Master's thesis

Master's degree in Industrial Engineering
(Automatic Control specialization)

Recognition of supermarket products using Computer Vision

REPORT

Author: Jaume Ribas Fernández
Supervisor: Bruno Siciliano (DIETI, Università degli Studi di Napoli Federico II)
Co-Supervisor: Riccardo Caccavale (DIETI, Università degli Studi di Napoli Federico II)

Summons: July 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona
and
Dipartimento di Ingegneria Elettrica e
delle Tecnologie dell'Informazione


DIETI. UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Abstract

Automatizing processes is one of the common current improvements implemented in companies which take advantages of the latest technologies to be competitive in the market. Using Computer Vision techniques is a way to reach it without investing a great amount of money, that directly implies, in short term, production increase and, in long term, earnings. For this reason, the current project develops a Computer Vision algorithm, taking advantage of Machine Learning techniques, to depalletize a supermarket cases pallet. In fact, it aimed to improve an algorithm already developed, but due to the sanitary emergency for Covid-19 pandemic, it has been harder than expected to work on it.

With the commented objective in mind, a brief research about what is Computer Vision is done, about its history and all the applications that are possible to find nowadays. In the same way, this is done for Machine Learning and Deep Learning, and the techniques usually applied in projects such the current one. Once understood these techniques and learned about the problem to be solved, some different algorithms are put into practice until the final approach is developed.

Finally, challenges encountered during the realization of this project are commented, and also the future work that may be done through multiple ways to improve the algorithm. Also it is commented a bit the planning and programming of the project, the cost estimation and the possible environmental impact.

Contents

1. Introduction

1.1. Context and motivation

Nowadays, the use of Computer Vision is widely spread, but it is just at the beginning of his expansion. Trying to take advantage of its power and knowing how to apply it is a great challenge. The multiple applications that can be imagined are enormous, and for this motive it is reasonable to think that in some years it will be at the top of its development.

In the same way, the well-known company Gartner publishes every year some graphics called Hype Cycle, which include the most important technology trends at the moment. As seen in the Figure 1.1, Computer Vision, that right now is going to the Trough of Disillusionment where the success expectations are low, is expected to reach the Plateau of Productivity between two and five years. What it does mean is that after this period of time, it is estimated that this technology will be very developed and used in many fields of our society, and the expectations will be stabilized at a medium level. At the same time, techniques used during this project, like Machine Learning and Deep Neural Networks are in a similar situation to Computer Vision.

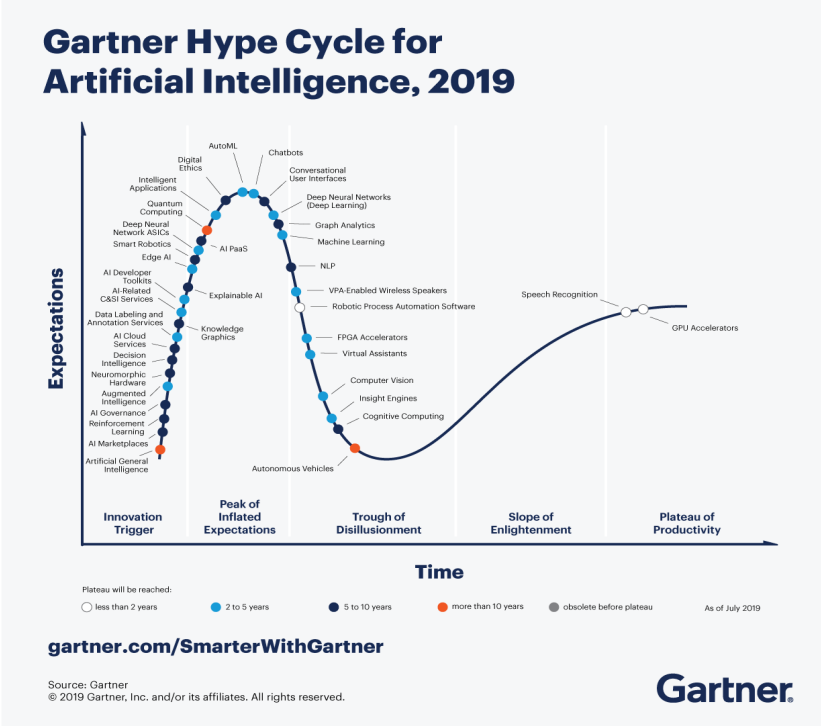

Figure 1.1. Gartner Hype Cycle for Artificial Intelligence of the last year, 2019. Source: [1].

Due to the enormous possibilities that Computer Vision techniques provide, and how they can be applied in anything we can think of, it is smart to import them to the industry sector, to

automate processes and make them faster. The comparative advantage of using them can be easily seen with money savings. Firstly, by reducing human resources, and secondly, by increasing the production.

For these reasons, in the current project, Computer Vision will be applied in a depalletizing robot for supermarket cases. The process consist in: having a pallet with cases stacked, extracting them from there and classifying them in other places based on the type of case. This procedure can be found in many industries, and more frequently in those working on logistics. For this reason, it is so important to automate it as much as possible.

More precisely, the process is as follows. A RGB-D camera gets the image of the pallet, that is used as input to the algorithm. This camera is located at the front and bottom part of the depalletizing robot. Once the image is obtained, it is processed by the algorithm discussed in this work. After this, with the cases recognized and located, the information is passed to another algorithm which is responsible of moving the robot to take the cases and situate them correctly.

Therefore, applying Computer Vision techniques to this process will be the focus of the project, for the mentioned reasons above.

1.2. Project goal and outline

The goal of this project is to improve the algorithm already developed, which is not fully implemented. Indeed, there is an algorithm created by the Prisma Lab, a laboratory research group in DIETI (Department of Electrical Engineering and Information Technology) at University of Naples Federico II, but the results obtained are not good enough, mainly in the feature extraction part, which will be more explained later. For this reason, another algorithm will be developed from scratch, but taking into account the steps followed by the previous one. Comparing different techniques and models frequently used in Computer Vision problems, and checking the results obtained will be the method to reach this goal.

More specifically, the purpose of the algorithm is to make the detection, recognition and localization of different types of cases located in a pallet, as it can be seen in the Figure 1.2. This is, from the database which contains photos of all the faces of the cases, and having an input image, recognize those cases existing in the database and provide their localization in the space of the image.

Figure 1.2. Pallet with stacked cases. Source: Prisma Lab.

Due to the robot restrictions, only the cases located in the front part of the pallet are the ones that will be picked, so those are the important ones for the algorithm. Similarly, those cases which do not have some kind of reference to the product contained in the external surfaces are discarded. In the same way, all the cases that are on the pallet are supposed to be in the database, otherwise those will not be detected and cannot be picked.

