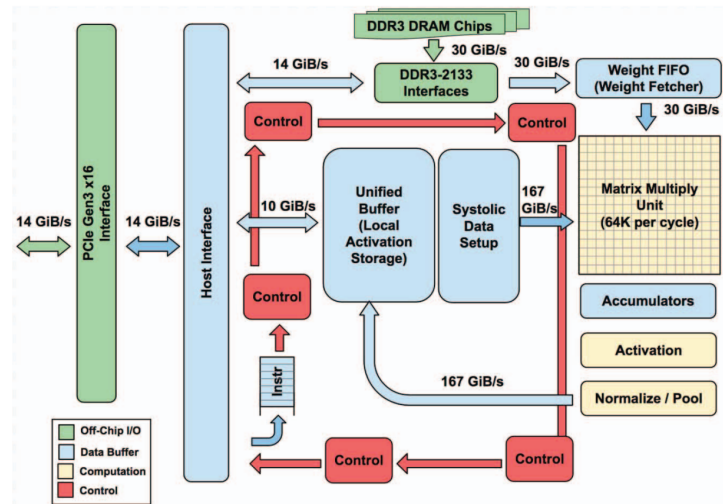


Source: Jordan (2018).

2.4 Tensor Processing Unit

With the increased adoption of Deep Neural Networks (DNN) in various workloads and their specialized and heavy compute nature, Google started the development of a domain specific architecture Domain Specific Architecture (DSA) which resulted in a first generation custom chip, named Tensor Processing Unit Tensor Processing Unit (TPU), deployed to their data centers since 2015 (JOU PPI *et al.*, 2018). The developed TPU had the target of improving the inference phase of DNNs and achieved a performance improvement of **15-30 times** when compared to contemporary hardware of paired or reduced power consumption. A block diagram of the TPU architecture is shown in Figure 19.

Figure 19 – TPU block diagram. The main computation, matrix multiplication, is done by the yellow units.

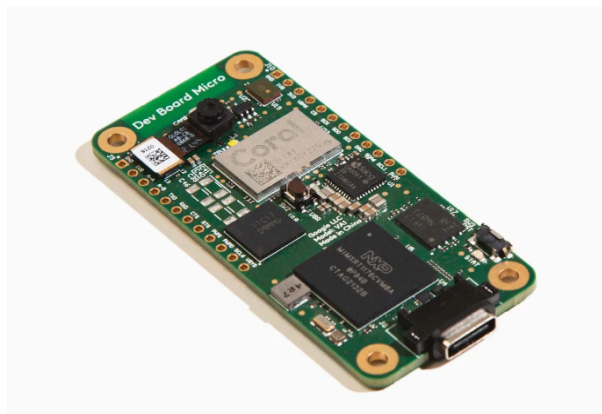


Source: Jouppi *et al.* (2018).

Since the first generation TPU described in Jouppi *et al.* (2018), Google has released the TPU into commercial products made available to third parties. Most notably, the Google Coral¹ initiative offers ready-to-use development boards that embed TPU chips, allowing developers to leverage the improved DNN performance in their applications.

The Coral Dev Board Micro, an example of those ready-to-use boards is shown in Figure 20.

Figure 20 – Coral Dev Board Micro. It includes a microphone, camera and the Coral Edge TPU in a single board package.



Source: Coral.ai (2022).

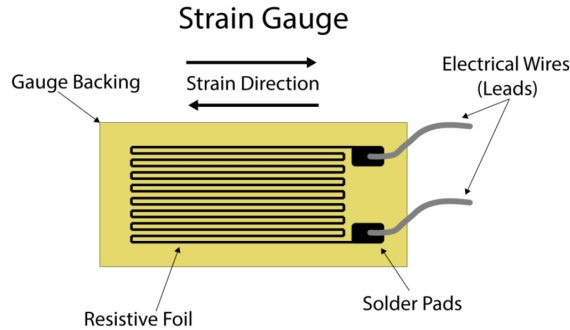
2.5 Strain Gauge

For measuring weight, one of the most common transducers used is the strain gauge. A strain gauge is a device whose measured electrical resistance varies with changes in its ap-

¹ <https://coral.ai>

plied force, as a consequence of the mechanical deformation (ȘTEFĂNESCU, 2011). A typical construction is shown in Figure 21.

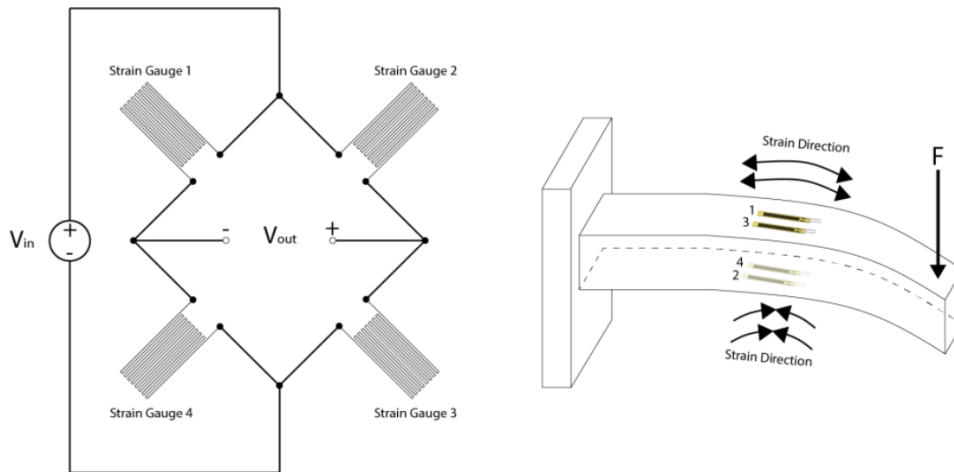
Figure 21 – Typical Strain Gauge construction



Source: Corporation (2020).

In practice, however, the resistance variations observed after applying a mechanical strain are minute and can be difficult to measure. To solve that issue, and to also provide a signal which can be later used as the input of an Analog-to-Digital Converter, the Wheatstone Bridge circuit, shown in Figure 22, can be used (CORPORATION, 2020).

Figure 22 – Wheatstone Bridge circuit using four strain gauges



Source: Corporation (2020).

When no load is applied, the bridge is balanced and the output voltage V_{out} should be zero. If any strain is applied to the gauges, the bridge will become unbalanced, and therefore will result in a non-zero output voltage. Since the voltage variation tends to be small, in the order of millivolts, signal amplification is usually required for pairing the bridge with commercially available ADCs (HOROWITZ; HILL, 2015).

It can be shown that the relation between V_{out} and V_{in} , S , shown on Figure 22, can be calculated as (ȘTEFĂNESCU, 2011):

$$S = \frac{V_{out}}{V_{in}} = k \frac{\Delta l}{l} \quad (3)$$

where k is known as the *gauge factor*, related to the physical construction and materials of the strain gauge, and $\frac{\Delta l}{l}$ is the relative variation of length or strain.

With that, Equation 3 indicates that the output voltage V_{out} will be linearly proportional to the amount of strain applied, providing the desired sensing capability.

2.6 Load Cell

Using strain gauges directly can be difficult since a proper mechanical structure and arrangement is crucial for the sensors to function properly. For that, commercially available *Load Cells* offer ready to use mechanical packages that embed the strain gauges for weight sensing using electronic circuits.

In general, they consist of a spring element onto which the gauges are placed. When load is applied to the cell, the strain gauges will be stressed and therefore provide the desired sensing (HBM, 2022).

An example of a commercial Load Cell is shown in Figure 23

Figure 23 – HBM Z6 commercial load cell



Source: HBM (2022).

3 DEVELOPMENT

In this section, we describe the development of the *zCart* smart cart prototype. All the source code related to the prototype is available at a GitHub repository¹

3.1 Design

As a first step in developing our prototype, a set of high level goals was defined to guide the initial technical design:

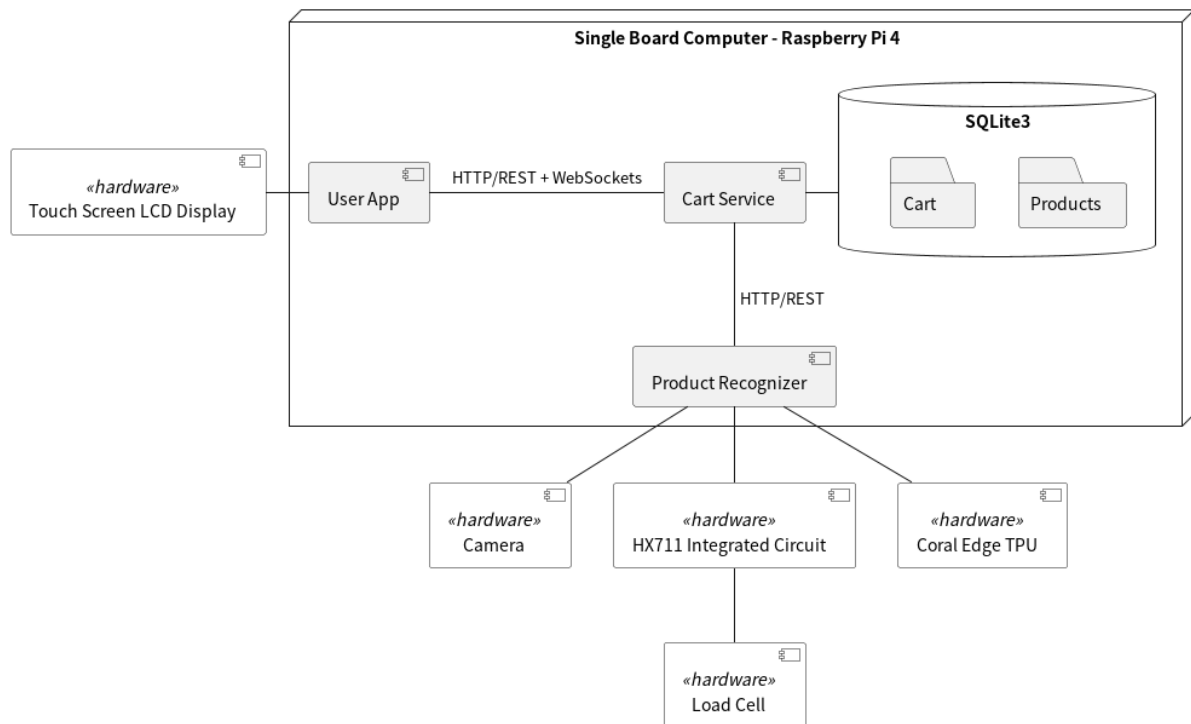
1. Handle user interactions and give visual feedback
2. Store the current set of products present in the cart and their respective information
3. Recognize the addition or removal of products, including quantities.

With those goals in mind, the high level architecture of the prototype was designed and is shown in Figure 24.

For each goal, a dedicated software application will be used and those applications will communicate using Transmission Control Protocol (TCP) network sockets (KUROSE; ROSS, 2013) with industry standard application protocols such as Hypertext Transfer Protocol (HTTP), first defined in Request For Comments (RFC) 1945, and WebSocket, defined in RFC 6455. Most of the HTTP communication will follow the widespread REST pattern (FIELDING, 2000).

¹ <https://github.com/fsmiamoto/zcart>

Figure 24 – High level architecture of zCart



Source: Own work (2022).

For handling the first goal, the *User App* will request data from other applications and will display the information to end users and allow them to interact with the cart using a touch enabled LCD display.

The *Cart Service* will be in charge of the second goal, handling requests from the *User App*, which asks for data on the products that are detected and notifies the completion of a purchase order when the user completes the checkout. The *Cart Service* uses a **relational database** (SILBERSCHATZ; FORTH; SUDARSHAN, 2010) as its primary database. Namely **SQLite**, considered to be the world's most deployed database due to its massive adoption on mobile devices². The simplicity of SQLite, the entire database is contained within a single file, great library support on common programming languages and the use of the familiar relational modeling, including Structured Query Language (SQL) (NIELD, 2016), were key factors in choosing it.

In order to provide real time updates to the *User App*, the *Cart Service* has a WebSocket API endpoint that allows the *User App* to listen to updates such as a product addition and then display a notification to the end user.

For the third goal, the *Product Recognizer* application will be responsible for processing a camera feed and weight sensor data to be able to tell if any products were added or removed from the cart. Any detected changes will be communicated to the *Cart Service*, which will be responsible for persisting those changes on the database.

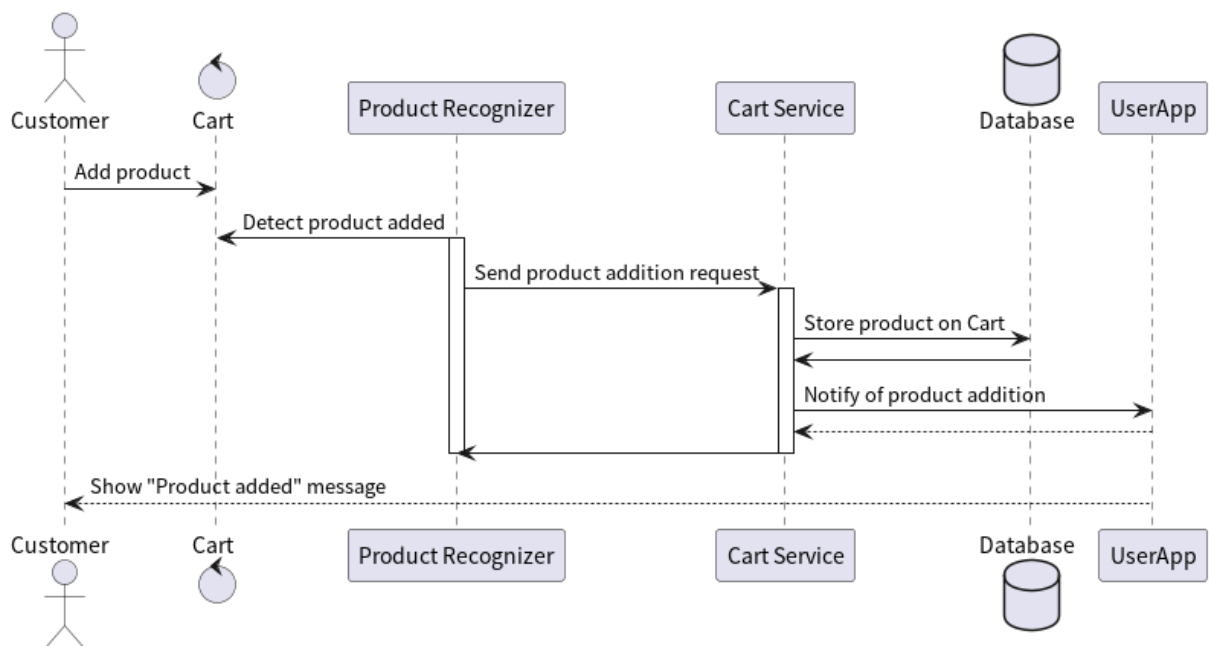
² <https://www.sqlite.org/mostdeployed.html>

All three applications will execute under a Linux (TANENBAUM; BOS, 2015) based environment on a Raspberry Pi³ 4 Single Board Computer (SBC) - a complete computer built on a single circuit board with a microprocessor, memory and input/output devices.

The advantages of using a Linux environment for the development are many, but being able to leverage its concurrency capabilities, having built-in drivers for readily available hardware and leveraging open source projects are some worth mentioning.

An End-to-End sequence diagram of an example action is shown in Figure 25.

Figure 25 – End-to-end sequence diagram for a product addition



Source: Own work (2022).

3.1.1 Architectural Guidelines

In creating the zCart architecture, the following guidelines were followed:

- Create a separate software application for each goal domain
- Use well defined standards for communication between applications.
- All databases should be owned by a single application.
- Any interaction that requires an update to a given database that is not owned by an application should be done through an API and not directly on the database.
- Decouple the user interface from how the data displayed is stored and transmitted

³ <https://www.raspberrypi.com>

These guideline are based on known best practices from the software development industry including API-first design and segregation of responsibilities (NEWMAN, 2021; KONG, 2022), which are key for future architectural evolution.

In the next sections, each application will be discussed in further detail.

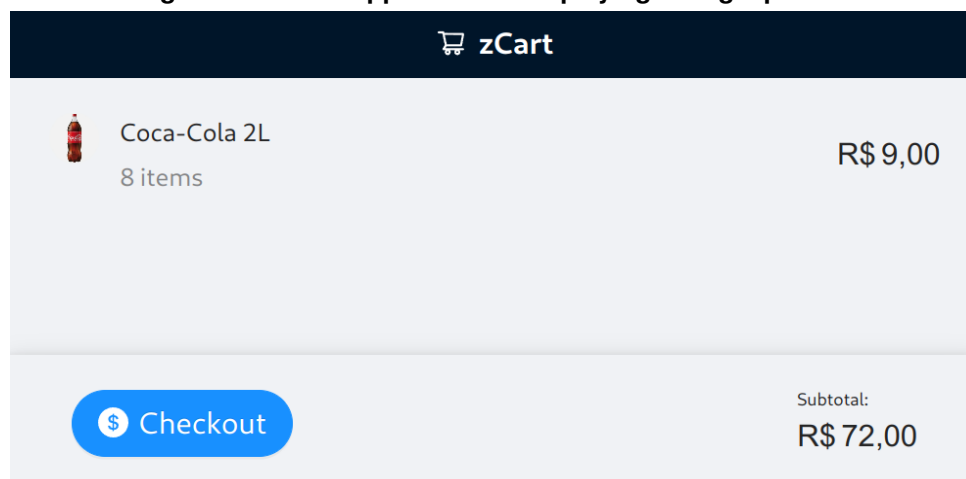
3.2 User App

For developing the User App, we have used web technologies such as HTML, CSS (DUCKETT, 2011) and JavaScript (FLANAGAN, 2020) using the React⁴ framework.

Using web-based technologies allows the User App to be displayed on any device capable of running a web browser; and having mature tooling for development, testing and debugging are important factors that influenced our decision.

As an alternative, developing Linux native graphical applications through toolkits such as GTK⁵ and Qt⁶ might have yielded better performance but our unfamiliarity with those would be a challenge.

Figure 26 – User App Interface displaying a single product



Source: Own work (2022).

As shown in Figure 26, the main objective of the User App is to provide a visual feedback mechanism for end users of the zCart. It displays the current products added to the cart, their amounts and also the price for each item. The subtotal price for all products added to the cart is calculated and also displayed on the interface. Notifications are also displayed when the user adds or removes a product from the cart.

Finally, the User App provides a *Checkout* button to simulate the payment process and act as a Proof-of-Concept (PoC), since a functional implementation of a payment mechanism is out of scope for our prototype.

⁴ <https://reactjs.org>

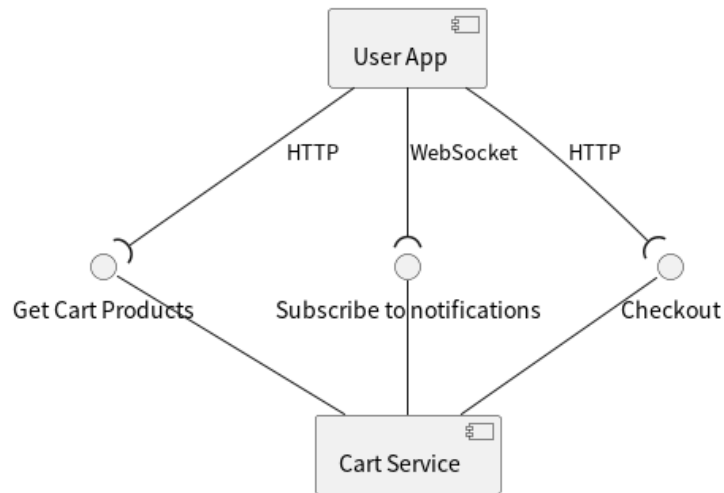
⁵ <https://gtk.org>

⁶ <https://qt.io>

More screenshots showing the user experience are available in Appendix A.

In terms of the data flow, the User App requests all data from the Cart Service, which exposes API endpoints for getting the list of products of a given cart, establishing a WebSocket connection for notifications and performing the PoC checkout. Figure 27 displays the endpoints of the Cart Service used by User App.

Figure 27 – User App and Cart Service interactions



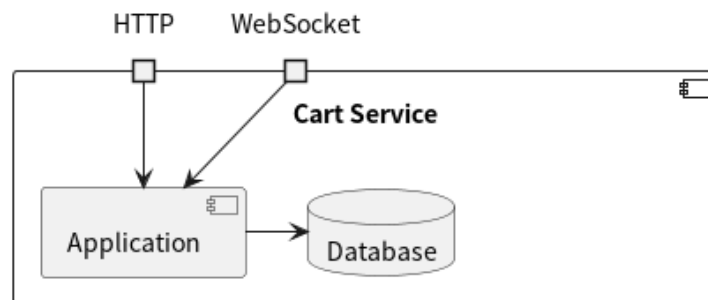
Source: Own work (2022).

3.3 Cart Service

As described in Section 3.1, the Cart Service will act as a centralized storage of the overall **state** of the cart.

For that, it will be responsible for managing the SQLite database and exposing API endpoints for the required state changes e.g. adding a product, as shown in Figure 28.

Figure 28 – Cart Service architecture. The database is contained within the Cart Service boundary and it is not exposed in the public APIs



Source: Own work (2022).

Source Code 1 – Example response for the GET /cart/:cartId endpoint using JSON

```

1 {
2   "id": "1",
3   "products": [
4     {
5       "cart_id": "1",
6       "product_id": "1",
7       "quantity": 11,
8       "product": {
9         "id": "1",
10        "name": "Coca Cola",
11        "description": null,
12        "price": 5.99,
13        "image_url": "https://zcart-test-images.s3.amazonaws.com/coca2l.png"
14      }
15    },
16  ]
17 }

```

Source: Own work (2022).

For the HTTP endpoints, the application uses the REST (FIELDING, 2000) pattern with JavaScript Object Notation (JSON)⁷ as the data-interchange format, both being widely employed in the software industry. Table 1 displays all the API endpoints created.

Table 1 – Cart Service API Endpoints.

HTTP Method	URI	Description
GET	/cart/:cartId	Get the cart data with the product listing
POST	/cart/:cartId/products	Add or remove a product from the cart
POST	/cart/:cartId/checkout	Perform checkout, emptying the cart

Source: Own work (2022).

An example response of the GET /cart/:cartId endpoint is shown in Source Code 1.

For implementing the Cart Service, the Go⁸ programming language was used alongside the Fiber⁹ framework, which provides great support for the creation of a HTTP Server and also handling the WebSocket connections.

One of the advantages of the Go language is the use of statically compiled native binaries, which allows running the application without the need to install any additional operating system libraries on the Linux environment.

⁷ <https://json.org>

⁸ <https://go.dev>

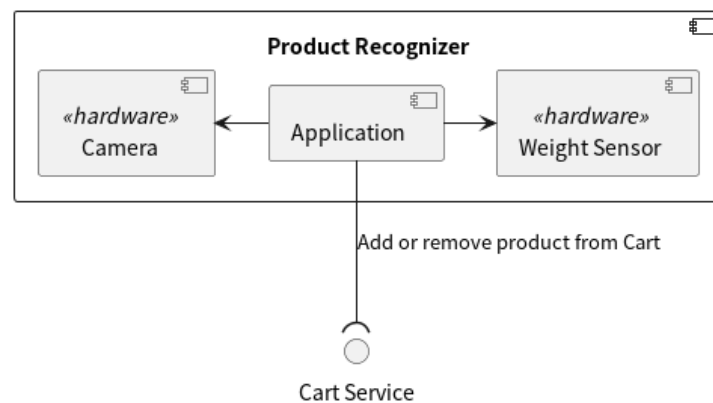
⁹ <https://gofiber.io/>

3.4 Product Recognizer

At the core of the zCart prototype is the Product Recognizer application, responsible for the product detection based on computer vision and weight sensing.

For achieving its goal, the Product Recognizer has three main components, as shown in Figure 29.

Figure 29 – Product Recognizer Components. Detected events will be communicated to the Cart Service for persistence



Source: Own work (2022).

Each of these components will be described in more detail in the next subsections.

3.4.1 Weight Sensor

For the Weight Sensor, we've used two main hardware components, shown on Figure 31:

- A 10 Kilogram Load Cell
- HX711 Integrated Circuit

The HX711 is a 24-bit Analog-to-Digital Converter (ADC) capable of outputting data in a serial interface (AVIA, 2022).

It has two channels for analog input with channel A having programmable gains of 128 or 64 and can function using both 3.3V and 5V standard digital voltage levels. The pinout is shown in Figure 30.

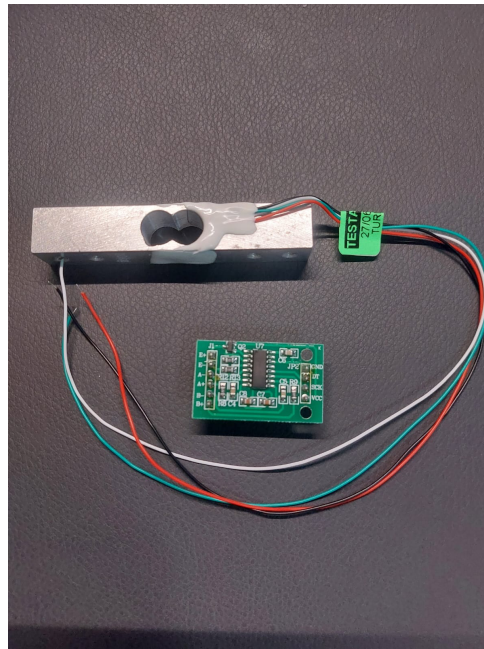
One of its advantages is that there's no need to program internal registers, all controls to the chip are through its pins. Additionally, it consumes only 1,5 mA under normal operation and has an on-chip power supply for the connected load cell, making it a great choice for our prototype.