Besides that, it was expected, if feasible, to implement the same algorithm for different kind of supermarket products, but because of the sanitary emergency for Covid-19 pandemic it has been impossible. Additionally, this emergency has not made possible to take more pictures of the cases, so the dataset considered is small and this has affected the way to choose the existing techniques to use. Also, this makes more difficult to completely assess how well the algorithm works.

1.3. Literature review

The problem discussed in this work is common in Computer Vision, and it has been solved in different ways in the past. Consequently, this part of the problem has already been done by the research group of the Prisma Lab. Hence, it will be only commented some of the paths followed in different solutions.

Firstly, one of them [2] is based on a laser and 2D images, but with the drawback that all the cases must be the same dimension. Despite this, it is an effective and robust solution, and it is invariant to the lightning conditions. In a similar way, there is a 3D-vertices detection

algorithm [3], using edges detection and robust line fitting, together with a time of flight laser sensor for data acquisition, which is also independent of lightning conditions.

To continue, there is also an algorithm [4] to palletize, which extracts the pose and dimensions of cargo boxes in large scale workspaces. It uses multiple RGB-D Kinect sensors, and it has a high accuracy. More related to depalletizing, improving the speed and the robustness of the process is the goal in [5], where some multi-resolution surfel models are used to detect and localize objects.

In [6], to decrease the cost of depalletization, PMD (low-cost photonic mixing device) technology is combined with a predetermined model of loading situations. Only focusing on the pallet, there is an algorithm [7] to detect, localize and track pallets using Neural Networks, a Deep Learning technique that will be discussed later, based on an on-board 2D laser rangefinder. The last algorithm [8] considered works with those packages that do not have some drawings related to the product contained in the external surfaces. It is based on planes extraction and border detection from different kind of images.

On the other hand, regarding the algorithm developed by the Prisma Lab, it divides the problem in three parts. The first one focuses on the detection of the cases, based on the detection of the features, an image segmentation and, afterwards, a detection of the faces. The second one is related to the geometry, and does the matching and the localization of the cases. The last one deals with the cases database.

Finally, about the specific techniques applied, the Scale Invariant Feature Transform (SIFT) algorithm has been used to achieve the detection part. At the same time, the image segmentation has been done using the Watershed Transform. These, together with mathematical models and methods, like the Random Sampling and Consensus (RANSAC), build the existing algorithm.

The weakest part of this algorithm is the feature detection, and this is why it is necessary to improve it.

2. Computer Vision

2.1. Introduction to Computer Vision

Computer Vision is a field of computer science, more specifically of Artificial Intelligence, focused to help computers to "see" [9]. In other words, it tries to make computers understand digital images and videos, enabling to work with them and even interact. This task can seem very easy for humans, due to our complex system we employ to do it, but it is not. The main problems are the lightning conditions, the occlusions or the orientations of the objects, and our system deals with all of this seamlessly, but the biggest challenge is trying to reproduce it into computers. One of the greatest attempts are the Neural Networks, which can achieve remarkable results.

In the context of Artificial Intelligence, which is responsible to make the machines more "human", this is, to make them capable of performing human tasks, Computer Vision takes care of the vision tasks. To do it, it takes advantages of techniques of other fields of Artificial Intelligence like Machine Learning or Deep Learning.

These techniques evolve and are more powerful every day, in part thanks to the improvement of electronics too, which is needed to deal with the enormous amount of data generated. For this reason, the limit of Computer Vision, and Artificial Intelligence in general, nowadays is unimaginable. In recent years, the accuracy of some identifying objects algorithms has grown from 50% to 99% [10], which actually makes them more useful than humans in some tasks.

2.2. History of Computer Vision

The first progress towards Computer Vision was in the 1950s [11], with some algorithms similar to what now are called Neural Networks. Later, in the 1960, Larry Roberts, who is known as the father of Computer Vision, but also as the founder of internet, published his book called "Machine Perception Of Three-Dimensional Solids", which is considered the basis of the field of Computer Vision [12]. He described how to get 3D models from 2D images.

At the same time, a professor of the Massachusetts Institute of Technology, Marvin Minsky, and one of his students, Gerald Sussman, developed a project "to connect a camera to a computer and get the machine to describe what it sees" [13], something which is actually not achieved yet.

Later, in 1979, Kunihiko Fukushima, a Japanese scientist, built the Neocognitron, a system that despite not working as expected, set the fundamentals for the Convolutional Neural

Networks, which were developed some years afterwards. This was a great step in the field of Computer Vision. Since then, the techniques have evolved fast until now, to the extent that anyone can use some algorithms implementing them, for example, in cell phones. For instance, the vast majority of smartphones implement some sort of face recognition algorithm, a clear application of Computer Vision.

2.3. How does Computer Vision work?

Computer Vision is based on the recognition of patterns and learning techniques, and to do it, it is necessary to train the computer using as many images as possible. This is, with the suitable algorithm, it can learn some patterns when the training images are analyzed. Once these patterns are obtained, the algorithm is ready to work on new images.

The steps followed while developing a Computer Vision algorithm try to reproduce those steps the humans do to understand the world perceived through the eyes. Firstly, some stimulus are received, secondly, these stimulus are processed and, finally, understood. In general, this can be extrapolated to Computer Vision as shown in Figure 2.1.

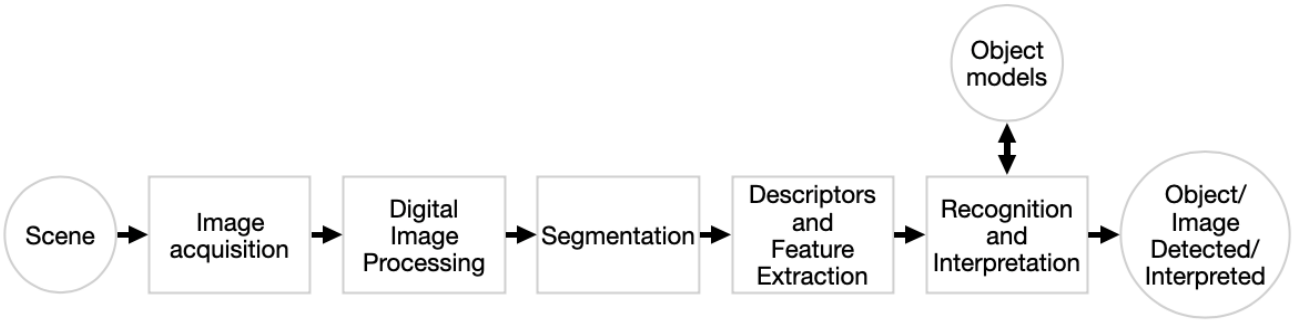

Figure 2.1. Computer Vision stages. Source: adapted from Perception Systems lectures of Master's degree in Industrial Engineering (Automatic Control specialization) at Universitat Politècnica de Catalunya.