Figure 30 – HX711 Pinout for the SOP-16L package

Regulator Power	VSUP	1	16	DVDD	Digital Power
Regulator Control Output	BASE	2	15	RATE	Output Data Rate Control Input
Analog Power	AVDD	3	14	XI	Crystal I/O and External Clock Input
Regulator Control Input	VFB	4	13	XO	Crystal I/O
Analog Ground	AGND	5	12	DOUT	Serial Data Output
Reference Bypass	VBG	6	11	PD_SCK	Power Down and Serial Clock Input
Ch. A Negative Input	INNA	7	10	INPB	Ch. B Positive Input
Ch. A Positive Input	INPA	8	9	INNB	Ch. B Negative Input

SOP-16L Package

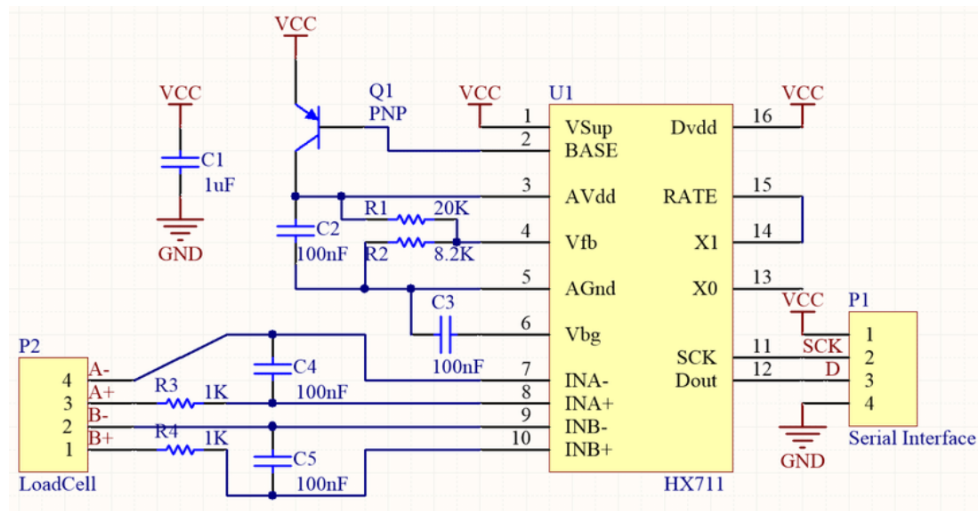
Source: Avia (2022).

Figure 31 – HX711 breakout board and load cell

Source: Own work (2022).

The HX711 integrated circuit comes in a breakout board, shown in Figure 31, that contains the necessary passive components and includes the pin headers for connecting with other boards. The typical application circuit of the HX711 integrated circuit is shown in Figure 32.

Figure 32 – Typical HX711 circuit



Source: Ross, Baji e Barnett (2019).

The Raspberry Pi board was connected to the HX711 through its GPIO pins, allowing it to obtain the sensor readings through the serial interface. The protocol used does not follow any known standard and can be described as a *Non-I2C compliant two-wire protocol*¹⁰.

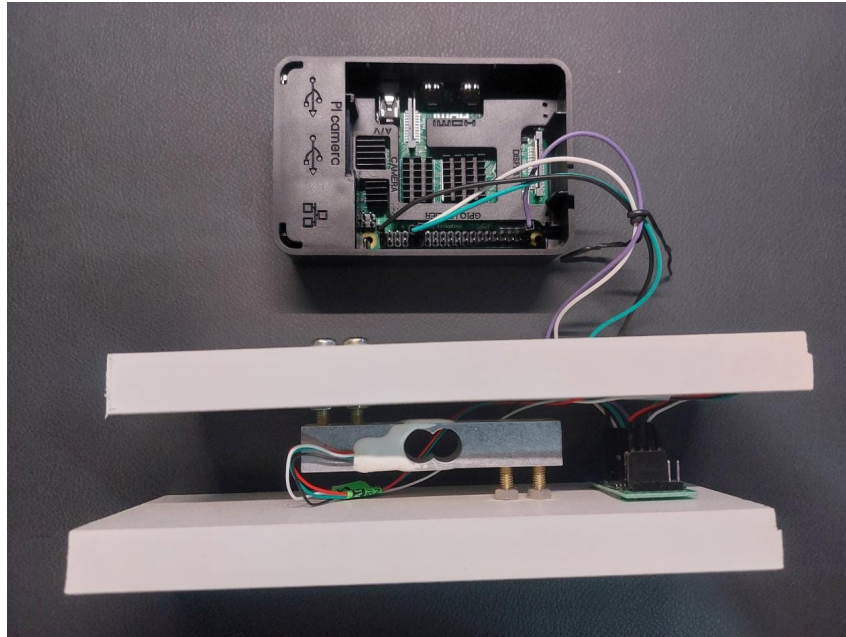
An open-source driver implementation was used¹¹, which included all the necessary features for the prototype.

As shown on Figure 33, the load cell requires a minimal mechanical assembly to be tested properly, and for that two small wood plates were used to secure the load cell and breakout board during development.

¹⁰ <https://github.com/queuetue/Q2-HX711-Arduino-Library>

¹¹ <https://github.com/tatobari/hx711py>

Figure 33 – Assembly used during development, including the Raspberry Pi, HX711 and load cell assembly



Source: Own work (2022).

3.4.2 Camera

In order to obtain a video feed to run object detection on, a camera system was required. For that, a standard consumer webcam was used for its reduced cost and good operating system driver support.

The webcam used was a Microsoft LifeCam Cinema¹², shown in Figure 34, capable of capturing video in 720p up to 30 Frames Per Second (FPS), more than enough for our prototyping requirements.

Figure 34 – Microsoft LifeCam Cinema Webcam used



Source: Own work (2022).

¹² <https://www.microsoft.com/pt-BR/accessories/products/webcams/lifecam-cinema>

3.4.3 Application

Developed in the Python¹³ programming language, the Product Recognizer application will provide the compute capabilities to apply our business logic for processing the video feed provided by the camera and the readings from the weight sensor.

The Python language was chosen for its vast tooling for working with computer vision, sensors and interacting with the Raspberry Pi's built-in devices. The language has also become a *lingua franca* in the Machine Learning and Data Science community as shown by volume of publications using it in the last decade (MCKINNEY, 2017; GRUS, 2019; MUELLER; GUIDO, 2016).

The Product Recognizer application also uses a multi-threaded design to allow concurrent processing, an important factor when considering the amount Input/Output (I/O) operations performed.

Since Python's threading implementation does not allow for multiple threads to execute in parallel – i.e. at the same time – a multi-process design could take better advantage of the four available processing cores in the Raspberry Pi CPU but that path was not explored and will be left for future iterations.

The three threads created are described below:

- Frame Reader Thread: Responsible for reading frames from the Camera and making them available to the Main Thread.
- Main Thread: Responsible for bootstrapping the application - including creating the other threads - and executing the main loop of object detection.
- Product Recognizer Thread: Responsible for applying the business logic using the objects detected and the weight sensor readings.

These threads communicate in synchronous and asynchronous means to achieve the overall goal of processing video and sensor data, as shown on Figure 35.

¹³ <https://python.org>

Source Code 2 – Loop of the Main Thread. Some details were omitted for the sake of brevity

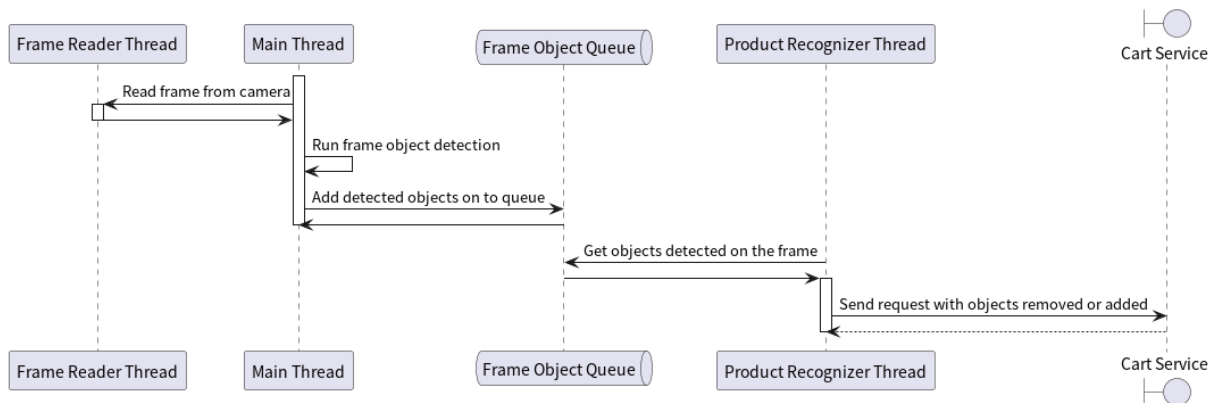
```

1  while True :
2      try :
3          # Get frame from camera, provided by Frame Reader thread
4          frame = stream.read_frame()
5
6          # Preprocess and run object detection
7          input = preprocessor.process(frame)
8          detector.infer(input)
9
10         # Filter objects by their class and confidence threshold
11         objects = detector.get_objects()
12         filtered_objects = object_filter.filter(objects)
13
14         # Add filtered objects to the queue
15         frame_objects_queue.put(filtered_objects)

```

Source: Own work (2022).

Figure 35 – Thread communication for the Product Recognizer Application



Source: Own work (2022).

As shown on Source Code 2 and in Figure 35, the Main thread will do the computational work to fetch the frames from the camera that were read by the Frame Reader thread, preprocess and run the object detection model in it. The objects are then filtered and then added to a message queue that will be used by the Product Recognizer thread to apply our product detection rules.

3.4.4 Product Recognizer Thread

The Product Recognizer Thread is responsible for applying the main business logic of detecting the addition and removal of products by leveraging the objects detected in the frame and the readings provided by the weight sensor.

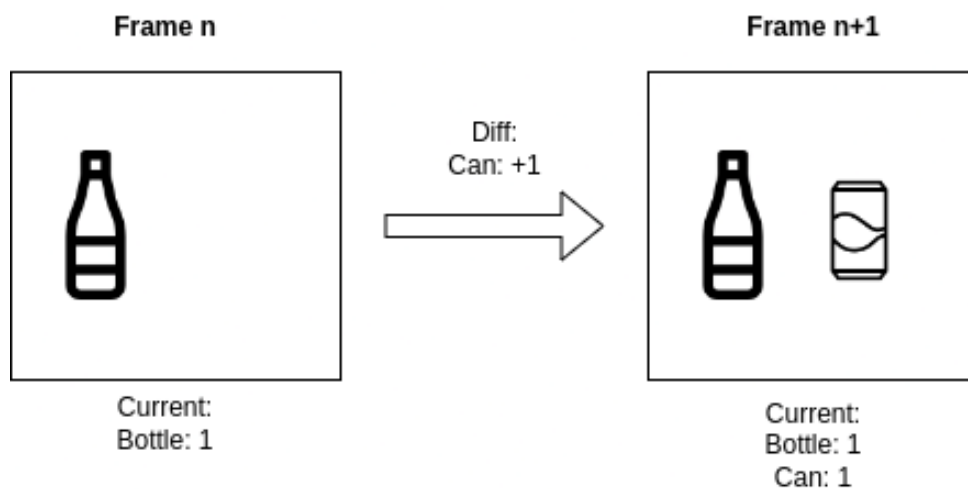
The thread will execute an infinite loop to constantly observe the objects detected in the frame and the weight readings to detect product additions and removals.

It uses two main variables to keep track of the current state:

- A dictionary of which products are present in the cart and their count.
- The weight associated with the products present in the cart.

The first important step in the thread's logic is the *Frame Diff*. This step calculates the difference in terms of the objects detected in the current and previous frames (e.g. whether objects were added, removed, or remain the same) and store it in a dictionary data structure¹⁴, by comparing the stored product dictionary and the received list of frame objects from the Main Thread.

Figure 36 – Illustration of the frame diff'ing scheme. A can was added on frame $n+1$ and therefore will generate a diff of one can. If the reading from the weight sensor matches the expected one, the product will be added to the cart.



Source: Own work (2022).

Given the frame object diff, it is then possible to calculate an expected weight change based on the weight of each item - and their quantity - contained in the diff. The expected weight difference is then compared to the actual one obtained from the weight sensor, considering a configurable tolerance.

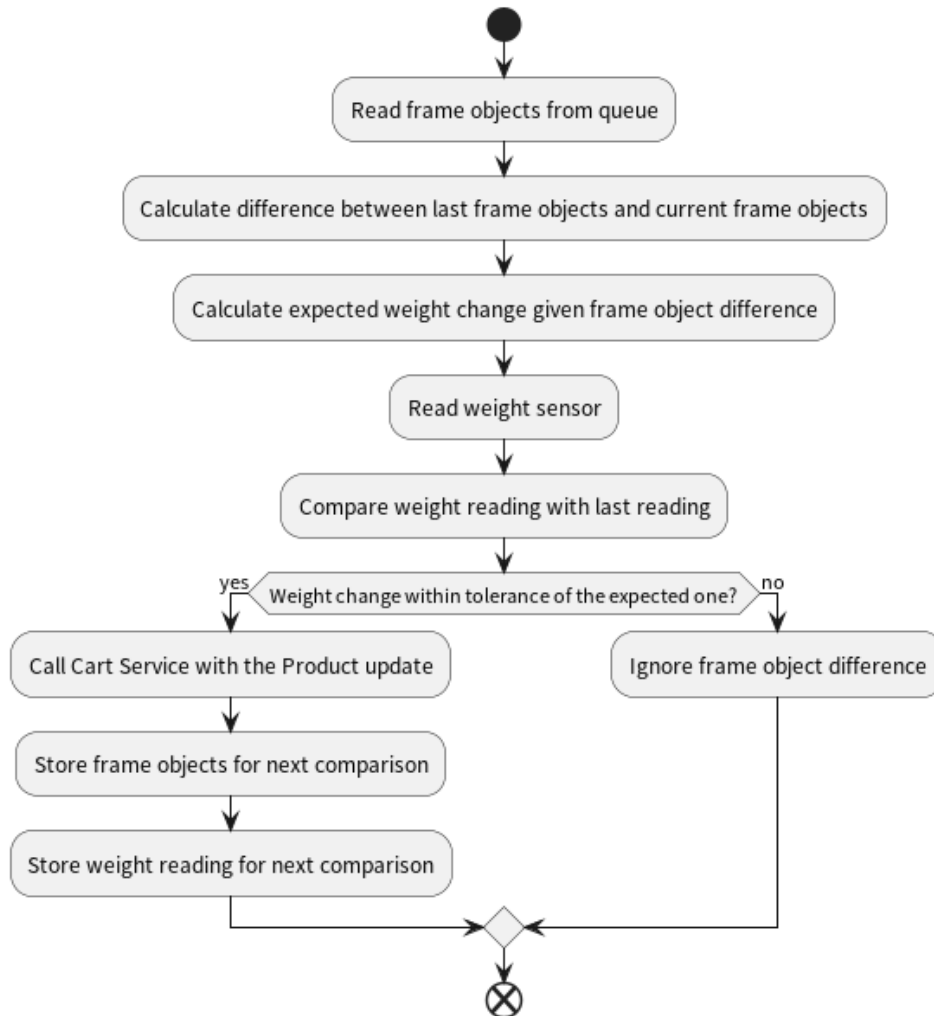
In this scenario, the weight readings are used as a filter and act as a *commit* step for differences detected in the frame objects. This way, objects are not added or removed from the cart simply by appearing or disappearing from the frame.