At the beginning, some images are acquired. Then, these images are digitally processed to be able to get information from them. It is important to emphasize that these two phases are crucial to later develop an algorithm that works properly. Once done, the next step is the segmentation, followed by the feature extraction. With this, using different models, it is possible to make the detection, classification or recognition of new images to get the desired result at the end.

Within these steps there are different paths to pursue, many kinds of techniques developed throughout the years of use in several problems. The key to get the best algorithm is to know when to apply each of these techniques. Depending on the project, usually, one of them fits better. However, in some cases, it is necessary to check multiple techniques because it is not

clear beforehand which one is more suitable. Hence, those that may be suitable for the present project will be discussed later.

Some of the main tasks of Computer Vision are about classification, identification, detection, recognition and segmentation. They differ on the goal that is wanted to reach. For example, if it is only about knowing what kind of object is in the image, then it is an Object Classification problem. If it is required to know also the localization, it becomes an Object Recognition problem. This is indeed the scenario in the current project, with the addition that it is not just for one but multiple objects. Hence, it can be framed as an Object Detection problem, but is not solved as it would be expected due to some issues that later will be commented. In the Figure 2.2 it can be seen the different types of problems commented, with an extra one called Instance Segmentation, which can be ignored at this point as an Object Detection approach is enough for the current project.

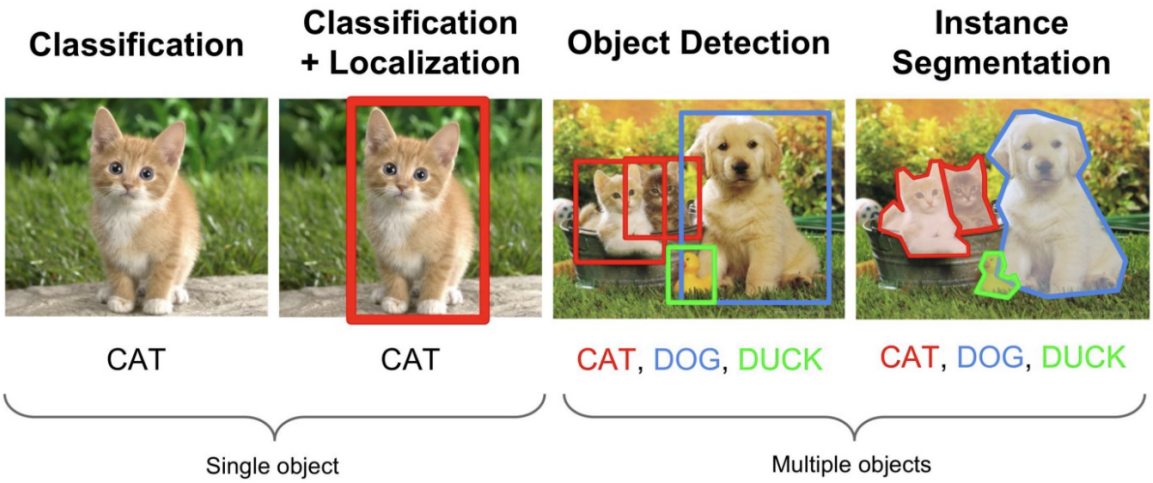

Figure 2.2. Comparison between Object Classification, Object Recognition, Object Detection and Instance Segmentation problems. Source: [14].

In the following chapters, techniques mainly used to develop Computer Vision algorithms that could be applied in this project will be explained. These can be divided in two types: Machine Learning and Deep Learning.

2.4. Applications

The use of Computer Vision algorithms is increasing every day, and it is not an unreachable technology for major part of the people. In fact, it is already in our hands, in a literal sense. As said before, cell phones have some algorithms integrated such that to unlock the phone with the face or to recognize faces in photographs, as seen in Figure 2.3.

Figure 2.3. Face recognition application for cell phones. Source: [15].

Another sector which takes advantage of this technology is the automotive industry, in particular for the self-driving cars [16]. In the same way, manufacturing industry uses it to detect defects more quickly. Furthermore, in hospitals it is used to diagnose cancers [17]. These are just some of the multiple examples where Computer Vision is applied.

3. Machine Learning

3.1. Introduction to Machine Learning

This is a field of Artificial Intelligence, which is responsible of giving to the computers the ability to learn by themselves [18]. It means that, giving some rules with an algorithm, the computer is able to learn from a data training set, and subsequently, predict the behavior of the system being studied.

It begins by getting information from the training set, searching for some kind of patterns that relate the samples between them [19]. The more data provided to the algorithm, the better it will work, but also the more complex and hard to compute it will be. Once learned, it is common to check its effectiveness against some extra validation data. Finally, it is able to make estimations on new data.

3.2. How does Machine Learning work?

The key of this science is that it enables computers to predict, while be indirectly programmed [20]. This is, without previously knowing the most important information of the data neither the system generating it, it is able to replicate it and use it as some sort of rules. In other words, it can be thought of as black box, where with the rules described and using the training data, it is possible to get the corresponding outputs correctly, usually without really knowing the underlying system.

These rules are based on statistical methods, but trial and error methods too [21]. The goal is to optimize some sort of measure (loss function) that evaluates its performance Through these methods, this measure should improve while the machine is learning, until it reaches a convergence point. This is, until the algorithm has reached an optimal point, which can be either local or global, depending on the algorithm used.

Due to the improvements in technologies and the increasing capacity to make calculations, the use of this science has grown a lot in the last decade, and it will be more used in the future yet. This is because, with high capacity to compute and with big amounts of data (known as Big Data) mostly available in recent years, better algorithms will be developed and better predictions will be achievable [22].

In general, most important Machine Learning techniques can be divided in two types: supervised learning and unsupervised learning [23]. The first one is based on known input and output data, and tries to predict outputs of new inputs. Two main techniques used in supervised learning are Classification and Regression: depending on the responses, if they are discrete or continuous, respectively, as can be seen in Figure 3.1.

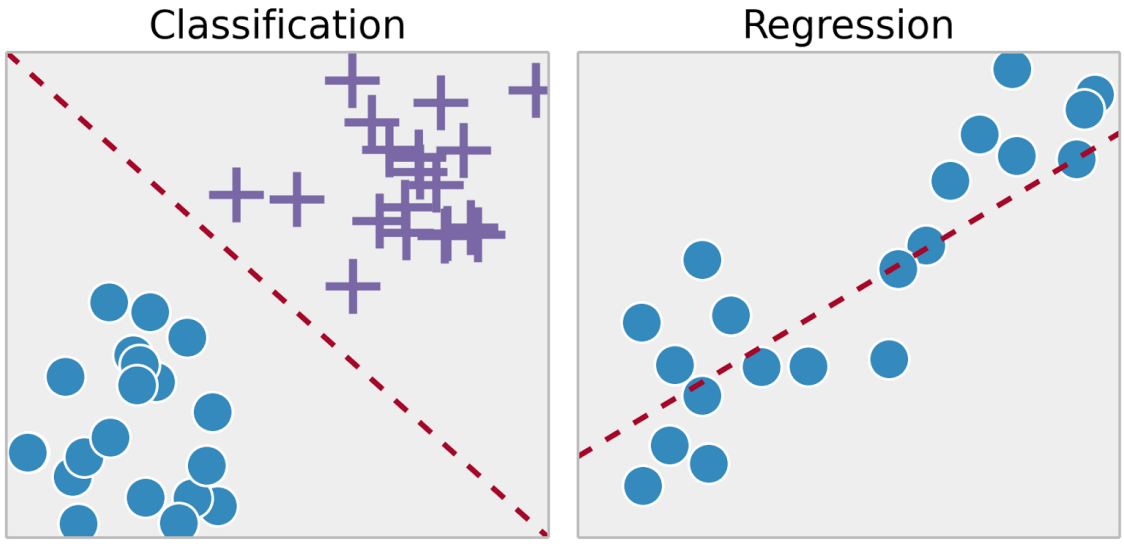

Figure 3.1. Comparison of the responses between Classification and Regression problems. Source: [24].

The second one is based on finding patterns in data, without labeled outputs. This is, try to find some kind of labels or structures within the data, as seen in Figure 3.2. The main technique used here is called Clustering, which can be implemented through different algorithms. The same happens with Classification and Regression.

Figure 3.2. Process of finding clusters in the data. Source: [23].

Because of the reasons explained before, Machine Learning is applied to solve many problems in different areas like Computational Finance, Computational Biology, Energy Production, Automotive, Aerospace, Manufacturing… and also Image Processing and Computer Vision. This is why it is important in this project, and coming next, some techniques that could be applied in the current problem will be described.

3.3. Machine Learning techniques useful for the current problem

On the one hand, there are useful algorithms for some of the steps of the Computer Vision workflow explained above. One of them are the Sliding Windows. As the name suggests, it is a technique based on a window that slides through the input image, over all positions and different scales, and allows to localize precisely where an object is. This is very important for an Object Recognition problem, and it is possible to combine it with an object classifier to know if there is an object there, and if so, to know also its class. A drawing of the technique can be seen in Figure 3.3, where it is explained for a single scale. The process can be then repeated for different scales of the image.

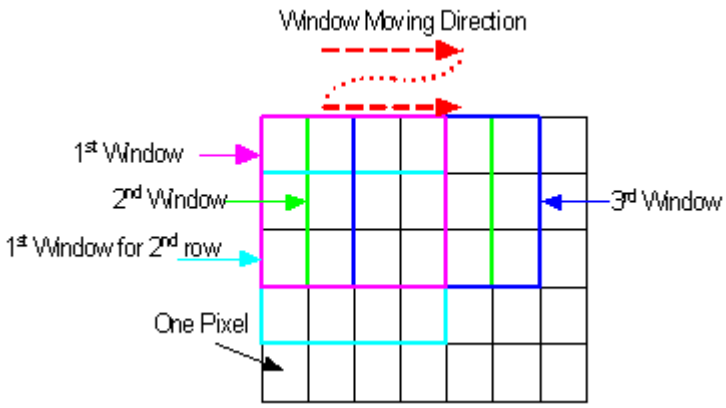

Figure 3.3. Representation of the Sliding Window movement over the pixels of an image. Source: [25].

Another algorithm that could be useful, in this case to do the classification, is called Support Vector Machine (SVM), which consists in finding the optimal hyperplane that divides the data points to distinguish all the classes. This results on a supervised learning model that assigns to a new input the class corresponding on the side where it is. To make this approach, it is needed before to extract features using another algorithm such as Scale-Invariant Feature Transform (SIFT) or Histogram of Oriented Gradients (HOG).

On the other hand, there are also the algorithms called Neural Networks, with many different types of possible architectures. They enclose what is known as Deep Learning. They are new methods, recently developed and increasingly used, due to their high power, that in many cases work really well and outperform more traditional methods. Coming next, it will be explained how Neural Networks work with more details, and its feasible application in the current project.

4. Deep Learning

4.1. Introduction to Deep Learning

As discussed above, Deep Learning is a subfield of Machine Learning which uses Neural Networks to try to reach the complete Artificial Intelligence. For this reason, these algorithms were designed to reproduce the human brain system [26]. Although it is hard to know how it really works, Neural Networks imitate as much as possible its functioning.

Deep Learning approach is based on multiple layers - which gives the word "*deep*" - and each of them learns different features with different levels of abstraction. Most of these models are trained by the Backpropagation algorithm, whose goal is to find the correct weights of the different features to obtain the best representation of the input data.

The importance gained last years by Deep Learning can be explained by the Figure 4.1, which shows how its performance increases with bigger amounts of data, unlike older learning algorithms, that end up reaching a plateau. This is a very relevant fact because of the great amount of data available nowadays, as commented before. At the same time, a high computing power is required, which is possible nowadays using graphics processing units (GPUs) thanks to their parallel architecture. For these reasons, Deep Learning algorithms can even surpass human performance in some tasks.

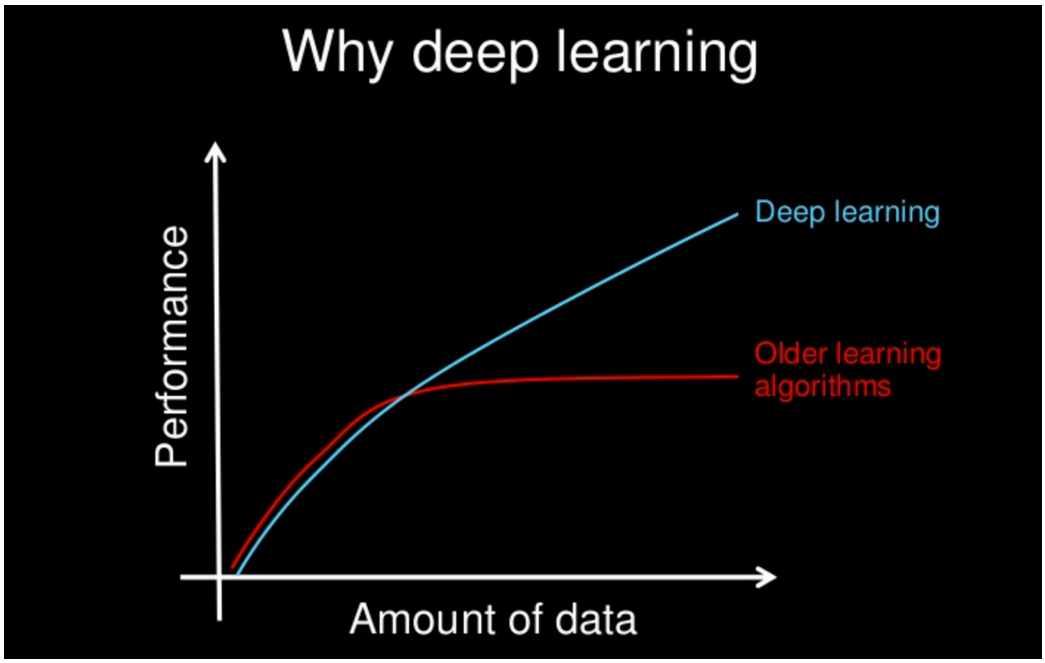

Figure 4.1. Comparison of the relation between the amount of data and the performance of the algorithm for Deep Learning and older learning algorithms. Source: [27].