In the example show in Figure 36, if the illustrated can weights an expected 400 grams, the weight difference expected should be close to 400 grams. If the weight difference does not match the expected one, the object will not be considered for addition or removal.

¹⁴ <https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

A more detailed activity diagram of the logic executed in the Product Recognizer thread loop is shown in Figure 37.

Figure 37 – Activity diagram for the Product Recognizer thread



Source: Own work (2022).

3.4.5 Object Detection Model

The code used to train our Object Detection models was made available in GitHub, in our Project's Repository¹⁵. It can be used for reference and for performing a further deep dive into this section.

Five products were chosen for the purpose of setting up and demonstrating the Object Recognition capability of our product: a Blue Pens Packet; a Card Deck; a Coke Can; a Guarana Soda Can; and a Post It Pack. The data used for training our custom Object Detection model was collected and labelled by us using Edge Impulse¹⁶, a development platform for Machine

¹⁵ https://github.com/fsmiamoto/zcart/blob/master/product_recognizer/model/notebooks/

¹⁶ <https://www.edgeimpulse.com/>

Source Code 3 – Product Recognizer thread logic

```

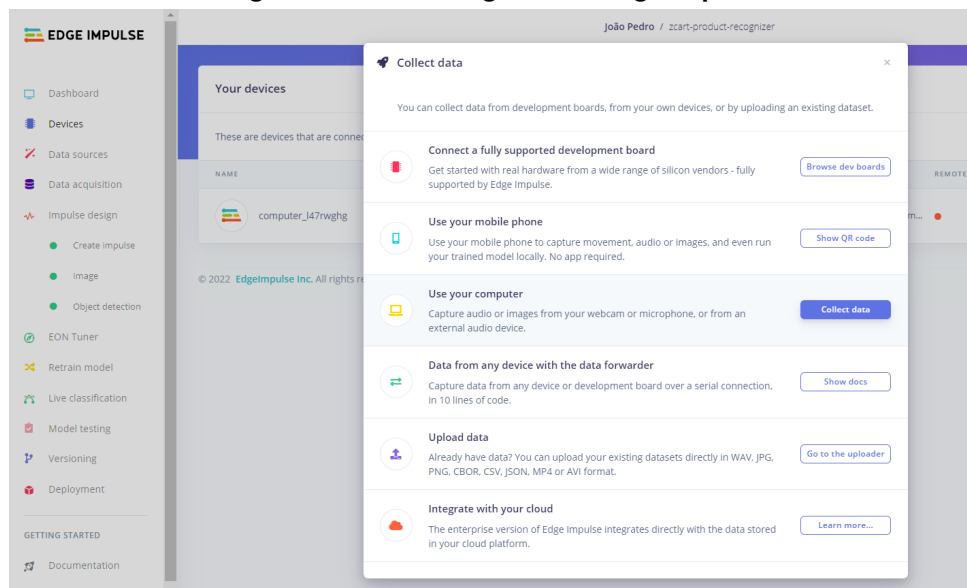
1  objects = self.queue.get_nowait()
2
3  # Preprocessing step just to format data
4  current_frame_objects = self.__build_object_dict(objects)
5
6  # Calculate the frame diff using the last and current frame objects
7  frame_diff = self.__get_frame_diff(
8      current_frame_objects, self.last_frame_objects
9  )
10
11 if len(frame_diff) == 0:
12     self.log.info("empty diff")
13     continue
14
15 # Get current weight reading
16 weight_reading = self.weight_sensor.get_reading(samples=5)
17
18 for label, count in frame_diff.items():
19     if not self.__valid_weight_difference(label, count, weight_reading):
20         self.log.info("ignoring, not valid weight difference")
21         continue
22
23     # Send request to cart service
24     self.__call_cart_service(label, count)
25
26     # Update stored state
27     self.last_weight_reading = weight_reading
28     self.last_frame_objects[label] = current_frame_objects[label]
29     if self.last_frame_objects[label] == 0:
30         del self.last_frame_objects[label]

```

Source: Own work (2022).

Learning on Edge devices. A total of **1000 photos** were taken using the Data Collection feature of the Edge Impulse Studio, shown in Figure 38, using the same webcam that is used for the inference in our final product. Our photos included images showing each one of the products in different scenarios and positions, and also images where more than one product was present.

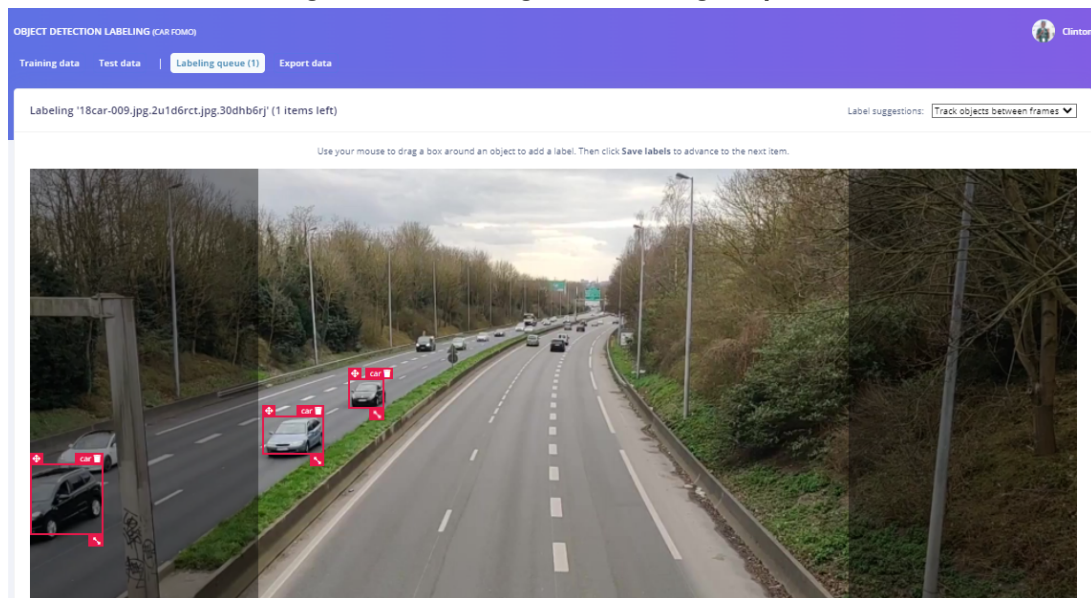
Figure 38 – Collecting Data in Edge Impulse



Source: Edge Impulse (2022).

The pictures were then labeled using the Labeling Queue¹⁷ feature of Edge Impulse, which allowed us to draw Bounding Boxes around the desired objects for detection, as shown in Figure 39.

Figure 39 – Labeling Queue in Edge Impulse



Source: Edge Impulse (2022).

The raw pictures and bounding boxes were then exported from Edge Impulse such that we could pre-process the data and model our custom object detection algorithm using the Python programming language. The pictures were exported in their raw JPEG¹⁸ file format, comprised

¹⁷ <https://docs.edgeimpulse.com/docs/edge-impulse-studio/data-acquisition/labeling-queue>

¹⁸ Defined in ISO/IEC 10918

Source Code 4 – Bounding boxes coordinates file exported from Edge Impulse

```

1  {
2    "version": 1,
3    "type": "bounding-box-labels",
4    "boundingBoxes": {
5      "im1.jpg": [
6        {
7          "label": "blue_pens",
8          "x": 120,
9          "y": 265,
10         "width": 172,
11         "height": 207
12       },
13       {
14         "label": "card_deck",
15         "x": 285,
16         "y": 120,
17         "width": 136,
18         "height": 247
19       }
20     ],
21     "im2.jpg": [
22       (...)
23     ]
24   }
25 }

```

Source: Own work (2022).

in a ZIP folder. The bounding boxes were exported in the format of a JSON file containing the coordinates for the boundaries and the metrics for each picture, allowing us to easily reconstruct the bounding boxes for each object in each image using programming instructions. The files were then loaded to a Cloud Object Storage Bucket in AWS¹⁹, making it easier for us to access the data by importing it from the web using any programming language.

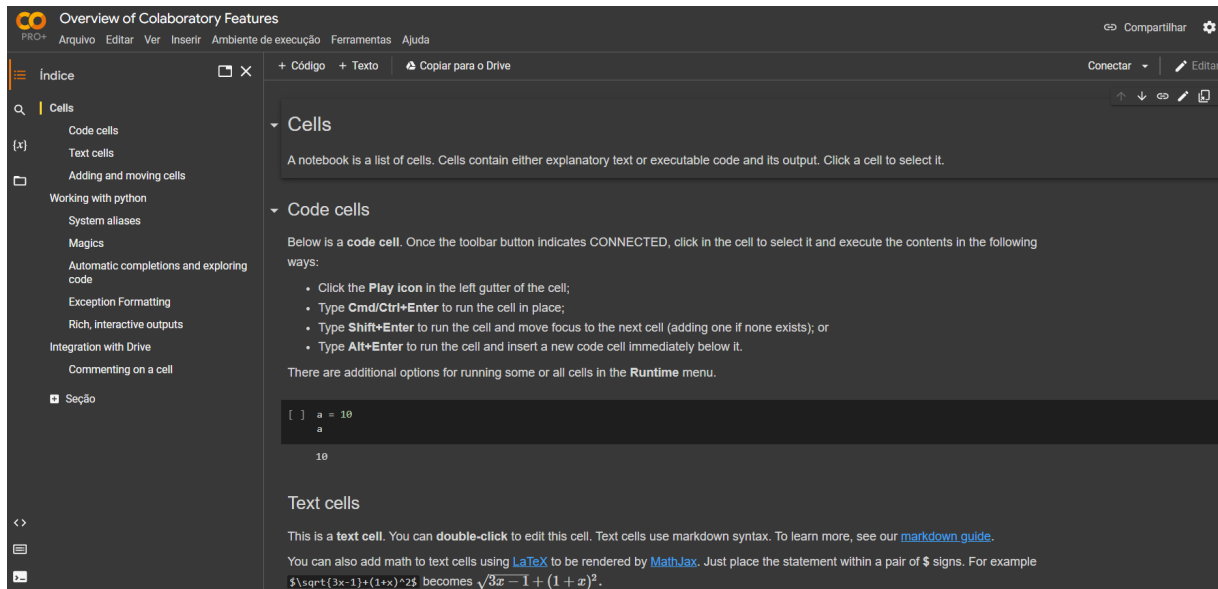
Source Code 4 shows a section of an example JSON file containing the bounding boxes.

The development environment used for writing the code for training our custom model was Google Colab²⁰, and the programming language of choice was Python. Google Colab consists on a web-based interface, shown in Figure 40, that allows developers to use Google's infrastructure (with GPUs and TPUs) for writing and executing code.

¹⁹ <https://aws.amazon.com/>

²⁰ <https://colab.research.google.com/>

Figure 40 – Google Colab - Overview of Colaboratory Features



Source: Google²¹ (2022).

The Edge Impulse files were loaded from our cloud object storage bucket, and then manipulated in Python from Google Collab such that we could utilize the images and bounding box coordinates for training our model.

We decided to use Google's TensorFlow²² framework to train our models, as it is one of the most popular frameworks employed in the Industry for training AI models and also because it features a lightweight library called TensorFlow Lite²³, which is appropriate for creating efficient edge and mobile AI models considering the typical hardware constraints from these types of devices.