With respect to Computer Vision, the main difference between Machine Learning and Deep Learning is that the second one usually does together the feature extraction and the classification, as seen in Figure 4.2. In other words, Neural Networks get automatically the most important information of the input to obtain an unknown model that is able then to classify new inputs and return outputs properly. In some cases, Neural Networks are only used to make the feature extraction and then some classifier that can handle great amounts of data, like a SVM, is applied.

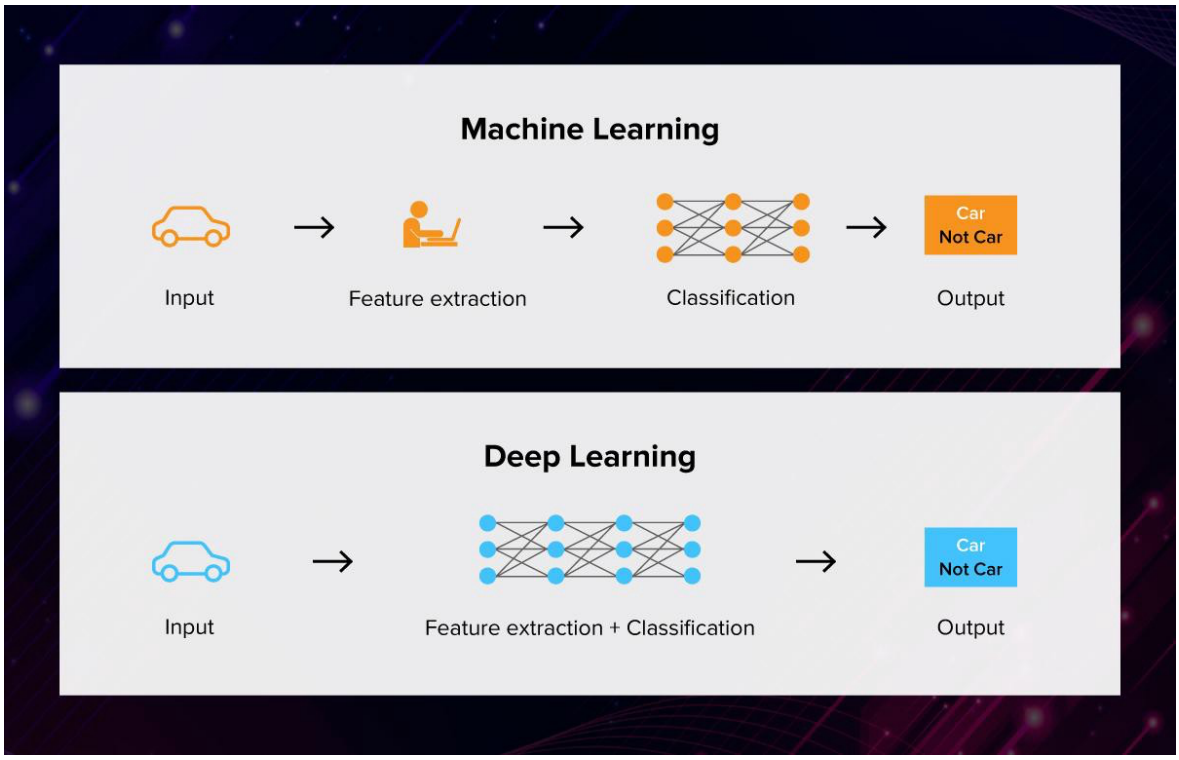

Figure 4.2. Comparison of the stages followed by Machine Learning and Deep Learning. Source: [26].

On the whole, in comparison with Machine Learning, it is better to use Deep Learning algorithms when a large set of data and high computer power are available.

4.2. How does Deep Learning work?

Understanding completely how Deep Learning algorithms work may be quite difficult, but here it is presented a general idea of it. It is based on two elements: a great amount of data and Neural Networks architectures. These architectures, as said above, consist in multiple layers connected between them. Inside each layer there are some nodes, which, for the networks most commonly used, called Fully Connected, are connected with all the nodes of the precedent layer and the next too, what constructs a net structure as can be seen in

Figure 4.3. These nodes, called neurons, receive their name due to their similarity with the brain neurons, which receive inputs and give some outputs accordingly. For these reasons, these algorithms are known as Neural Networks.