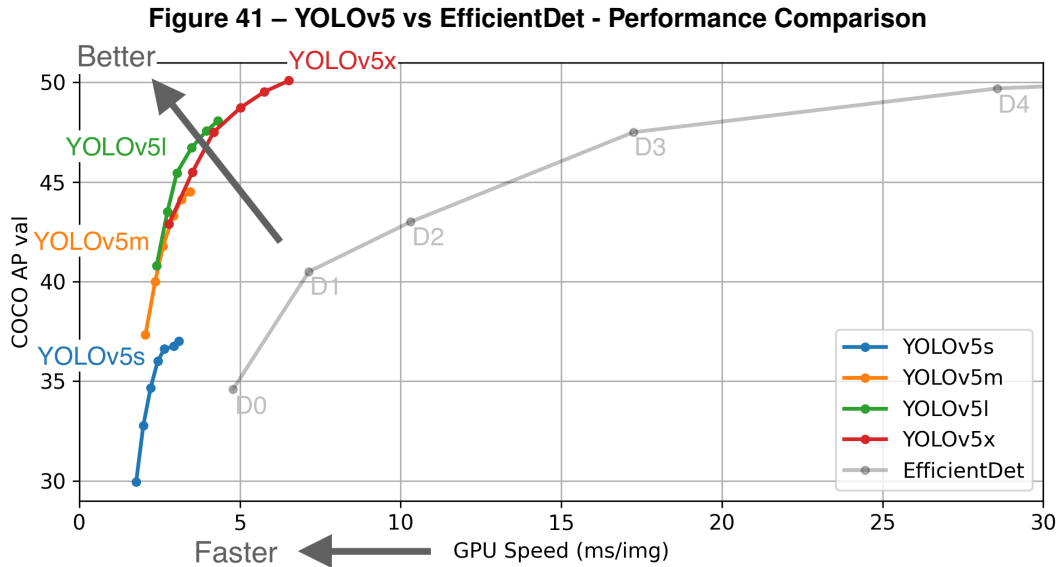
To train our model, we used the TensorFlow Lite's ModelMaker API²⁴ for Object Detection, which simplifies the process of training our models by breaking down the most complex concepts of deep learning into parameterized objects and methods, allowing us to spend more time on the pre-processing steps and on experimenting with different network configurations to improve the model's accuracy.

As of writing this work, the ModelMaker API only has compatibility with the EfficientDet family of architectures (TAN; PANG; LE, 2020) for training Deep Learning models for Object Detection. The EfficientDet family is efficient for object recognition in edge devices, although contemporary architectures, such as the YOLOv5 and its successors, can have better performance, as shown in Figure 41.

²² <https://www.tensorflow.org/>

²³ <https://www.tensorflow.org/lite>

²⁴ https://www.tensorflow.org/lite/models/modify/model_maker



Source: YOLOv5 Release Notes²⁵ (2022).

Our initial plan was to use the YOLOv5 architecture, but since its implementation is not native to TensorFlow Lite, it would require us to create additional wrappers around the outputs of the YOLOv5 network to get it working as expected. Even though it is possible to convert the models from their original PyTorch²⁶ format to a TensorFlow Lite format, the conversion does not cover some of the features from the original implementation²⁷, which limits its direct functionality. Similar compatibility issues happen with the latest YOLO implementations, such as the YOLOv7²⁸. Thus, we have decided to proceed with the TensorFlow Lite's Model Maker API compatible architectures – namely the EfficientDet family – since it would have taken a considerable time to troubleshoot conversion defects from the YOLO architecture and all that work would not bring any additional value to our prototype.

Since our original images were saved in the 640x480 resolution and the expected input of the EfficientDet network is of 320x320 px, two pre-processing approaches were tried out for the training dataset: resizing and cropping the images. The bounding box coordinates and dimensions were also adjusted accordingly such that the data integrity was preserved. Figure 42 illustrates both of the approaches used.

²⁶ <https://pytorch.org/>

²⁷ <https://github.com/ultralytics/yolov5/issues/1981>

²⁸ <https://medium.com/geekculture/journey-putting-yolo-v7-model-into-tensorflow-lite-object-detection-api-model-running-on-android-e3f746a02fc4>

Source Code 5 – CSV format for specifying the Train, Test and Validation image sets for training models using the TensorFlow’s Model Maker API for Object Detection

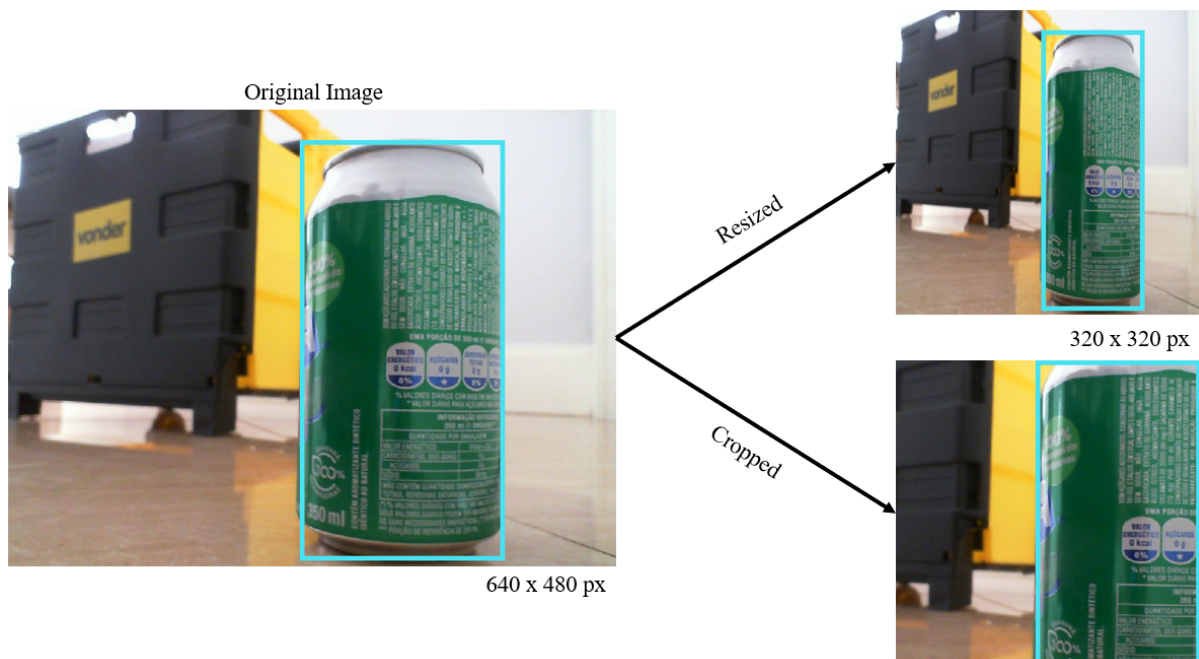
```

1 Template :
2 set , path , label , x_min , y_min , , , x_max , y_max , ,
3 set , path , label , x_min , y_min , x_max , y_min , x_max , y_max , x_min , y_max
4
5 Examples :
6 TEST , ./ images_path / im3 . png , 0.5 , 0.6 , , , 0.2 , 0.9 , ,
7 VALIDATION , ./ images_path / im4 . png , coke_soda , 0.3 , 0.4 , , , 0.4 , 0.8 , ,
8 TRAIN , ./ images_path / im5 . png , guarana_soda , 0.3 , 0.3 , 0.8 , 0.8 , 1.0 , 0.9 , 0.1 , 1.0
9 TRAIN , ./ images_path / im5 . png , post_it , 0.3 , 0.1 , , , 0.3 , 0.4 , ,

```

Source: Own work (2022).

Figure 42 – Image Preprocessing Strategies



Source: Own work (2022).

Once the images and bounding boxes were pre-processed, the images were split into Train, Validation and Test sets; and saved in the standardized format required by the TensorFlow Lite’s Model Maker for Object Detection API for training, which consists of a Comma Separated Values Comma Separated Values (CSV) file with the structure shown in Source Code 5.

Approximately 70% of the photos shot were moved to the Train set, which is the set used for effectively tuning the weights and biases of the custom models; About 21% were moved to the Validation set, being used to understand our model’s performance under different training scenarios and steps; and the rest was used for testing the custom model after it was trained, which allowed us to get unbiased metrics on how it would approximately perform in real life (WILBER; WERNESS, 2022).

Table 2 describes the data sets created for training and evaluation in greater detail.

Table 2 – Image Sets Created for Training, Validation and Testing

Class	Number of Pictures	Pictures in the Train Set	Pictures in the Validation Set	Pictures in the Test Set
Blue Pens Packet	293	206	61	26
Card Deck	306	215	64	27
Coke Can	276	194	58	24
Guarana Soda Can	283	199	59	25
Post It Pack	273	192	57	24

Source: Own work (2022).

Finally, we defined a programming loop to train different models using the EfficientDet-D0, EfficientDet-D1, EfficientDet-D2, EfficientDet-D3 and EfficientDet-D4 architectures; both by applying Transfer Learning on the top of pre-trained weights from training with the COCO-2017 dataset²⁹ and by training the entire networks based in our custom data.

We ran this entire loop using the cropped images first; and then executed it using resized images with the EfficientDet-D0, EfficientDet-D1 and EfficientDet-D2 architectures too, as they were the ones who offered better performance balance considering our hardware constraints. With that, we came to have 16 distinct custom models for experimenting, as shown in Table 3.

Table 3 – Models Trained

Model	Image Pre-Processing Strategy	Architecture	Whole Trained/Transfer Learning
1	Cropping	EfficientDet-D0	Whole Trained
2	Cropping	EfficientDet-D1	Whole Trained
3	Cropping	EfficientDet-D2	Whole Trained
4	Cropping	EfficientDet-D3	Whole Trained
5	Cropping	EfficientDet-D4	Whole Trained
6	Cropping	EfficientDet-D0	Transfer Learning on COCO-2017
7	Cropping	EfficientDet-D1	Transfer Learning on COCO-2017
8	Cropping	EfficientDet-D2	Transfer Learning on COCO-2017
9	Cropping	EfficientDet-D3	Transfer Learning on COCO-2017
10	Cropping	EfficientDet-D4	Transfer Learning on COCO-2017
11	Cropping	EfficientDet-D0	Whole Trained
12	Cropping	EfficientDet-D1	Whole Trained
13	Resizing	EfficientDet-D2	Whole Trained
14	Resizing	EfficientDet-D0	Transfer Learning on COCO-2017
15	Resizing	EfficientDet-D1	Transfer Learning on COCO-2017
16	Resizing	EfficientDet-D2	Transfer Learning on COCO-2017

Source: Own work (2022).

3.5 Mechanical Assembly

A foldable utility cart was selected as a core component with two additional wood plates, used for creating a *false bottom* shown in Figure 45. Between the plates, the load cell and Raspberry Pi board were secured in place using bolts, screws and velcro.

²⁹ <https://cocodataset.org/#home>

Figures 43 and 44 show an overview of the structure built.

Figure 43 – Overall mechanical assembly including the LCD and Camera



Source: Own work (2022).

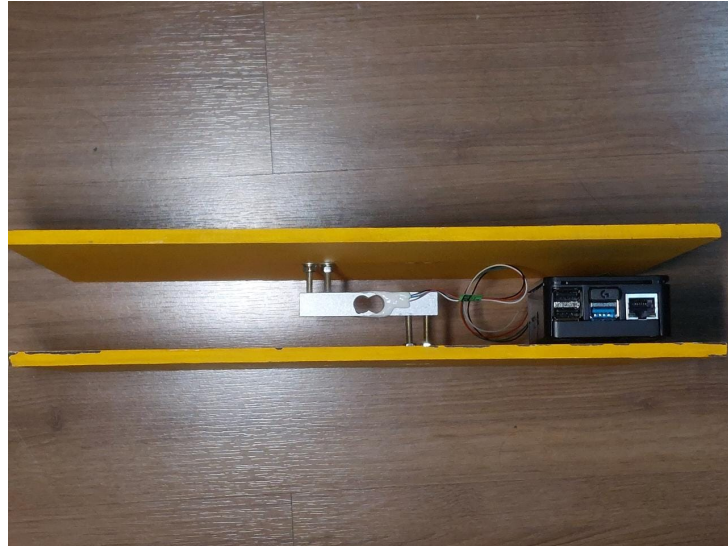
Figure 44 – Top view of the assembly showing the false bottom



Source: Own work (2022).

Considering the objectives of the prototype, the shape of the mechanical housing was not considered to be of great relevance and using a real supermarket cart would have been impractical considering its size and cost. Still, we wanted to keep a shape that would represent the overall idea of a smart cart.

Figure 45 – False bottom structure with the load cell and Raspberry Pi board in between



Source: Own work (2022).

Additional photos of the prototype can be seen on Appendix B.

4 RESULTS

4.1 Model Accuracy

The most lightweight models were the preferred ones considering our edge device capabilities, as they offer better inference time performance. However, we were also looking for the most optimal accuracy and loss metrics, therefore the models were evaluated with test images, which were never introduced to the models during training, to make sure that they were efficient on the what they are ultimately supposed to do, which is to detect products.

Figures 46, 47, 48 and 49 show examples of Single and Multi label classification.

Figure 46 – Single-Label Classification Example 1



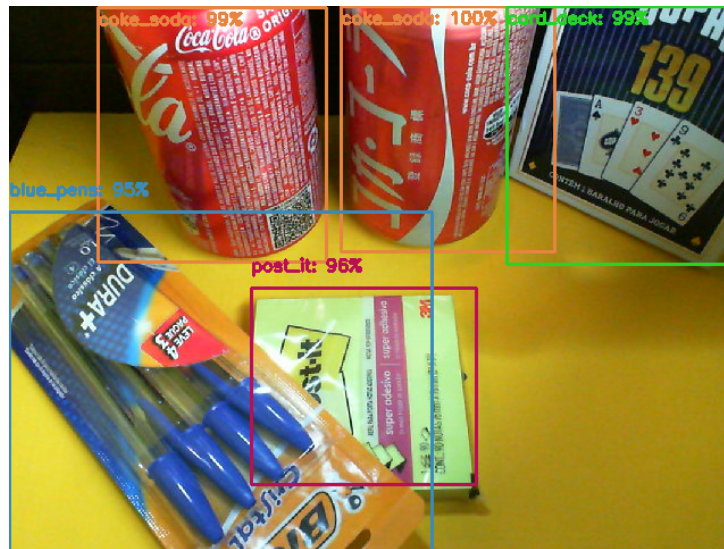
Source: Own work (2022).

Figure 47 – Single-Label Classification Example 2



Source: Own work (2022).

Figure 48 – Multi-Label Classification Example 1



Source: Own work (2022).

Figure 49 – Multi-Label Classification Example 2



Source: Own work (2022).

The accuracy metrics were also inferred in our Test data set by using the TensorFlow Lite's Object Detector *evaluate* method¹ for all different model configurations that were applied. The main evaluation metrics for the models considered for our product, which were the ones based in the EfficientDet-D0, D1 and D2 architectures – as the D3 and D4 architectures were too computationally expensive for our device – are listed in Table 4.

The numbers exhibited in the table consist of the classification AP, which measures the percentage of correctly labeled predictions amongst all predictions; the AP with an IoU of 50%, which means that there is at least 50% of overlap between the predicted and the actual bounding

¹ https://www.tensorflow.org/lite/api_docs/python/tflite_model_maker/object_detector/ObjectDetector#evaluate

boxes; the AP with an IoU of 75%; and the individual classification APs for each label that was forecasted.

Table 4 – Test Evaluation Metrics for the Different Strategies and Models Applied for Inference

Architecture	Preprocessing Strategy	Training Strategy	AP	AP 50 IoU	AP 75 IoU	AP (post_it)	AP (guarana)	AP (coke)	AP (card_deck)	AP (blue_pens)
EfficientdetD0	Resizing	Transfer Learning	0.527	0.700	0.644	0.521	0.443	0.685	0.490	0.497
EfficientdetD1	Resizing	Transfer Learning	0.623	0.786	0.740	0.409	0.639	0.812	0.670	0.588
EfficientdetD2	Resizing	Transfer Learning	0.653	0.799	0.774	0.587	0.659	0.825	0.669	0.524
EfficientdetD0	Resizing	Whole	0.813	0.978	0.906	0.752	0.765	0.891	0.921	0.736
EfficientdetD1	Resizing	Whole	0.802	0.939	0.875	0.719	0.675	0.903	0.928	0.785
EfficientdetD2	Resizing	Whole	0.826	0.967	0.922	0.763	0.776	0.910	0.895	0.784
EfficientdetD0	Cropping	Transfer Learning	0.490	0.668	0.592	0.357	0.422	0.595	0.633	0.443
EfficientdetD1	Cropping	Transfer Learning	0.580	0.730	0.686	0.312	0.605	0.716	0.736	0.531
EfficientdetD2	Cropping	Transfer Learning	0.562	0.722	0.699	0.463	0.569	0.576	0.661	0.539
EfficientdetD0	Cropping	Whole	0.832	0.954	0.934	0.869	0.854	0.717	0.922	0.800
EfficientdetD1	Cropping	Whole	0.792	0.897	0.880	0.823	0.763	0.586	0.933	0.852
EfficientdetD2	Cropping	Whole	0.803	0.923	0.893	0.795	0.808	0.631	0.941	0.840

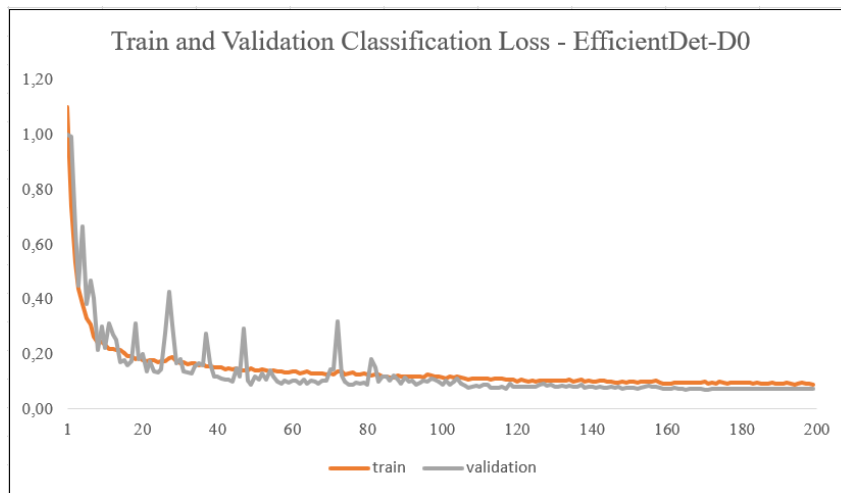
Source: Own work (2022).

We could clearly see that, while the Transfer Learning models were much faster to train, in our particular case, the Whole-trained models outperformed them. This could be due to the fact that our pictures and objects are different from the ones that are present in the COCO-2017 dataset; or because a custom feature extractor- with custom hidden layer weights and biases- could have better performance with our pictures. In terms of the architectures, as expected, the D2 architecture offered more robust results, specially amongst the models that applied resizing as a preprocessing strategy.

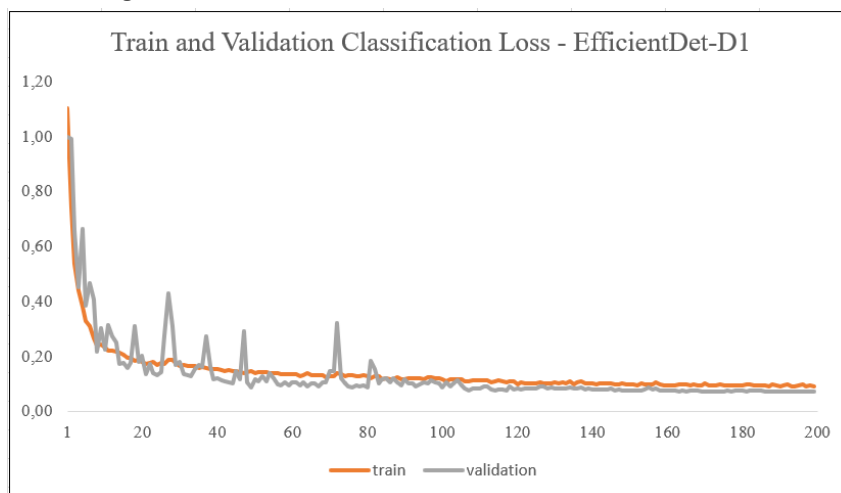
We could not see much superior metrics for the models that used cropping for the images preprocessing, and that might be because most of the bounding boxes ended up being cropped as well and, with that, we lost a portion of valuable label data. Although the EfficientDet-D0 model that was Whole-Trained with Cropped images had a comparable performance in our Test Data evaluation, considering the end-to-end usability tests that we executed with the assembled prototype and the overall better performance presented by the D2 architecture, as shown in figure 41, we selected the EfficientDet-D2 model that was Whole-Trained with Resized images as our champion model.

The loss (error) metrics during training were also computed for our Train and Validation sets during the 200 epochs that were used for training our models using batches of 16 images, for all different settings that were employed to train them. We can clearly state that the model showed significant improvement as the epochs progressed, and maybe more epochs would even have brought greater performance, as the weights would have been even more fine-tuned. The trade-off, though, is that it would have taken more time and computer power to train the models.

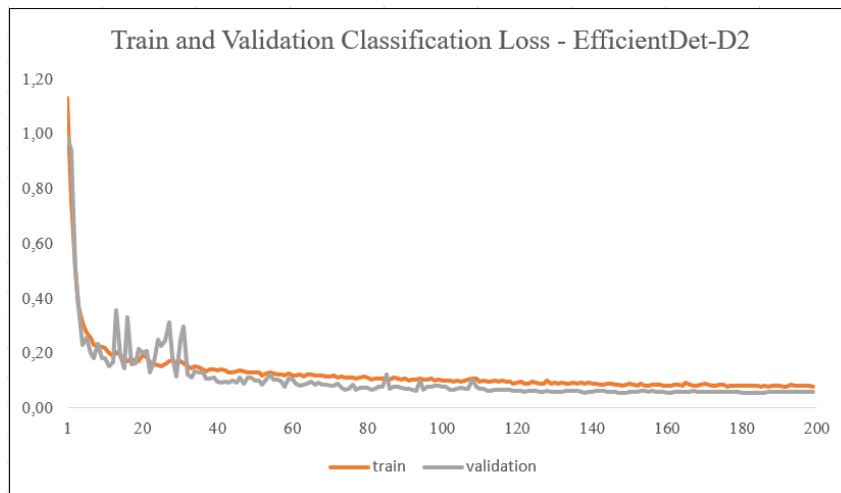
In Figures 50, 51 and 52, you can find the Classification Loss chart for our three top models of choice, which were the EfficientDet-D0, D1 and D2, whole-trained and that used resizing as a preprocessing strategy.

Figure 50 – Loss Chart for the EfficientDet-D0 Network

Source: Own work (2022).

Figure 51 – Loss Chart for the EfficientDet-D1 Network

Source: Own work (2022).

Figure 52 – Loss Chart for the EfficientDet-D2 Network

Source: Own work (2022).

Note that those spikes in the loss values are probably due to the batch size, as it could be that in specific batches of 16 images the model was not fully prepared to predict the labels in those images. Using a larger batch would likely solve it, however it would also take more RAM memory consumption.

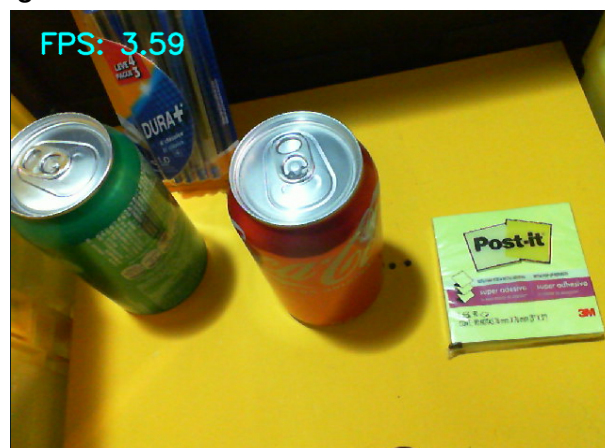
Finally, to improve inference time performance, we also compiled our TensorFlow Lite models for usage with an Edge TPU, which further reduced our inference times.

Table 5 – Inference Performance Metrics with and without the TPU

Model	FPS without the TPU	FPS using the TPU	CPU Ops using the TPU	TPU Ops using the TPU
Efficientdet-D0 Whole-Trained on Resized Images	3.69	6.87	3	264
Efficientdet-D1 Whole-Trained on Resized Images	2.09	5.14	131	191
Efficientdet-D2 Whole-Trained on Resized Images	1.36	3.50	131	226

Source: Own work (2022).

Figure 53 displays an example frame from the camera feed displaying the current FPS count.

Figure 53 – Frame from camera feed with FPS count

Source: Own work (2022).

The inference speed has thoroughly improved - about 50% on average - resulting in a greater capability to process more Frames Per Second. When we look at the number of CPU and TPU Operations, though, what we would expect when using the TPU is that most Operations happen on the TPU side, however this behavior could only be seen in the EfficientDet-D0 model. Part of it is because the EfficientDet-D0 model has a simpler architecture, but it could also be due to the conversion of the model for usage with the TPU or to the limited capacity of the TPU device that was used. For instance, Google suggests using two TPU cores for the EfficientDet-D2 model², because the tensors are too large to fit in the chip's memory.

Overall, after considering the tradeoff between precision and performance, we have decided to use the EfficientDet-D2 model since its performance when paired with the Coral TPU was more than enough for our purposes (around 4 FPS) and had the best precision, improving the overall experience of the prototype.

4.2 Power Consumption

For estimating the overall power consumption, a commercial wattmeter was used to observe the total power required by the power adapter used, as shown in Figure 54.

Figure 54 – Power consumption measurement using a wattmeter



Source: Own work (2022).

² https://www.tensorflow.org/lite/models/modify/model_maker/object_detection

During our tests, we have observed an average consumption of **10,4W** when running the all the software and hardware components of the prototype through an external power adapter.

The tests were performed on a 220V power line using the following equipment:

- Sinotimer DDS108 Digital Wattmeter
- Baseus Quick Charger GaN 65W

Since both of these products do not have detailed accuracy and efficiency data available, we estimate an overall 10% margin of error considering their construction (CHEN *et al.*, 2017).

With that, we can assume that the actual power draw is between 9,4W and 11,4W. Considering a desired battery life of 24 hours, it would require a battery of about 240 Wh, which can be found commercially and would not impose a insurmountable practical barrier.

4.3 Cost

One of the important aspects when developing a marketable product is its **cost**. A more detailed overview of the items and their costs is shown on Table 6.

Table 6 – Items used on the prototype and their retail cost in October 2022

Item	Quantity	Cost per item (BRL/USD)
Raspberry Pi 4 8GB Board	1	R\$ 1072,40 / US\$ 201,94
Coral Edge TPU	1	R\$ 318,51 / US\$ 59,99
HX711 with breakout board	1	R\$ 2,55 / US\$ 0,48
10Kg Load Cell	1	R\$ 6,89 / US\$ 1,30
MDF Board	2	R\$ 10,00 / US\$ 1,89
Mounting hardware (screws, bolts and nuts)	1	R\$ 5,00 / US\$ 0,94
Foldable utility cart	1	R\$ 125,00 / US\$ 23,60
Webcam	1	R\$ 100,00 / US\$ 18,88
Total cost		R\$ 1650,35 / US\$ 310,91

Source: Own work (2022).