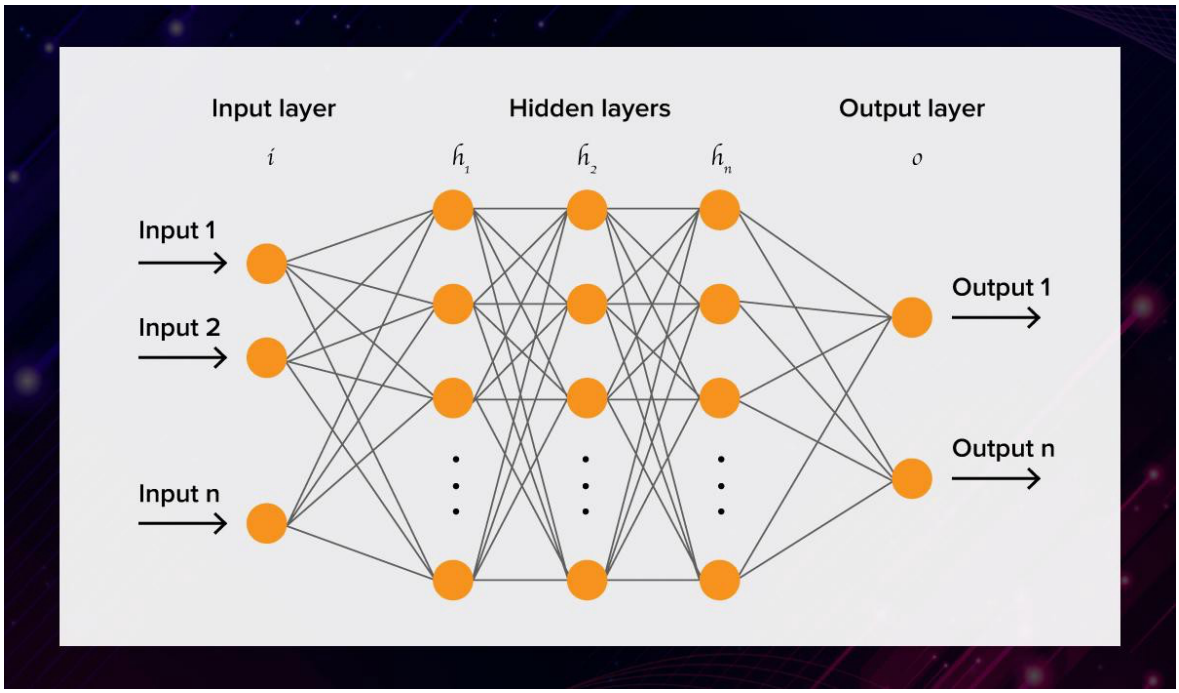

Figure 4.3. General architecture of a Neural Network. Source: [26].

The connections made depend on the weights that are computed through the training data to get the model. The output of each node is obtained by an activation function, that receives the sum of the weighted values of precedent neurons and also a number, different for each layer, called bias. If the function is activated, then information is passed through it to the following layer. Because of this function, it is possible to get nonlinear models.

There are different types of activation functions, and the use of them depends on the goal that is wanted to achieve. For example, the Sigmoid function is mainly used for problems were a probability of something in the output is required. This is because this function outputs values in the range of 0 and 1, as a probability. In the same way, it is possible to choose the number of “hidden layers” and also the number of neurons for each of them. In other words, the number of neurons of the input layer, the first one, is given by the size of the input data. The last layer has the size also defined by the desired outputs. Then, the other layers, called “hidden layers”, are those susceptible to be selected by the user.

The training of a Neural Network is achieved using the Backpropagation algorithm together with Stochastic Gradient Descent with the goal to reduce the value of a cost function. This function computes indirectly, in the case of supervised learning, the difference between the output of one value and the calculated with the Neural Network. So, when this function gets

closer to zero the accuracy improves, because it means that the algorithm calculates correctly the desired output. To achieve this, the data is iterated to obtain the weights that fit better. When starting the training, it is important to initialize these weights randomly, to avoid the algorithm getting stuck. All of this has some complex mathematics behind [28], but it is feasible to use Deep Learning without understanding it completely.

Moreover, it is possible to apply Deep Learning in two ways: training a model from scratch or using transfer learning [29]. The first case is often used for new problems and when a huge amount of data is available. The second one is quite frequently used as well, and it is based on existing networks previously trained for some other problem. Then, it is only necessary to input some new data for the application required, but that has to be somehow related to the one used with the pretrained network. The advantage of this approach is that much less data is needed to make the model achieve a high accuracy.

Deep Learning applications are the same as seen in Machine Learning section, even it is applied in other fields because of the high accuracy explained above. One example is in the industry of Automated Driving, where high accuracy is really required for human safety. At the same time, it is used also in industries like Aerospace and Defense, Medical Research, Industrial Automation, Electronics and Computer Vision. Coming next, it is explained how it can be applied to the current problem.

4.3. Neural Networks architectures applicable to the current problem

There are a lot of possibilities for Neural Network architectures, and the user can decide the one that better suits the purpose, but the two most popular ones (a part of Fully Connected networks) are Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). For the current project, the second one is the relevant, as it allows to achieve high accuracy in Object Detection problems with images [30]. This is, it learns features from input data using 2D Convolutional Layers, which is very appropriate to process images, as can be represented as 2D data [31].

A CNN is composed by three different types of layers: Convolutional Layer, Pooling Layer and Rectified Linear Unit (RELU) Layer [32]. Convolutional Layers and Pooling Layers are alternated among all the Neural Network. In this way, features are extracted in different levels of abstraction. With more layers, more complex can be the features learned. At the same time, the size of the data is reduced. Between these layers, there are also activation layers also called RELU. These three layers take care of the feature learning, as can be seen in Figure 4.4. Afterwards, there is the classification done by one or several fully connected layers. The last fully connected layer outputs a vector which dimension is equal to the number of classes wanted to predict, and the corresponding probability of the object to be each class.

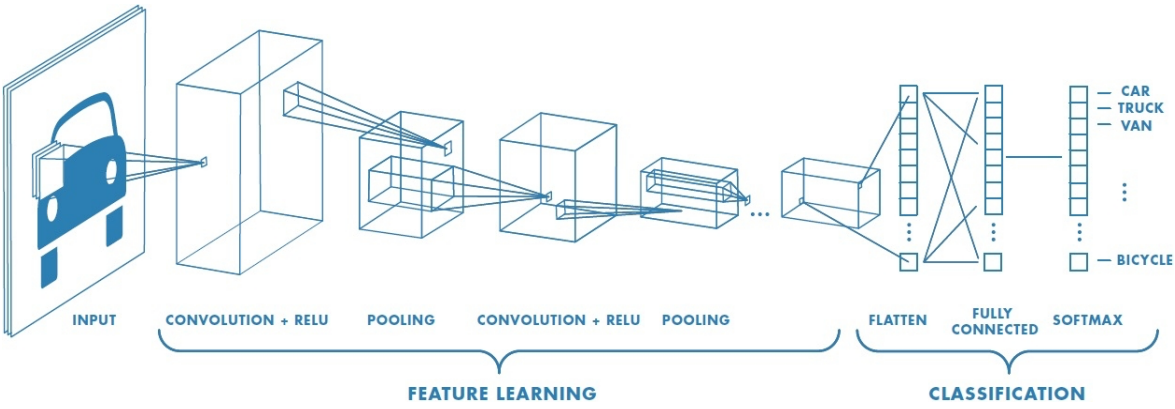

Figure 4.4. Example of a Neural Network composed by many convolutional layers. Source: [32].

The output of this type of Neural Network is the class of the corresponding object of the input image. So, to get a multiple object detection there are some options [33]. The first one consists in dividing the input image in small regions, treat these regions as individual images and then rejoin them with the detected objects and their classes. This can give some errors due to, for example, the objects being in different scales. To solve this, new algorithms that deal with it have been developed, such as RCNN, Fast RCNN and Faster RCNN. They are more complex but they are able to work faster.

In the same way, there are other approaches to the Object Detection problem that are being used nowadays. They use also Convolutional Layers but the architecture of the Neural Network is different. YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector) [34] are some of the 'state-of-the-art real-time object detection systems' [35].