Of course, the developed prototype does not include all the necessary hardware and software structure to deliver a successful product, but still it might show that at a fundamental level, the cost of such a solution might not reach the costs that current smart carts in the market sell for³.

Therefore, it might be possible to conclude that most of the retail cost of the existing smart carts is not composed of the production and infrastructure costs but from the required repayment of the research and development costs that such a product demand.

³ The Nextop cart shown on the introduction currently retails for R\$ 120.000

4.4 Challenges and future work

As one of the expected outcomes of our work, we have been able to identify several practical challenges that would need to be worked on for a marketable product and will be discussed in the next sections.

4.4.1 Extending the model for new products

In a supermarket use case, we expect that products will need to be added or removed from detection model on a regular basis. That becomes a challenge when we consider the amount of data necessary to train the model used in the developed prototype.

Considering that, an important next step on the development would be to work on a model that can be easily extend to support new products without requiring too much computational power for retraining.

4.4.2 Deploying updates

Considering the compute locality of the detection model used in the prototype, which is the board embedded in the cart, deploying updates to the model to it might become a challenge.

Changing the compute locality to a Cloud infrastructure (AWS, 2022) might allow for easier deployment for updates but that comes with a trade off in terms of latency, since networks calls would be required, and that can become detrimental in such a real-time based product.

Investigating that trade off or even developing a solution for easy deployment of model updates is another import development to be worked in the future.

4.4.3 Batteries and charging

Another challenge identified for creating a viable product is to develop and energy efficient solution that is capable of running on a reasonable battery for at least an entire day.

As described in the testing section, the prototype is already capable of being deployed with a reasonable commercial battery but we believe that there are still margin for improvement.

Evidently, it is possible to include a battery with bigger capacity to the product to provide better battery life but that comes with the trade off of additional weight and cost, undesired characteristics from the end user and grosser perspective respectively.

Additionally, it would be important to developed a practical mechanism to allow the carts to be charged such as a docking mechanism or even by wireless charging (TREFFERS, 2015), reducing the maintenance effort from the grocery's perspective.

4.4.4 Loss Prevention

As our research has shown, loss prevention is a key feature of a smart cart, specially in the Brazilian context (NEXTOP, 2022).

In such scenario, it would be important to work on possible extra features that would give the grosser the extra confidence to deploy the cart to his/her retail chain.

4.4.5 Improving accuracy and reliability

Related to the subsection above, improving the accuracy and reliability of the overall system is key not just for loss prevention but to provide a great user experience. We believe that a sub par experience will eventually lead to disuse and therefore our objective would be to achieve a *transparent* experience, where the user might even forget about all the technological feat that allows the cart to function.

5 CONCLUSIONS

This work has shown the development of *zCart*, a functional smart cart prototype using computer vision and sensor data, the same technological framework used on similar commercial products. The prototype development has successfully reached all of the desired objectives including the training of a object detection model and developing an interactive user interface.

The real time performance achieved using the selected hardware was sufficient for our purposes, at around 4 FPS with the EfficientDet-D2 model, which showed an average precision above 80% during validation.

Additionally, we were able to present some of the challenges and future work discussions related to the current state of the prototype and also its cost structure.

BIBLIOGRAPHY

- ABRAS. **Superhiper - Ranking ABRAS 2022**. 2022. Disponível em: <https://superhiper.abras.com.br/pdf/280.pdf>. Acesso em: October, 11th 2022.
- AMAZON. **Amazon Rekognition: Developer Guide**. 2022. Disponível em: <https://docs.aws.amazon.com/rekognition/latest/dg/images-displaying-bounding-boxes.html/>. Acesso em: December, 4th 2022.
- AMAZON. **What Are Alexa Skills?** 2022. Disponível em: <https://developer.amazon.com/en-US/alexa/alexa-skills-kit>. Acesso em: October, 10th 2022.
- ARAKI, H.; OMATU, S. 6 - artificial olfactory sense and recognition system. *In*: NGO, T. D. (Ed.). **Biomimetic Technologies**. Woodhead Publishing, 2015, (Woodhead Publishing Series in Electronic and Optical Materials). p. 121–139. ISBN 978-0-08-100249-0. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780081002490000069>.
- AVIA. **24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales**. 2022.
- AWS. **What is cloud computing?** 2022. Disponível em: <https://aws.amazon.com/what-is-cloud-computing/>.
- BRASIL, I. **Predictions Brazil 2021**. 2021. Disponível em: https://www.idclatin.com/2021/events/02_04_br/ppt.pdf. Acesso em: December, 17th 2022.
- CAPER. **The Rist of Smart Cart**. 2020. Disponível em: <https://www.caper.ai/post/the-rise-of-smart-carts>. Acesso em: October, 11th 2022.
- CAPGEMINI. **Smart Stores: Rebooting the retail store through in-store automation**. 2020. Disponível em: <https://www.capgemini.com/wp-content/uploads/2020/01/Report-%E2%80%93Smart-Stores-1.pdf>. Acesso em: October, 11th 2022.
- CHEN, K. J. *et al.* Gan-on-si power technology: Devices and applications. **IEEE Transactions on Electron Devices**, v. 64, n. 3, p. 779–795, 2017.
- CORPORATION, M. S. **What is a Strain Gauge and How does it work?** 2020. Disponível em: <https://www.michsci.com/what-is-a-strain-gauge/>. Acesso em: November, 6th 2022.
- COURSERA. **Types of Machine Learning**. 2022. Disponível em: <https://www.coursera.org/articles/types-of-machine-learning>. Acesso em: November, 20th 2022.
- DEEPAI. **Hidden Layer**. 2022. Disponível em: <https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning>. Acesso em: November, 19th 2022.
- DUCKETT, J. **HTML and CSS: Design and build websites**. 1. ed. [S.l.]: Wiley, 2011.
- FIELDING, R. T. Architectural styles and the design of network-based software architectures. 2000. Acesso em: October, 16th 2022.
- FLANAGAN, D. **JavaScript: The Definitive Guide**: Master the world's most-used programming language. 7. ed. [S.l.]: O'Reilly, 2020.
- GAO, Y. *et al.* Alexa, my love: Analyzing reviews of amazon echo. *In*: **2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable**

Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). [S.l.: s.n.], 2018. p. 372–380.

GRUS, J. **Data Science from Scratch: First principle with python.** 2. ed. [S.l.]: O'Reilly, 2019.

HBM. **What is a load cell? How does a load cell work?** 2022. Disponível em: <https://www.hbm.com/en/6768/what-is-a-load-cell-and-how-does-a-load-cell-work/>. Acesso em: November, 6th 2022.

HOROWITZ, P.; HILL, W. **The Art of Electronics.** 3. ed. [S.l.]: Cambridge University Press, 2015.

IBM. **Deep Learning.** 2020. Disponível em: <https://www.ibm.com/cloud/learn/deep-learning>. Acesso em: November, 19th 2022.

IBM. **Neural Networks.** 2020. Disponível em: <https://www.ibm.com/cloud/learn/neural-networks>. Acesso em: November, 19th 2022.

INTELLIGENCE, I. **A not-so-smart rise in smart speaker ownership.** 2022. Disponível em: <https://www.insiderintelligence.com/content/smart-speaker-ownership>. Acesso em: October, 10th 2022.

JORDAN, J. **An overview of object detection: one-stage methods.** 2018. Disponível em: <https://www.jeremyjordan.me/object-detection-one-stage/>. Acesso em: December, 4th 2022.

JOUPPI, N. *et al.* Motivation for and evaluation of the first tensor processing unit. **IEEE Micro**, v. 38, n. 3, p. 10–19, 2018.

KANTAR. **Winning Omnichannel.** 2022. Disponível em: <https://kantar.turtl.co/story/winning-omnichannel-2022-c>. Acesso em: October, 11th 2022.

KONG. **The API Mandate: How a mythical memo from Jeff Bezos changed software forever.** 2022. Disponível em: <https://konghq.com/blog/api-mandate>. Acesso em: October, 16th 2022.

KUROSE, J.; ROSS, K. **Redes de computadores e a Internet: Uma abordagem top-down.** 6. ed. [S.l.]: Pearson, 2013.

LI, F.-F. **Stanford University - Deep Learning for Computer Vision Class Notes.** 2022. Disponível em: <https://cs231n.github.io/transfer-learning/>. Acesso em: November, 24th 2022.

MCGONAGLE, J. *et al.* **Backpropagation.** 2022. Disponível em: <https://brilliant.org/wiki/backpropagation/>. Acesso em: November, 19th 2022.

MCKINNEY, W. **Python for Data Analysis: Data wrangling with pandas, numpy, and ipython.** 2. ed. [S.l.]: O'Reilly, 2017.

MUELLER, A.; GUIDO, S. **Introduction to Machine Learning with Python: A guide for data scientists.** 1. ed. [S.l.]: O'Reilly, 2016.

NEWMAN, S. **Building Microservices: Designing fine-grained systems.** 2. ed. [S.l.]: O'Reilly, 2021.

NEXTOP. **Smart Cart - Carrinho Inteligente.** 2022. Disponível em: <https://nextop.com.br/smart-cart-carrinho-inteligente/>. Acesso em: October, 12th 2022.

NIELD, T. **Introdução à Linguagem SQL: Abordagem prática para iniciantes.** 1. ed. [S.l.]: Novatec, 2016.

NIELSEN, M. A. **Neural Networks and Deep Learning**. [S.l.]: Determination Press, 2015.

PARANJAPPE, J. N.; DUBEY, R. K.; GOPALAN, V. V. Exploring the role of input and output layers of a deep neural network in adversarial defense. p. 5, 06 2020.

PHDATA. **What is the cost to deploy and maintain a machine learning model?** 2021. Disponível em: <https://www.phdata.io/blog/what-is-the-cost-to-deploy-and-maintain-a-machine-learning-model/>. Acesso em: October, 10th 2022.

POST, T. W. **Smart shopping carts on the rise as stores adapt to pandemic era**. 2021. Disponível em: <https://www.washingtonpost.com/technology/2021/01/29/smart-shopping-carts-pandemic-innovations/>. Acesso em: October, 11th 2022.

PRICE, J. *et al.* **Stochastic gradient descent**. 2020. Disponível em: <https://optimization.cbe.cornell.edu/>. Acesso em: November, 19th 2022.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, p. 386–408, 1958.

ROSS, R.; BAJI, A.; BARNETT, D. Inner profile measurement for pipes using penetration testing. **Sensors**, v. 19, p. 237, 01 2019.

SHAFIQUE, K. *et al.* Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. **IEEE Access**, v. 8, p. 23022–23040, 2020.

SILBERSCHATZ, A.; FORTH, H.; SUDARSHAN, S. **Sistemas de Bancos de Dados**. 2. ed. [S.l.]: GEN LTC, 2010.

TAN, M.; PANG, R.; LE, Q. V. Efficientdet: Scalable and efficient object detection. p. 10, 07 2020.

TANENBAUM, A.; BOS, H. **Sistemas Operacionais Modernos**. 4. ed. [S.l.]: Pearson, 2015.

TOTAL, P. **Enxuto supermarket chain launches 1st smart cart in Latin America**. 2022. Disponível em: <https://www.paraibatotal.com.br/2022/05/05/rede-de-supermercados-enxuto-lanca-1o-carrinho-inteligente-da-america-latina/>. Acesso em: October, 12th 2022.

TREFFERS, M. History, current status and future of the wireless power consortium and the qi interface specification. **IEEE Circuits and Systems Magazine**, v. 15, n. 2, p. 28–31, 2015.

WILBER, J.; WERNESS, B. **The Importance of Data Splitting**. 2022. Disponível em: <https://mlu-explain.github.io/train-test-validation/>. Acesso em: November, 24th 2022.




YOUNG, T. *et al.* Recent trends in deep learning based natural language processing [review article]. **IEEE Computational Intelligence Magazine**, v. 13, n. 3, p. 55–75, 2018.

ZHAO, Z.-Q. *et al.* Object detection with deep learning: A review. 4 2019.

ȘTEFĂNESCU, D. M. Strain gauges and wheatstone bridges — basic instrumentation and new applications for electrical measurement of non-electrical quantities. *In: Eighth International Multi-Conference on Systems, Signals & Devices*. [S.l.: s.n.], 2011. p. 1–5.