5. Problem to solve

It has been briefly explained at the beginning, and here some more details about the problem to solve will be given. So, the problem is to detect, classify and localize cases stacked on a pallet. To do it, an algorithm should be developed, that could be able to process different kind of images. The input images are taken by a RGB-D camera located as shown in Figure 5.1.

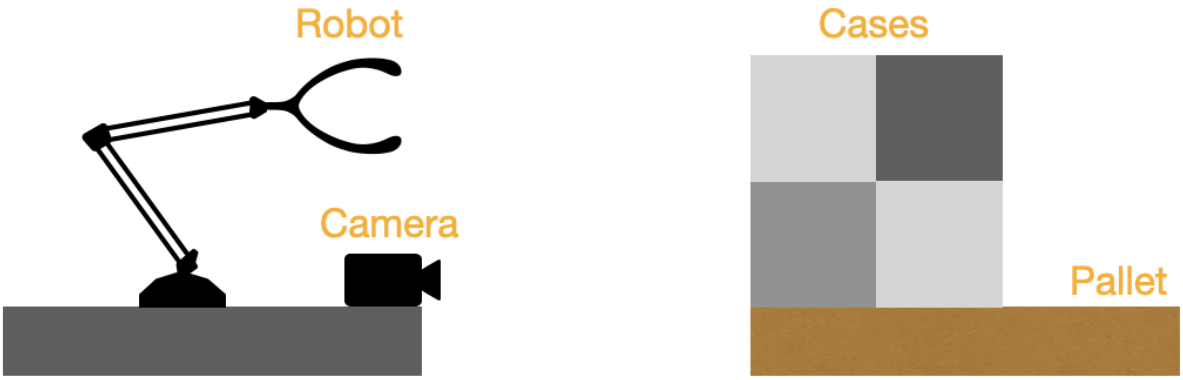

Figure 5.1. Representation of the setup for the problem at hand. Source: own.

To deal with this problem a database has been built beforehand. This database is constituted by twelve different cases with one image of each of their faces, except for the bottom face. Also, for those cases that have some faces equals, there is only one image of those faces. Hence, there are in total 56 pictures. These images are like the Figure 5.2, where the face comprises the whole image. It is important to emphasize that all the faces in the dataset have some signs indicating the product contained inside the case.

Figure 5.2. Database image example. Source: Prisma Lab.

Thus, at the end, for one input image of the whole pallet, the algorithm developed should be able to detect, classify and localize the cases. What this means is that the output of the Object Detection has to be the input image with bounding boxes around the found cases and some reference about which is their class. At the same time, as it is required to embed this algorithm with another one in charge of moving the robot to pick the cases and move them, all this information should be returned in some computer variables. In this way, these variables are easy to be used by the other algorithm and, thus, get a complete automated depalletization.

6. Proposed approach

6.1. Initial approaches

Several approaches have been tried to solve the problem at hand. Here each of them will be described, along with the reasons why some of them have been discarded and finally the algorithm implemented.

First of all, the Python language is used because it has become the standard for this kind of problems in the last years and because it is open source too. It has some libraries that are very useful for Computer Vision problems, such as OpenCV [36], so it is another point in favour of this language. Despite this, some tests have been done using Matlab due to its good interface to start working on the project.

The first approach was based on trying to make use of a Matlab project already developed which uses Principal Component Analysis, a method to reduce the data dimensionality, aimed to recognize faces. The impediment was that it uses a detector specialized for faces, so it was impossible to apply it to the current problem, as it could be expected. Then, some simple Deep Learning algorithms using Matlab were run to explore the differences between Object Classification and Object Detection.

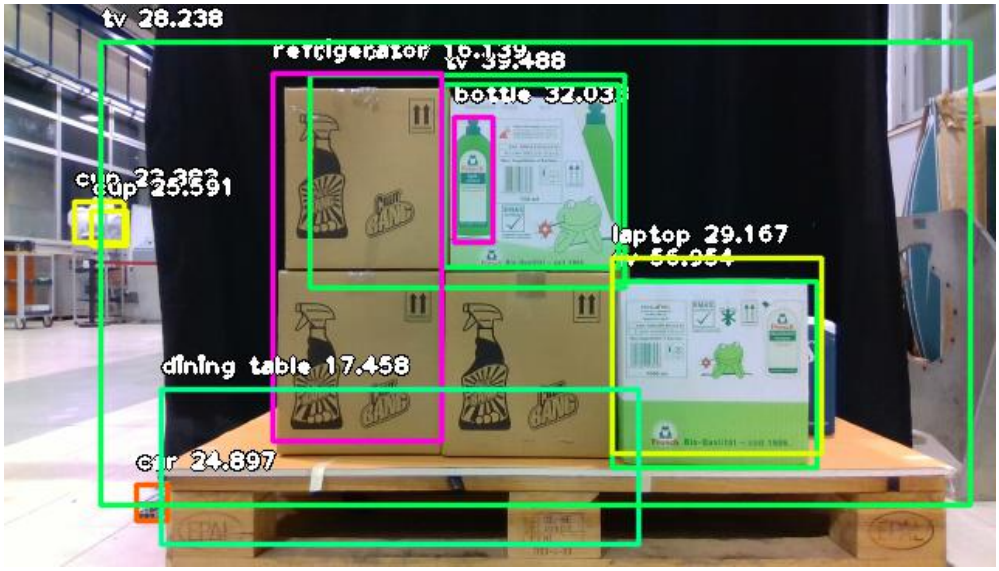

Figure 6.1. Output image obtained applying a pre-trained Neural Network for different types of objects. Source: own.

Once some basic knowledge about the problem to be treated was acquired, a step forward was done by taking a look at Python state-of-art algorithms. The first algorithm tested (Appendix A) was based on taking advantage of a pre-trained Neural Network [37], a RetinaNet. One example of the result obtained can be seen in Figure 6.1. From this, it is understandable that using a pre-trained Neural Network on different types of objects would

not be suitable for the current problem.

Following with the idea to implement a Neural Network, because of his power and his increasing applications last years, Data Augmentation was tried to improve the results. This is, either taking advantage of a pre-trained Neural Network or starting from scratch with a new one, it was necessary to obtain a higher number of photos of the cases through Data Augmentation algorithms, because it was impossible to take photos due to the quarantine period, as commented above. To do so, the most used Deep Learning framework was used: Tensorflow with Keras API (Application Programming Interface) [38]. This is said to be the 'core open source library to develop and train Machine Learning models' [39]. With that, the Data Augmentation was achieved by rotating the images, changing the brightness and contrast, and so.

Once several images were obtained, the dataset could be passed through a Neural Network for Object Detection. This was made using the Computer Vision library ImageAI [40] and the YOLOv3 architecture. Indeed, transfer learning was applied using a pre-trained YOLOv3 model, due to the reduced number of images available even with the Data Augmentation.