APPENDIX A – User App Screenshots

Figure 55 – Product listing and subtotal

🛒 zCart		
	Leite Longa Vida 1L 1 items	R\$ 3,99
	Tang 5 items	R\$ 1,99
	Chamyto 4 items	R\$ 5,99
	Café 5 items	R\$ 9,99
\$ Checkout		Subtotal: R\$ 194,85

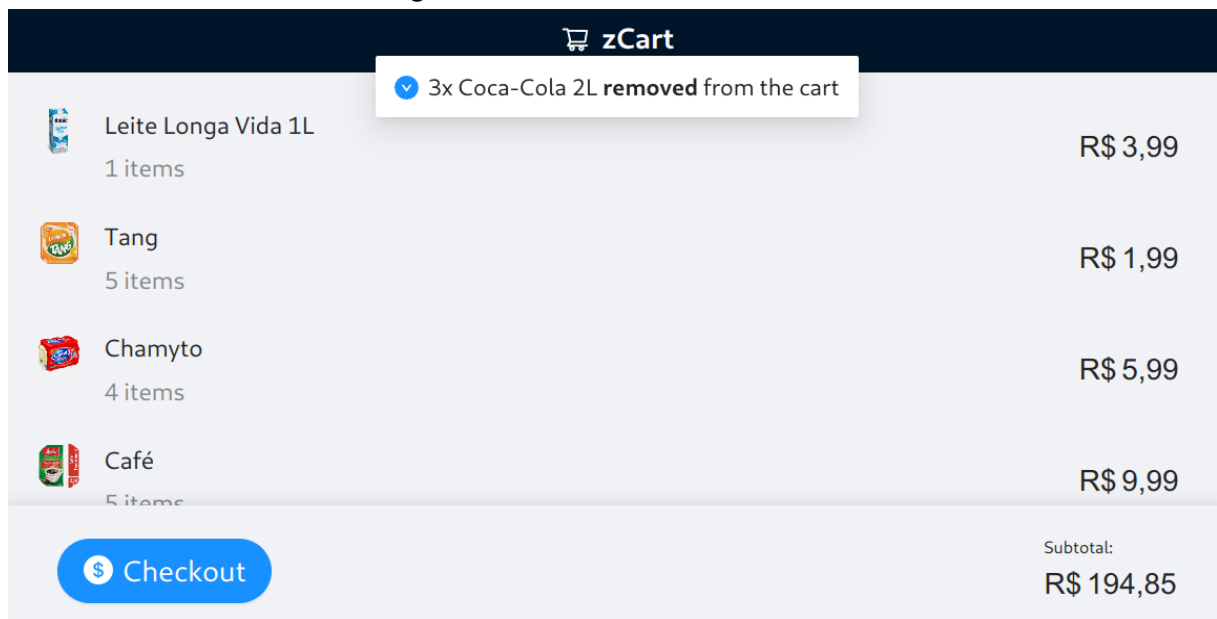
Source: Own work (2022).

Figure 56 – Item addition notification

🛒 zCart		
<div>  2x Leite Longa Vida 1L added to the cart </div>		
	Leite Longa Vida 1L 1 items	R\$ 3,99
	Tang 5 items	R\$ 1,99
	Chamyto 4 items	R\$ 5,99
	Café 5 items	R\$ 9,99
\$ Checkout		Subtotal: R\$ 194,85

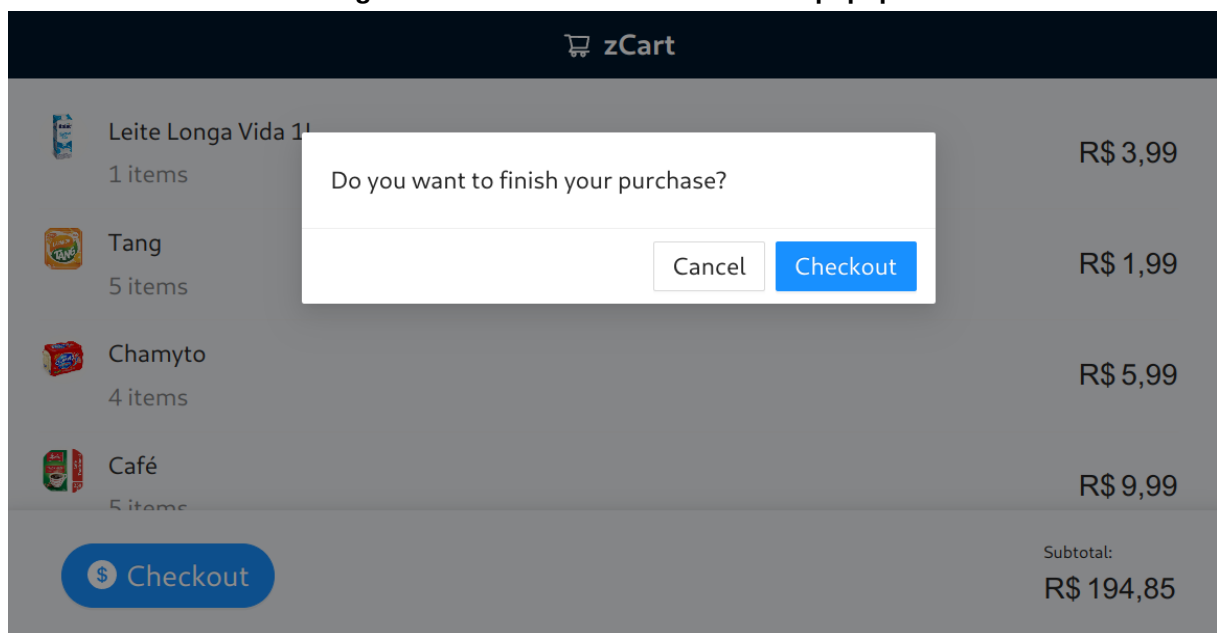
Source: Own work (2022).

Figure 57 – Item removal notification



Source: Own work (2022).

Figure 58 – Pre-checkout confirmation popup



Source: Own work (2022).

Figure 59 – Post checkout screen



Thank you for you purchase!

Buy Again

Source: Own work (2022).

APPENDIX B – Prototype Photos

Figure 60 – Frontal view of the prototype



Source: Own work (2022).

Figure 61 – Top view displaying the products in the cart



Source: Own work (2022).

Figure 62 – Frontal view of the prototype displaying the video camera feed



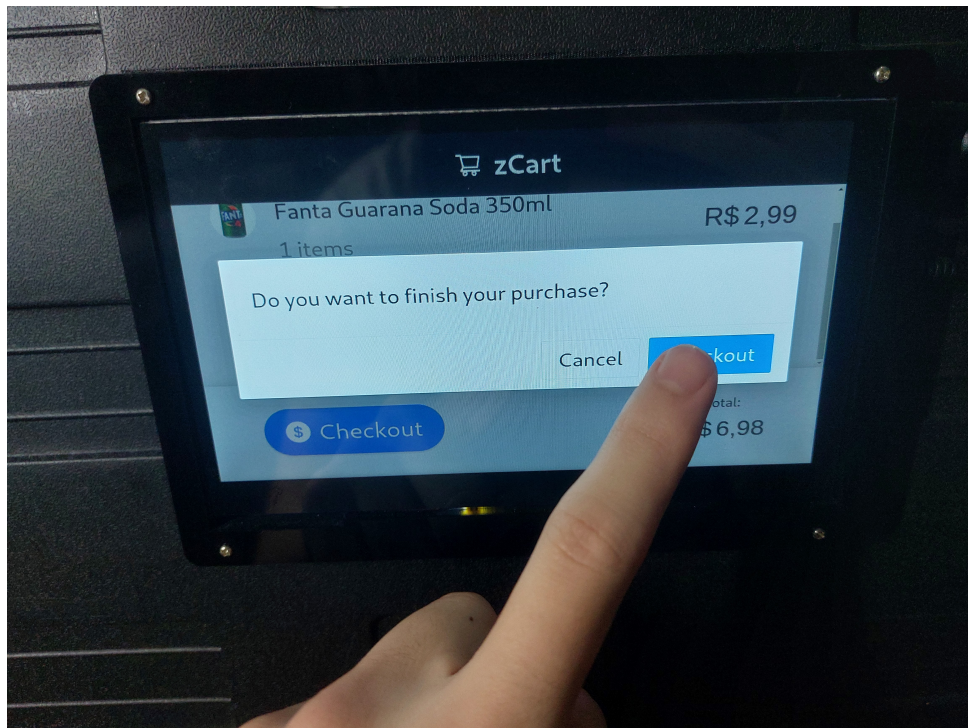
Source: Own work (2022).

Figure 63 – Frontal view of the prototype displaying the video camera feed



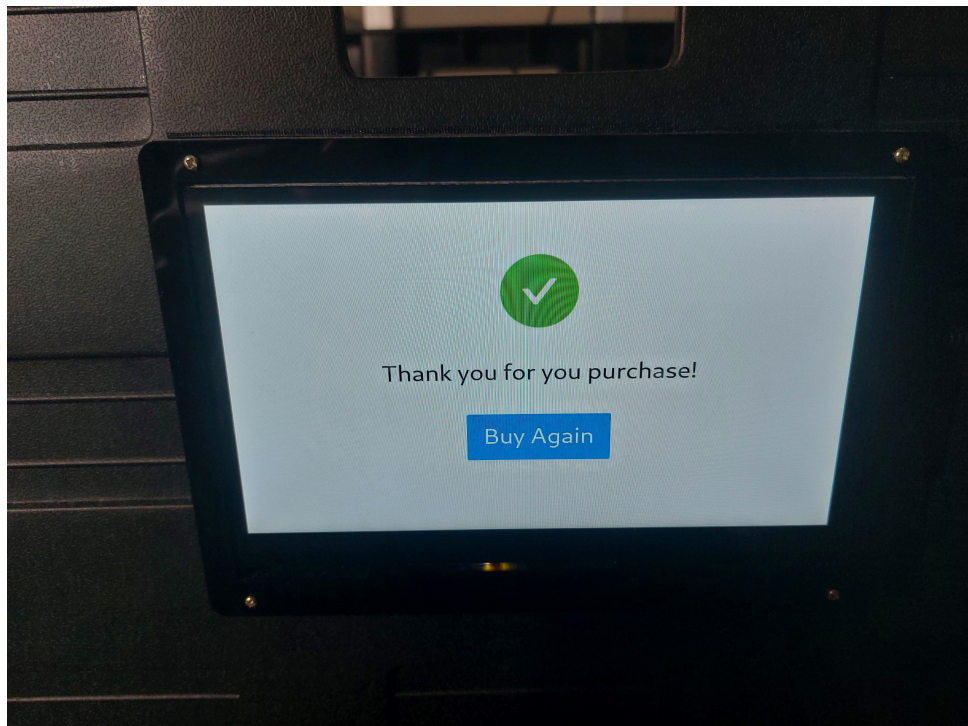
Source: Own work (2022).

Figure 64 – Popup confirmation before finishing the purchase

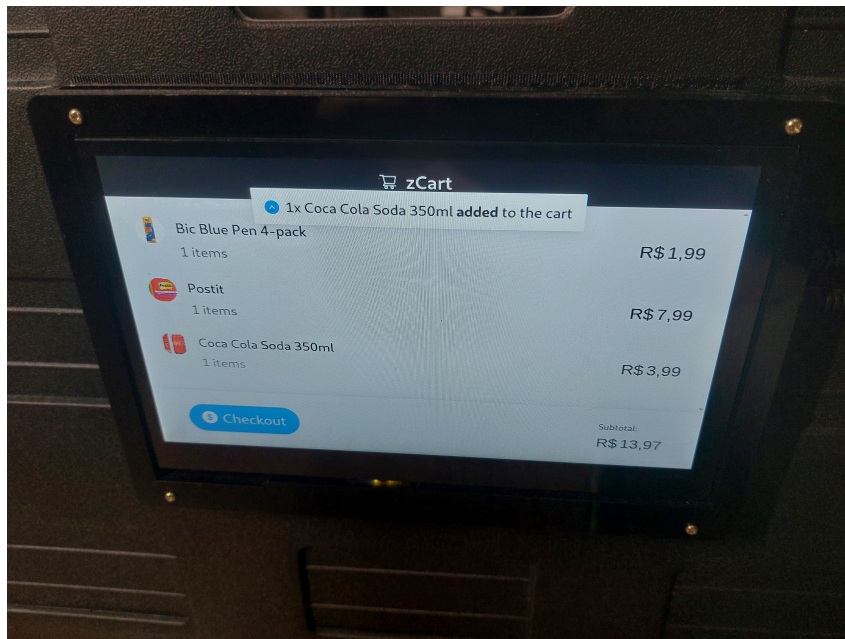


Source: Own work (2022).

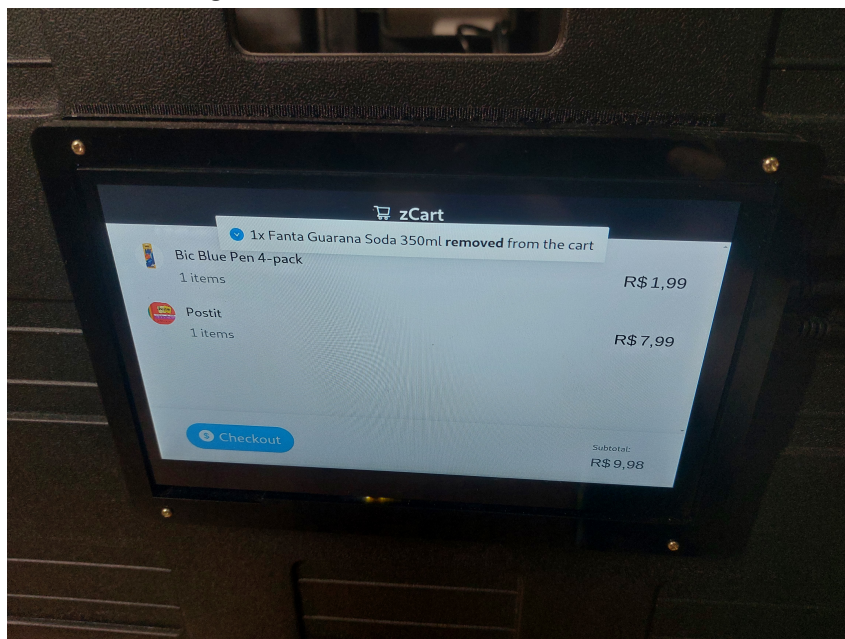
Figure 65 – Checkout confirmation screen



Source: Own work (2022).

Figure 66 – Product addition notification

Source: Own work (2022).

Figure 67 – Product removal notification

Source: Own work (2022).