Following the steps described in [41], it is only possible to achieve a poor Object Detection model (Appendix B). The main problem here is that the type of images in the dataset is different from the input images. In other words, to apply this algorithm correctly the dataset images should be like the input images, where the cases are stacked on a pallet. Thus, by labelling the cases manually beforehand, and using this as the training dataset, the Neural Network could learn properly to get the desired result. Anyway, an output image obtained with the method just described is shown in Figure 6.2, where it hardly detects some cases, like an '*ariel*' case, but the bounding boxes are very imprecise.

Figure 6.2. Output image obtained applying a Neural Network built by transfer learning. Source: own.

At this point, with the experience learned from the previous algorithm, it was attempted to elaborate another more simpler algorithm taking only pieces of the input images, framed into cases. Then, it was only needed to first pass the input image to another algorithm able to slice it or to directly detect the cases, and iterate over them. Thus, that would allow to obtain at the same time the classified cases (Neural Network) and their localization (slicing algorithm).

To reach this goal, the Keras API was used again, starting from [42]. A Convolutional Neural Network was designed consisting on some simple Convolutional Layers (Appendix C). To begin with the code, as with many Deep Learning algorithms, the dataset is divided in three different sets: train, validation and test. This is done to check the accuracy of the algorithm and avoid overfitting. Then, Data Augmentation is implemented in the same way as done before. Lastly, the model is built concatenating some layers and compiled. Finally, the training is done passing the images, selecting the number of epochs and steps and indicating the number of classes.

Furthermore, the training results obtained were plotted to check its performance. Specifically, two graphics were plotted that represent the accuracy and the loss for the training and validation sets. When these values are stabilized, it means that the algorithm has converged and that more training epochs are not needed. As can be seen in Figure 6.3, the accuracy reached was around 95% for both sets, which can be considered a good result.

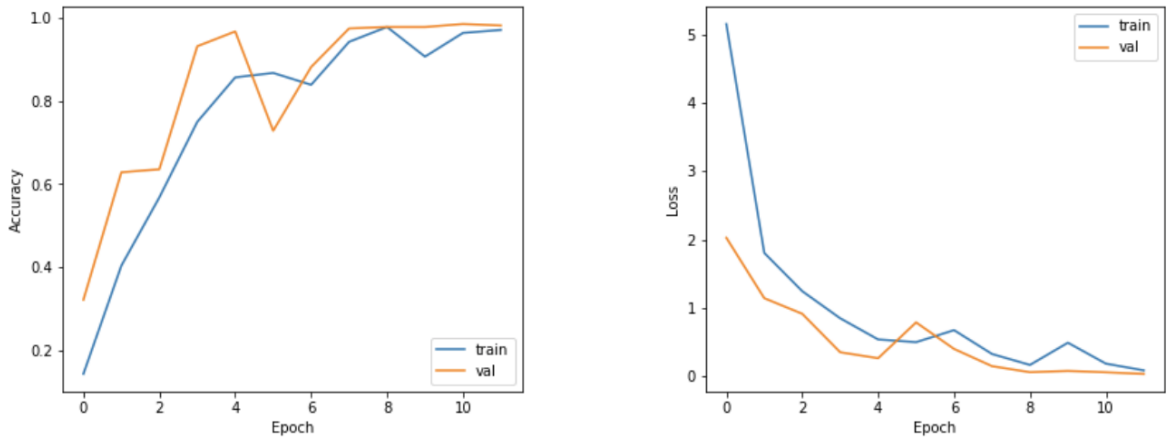

Figure 6.3. Graphics that show the accuracy and the loss through the epochs of the algorithm for the training and validation datasets. Source: own.

Once trained the algorithm, the last step was to evaluate the model. To do so, the images of the test set were used. These images were expected to be similar to those that would be obtained with the preceding slicing or detecting algorithm applied to the input image. What was obtained from the CNN for each test image was the weights corresponding to the predictions of each of the classes. The biggest value was selected and showed as the prediction of the case. For example, with the following indexes of the classes:

{*0: 'always',*
1: 'ariel',
2: 'bang',
3: 'frisch-black',
4: 'frisch-blue',
5: 'frosch-mandelmilch',
6: 'frosch-spulmittel',
7: 'pampers',
8: 'proper',
9: 'sidolin',
10: 'spee',
11: 'viss'}

Figure 6.4. Example image passed to the algorithm to get its prediction. Source: Prisma Lab.

If the image in Figure 6.4 was passed, the weights obtained were:

[[3.69623733e-12
1.20962314e-01
8.78833294e-01
5.80994808e-10
4.30229428e-11
1.83847576e-10
2.04426062e-04
3.62235075e-10
9.25389594e-14
5.89575589e-15
1.41097394e-11
2.70308654e-11]]

The third value was the biggest one, so the corresponding class was '*bang*', and this was showed as the prediction. So, the prediction was correctly done for this case and also for most of the cases. Hence, it only remained to develop the slicing or detecting algorithm. Some research was done, and the better way to proceed was to use some point feature extraction as done by the Prisma Lab, although it was not really what was desired. In fact, a shape detector, which was more similar to what was required, was tried to implement, but it did not work. As a result of this, and as can be appreciated coming next, this Deep Learning algorithm is not applied for the final approach, but may be used in future works.

SIFT was the algorithm used by the Prisma Lab, but as it is no longer open source, other similar methods were studied. Speeded-Up Robust Features (SURF) and Oriented FAST

and Rotated BRIEF (ORB) are similar methods that try to improve SIFT. The first one is faster than SIFT, but it is not available in open source either. On the other hand, ORB has the power of its precedents algorithms and it is a free option available in OpenCV library [43]. In fact, there are other algorithms working on the problem of feature detection and description, such as FAST and BRIEF. Because of the combination of these methods, ORB gets its name. This is, ORB uses the FAST keypoint detector together with Harris corner measure and also pyramid to work with multiscale-features. Then, some improvements are made to get a better performance, specifically to solve the rotation invariance. In addition, ORB takes advantage of BRIEF descriptors, modifying them a bit using the orientation of keypoints. This gives a little transformed algorithm called rBRIEF.

At this point, a code was found that mixes feature matching and homography to find known objects in images [44]. This consists in embedding a feature matching algorithm [45] with the function *findHomography* from calib3d module [46]. This feature matching is based on a Brute-Force Matcher, that 'takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation', and the nearest match is returned. Once the feature points are obtained, the *findHomography* function allows to get the perspective transformation of the desired object and then find exactly where it is. In other words, having an image with a known object, it is possible to know if this object is in another image and also its location. This ability was the required for the current project, so this approach was applied with some modifications and improvements.

Thus, the final approach is composed of three principal algorithms. The one commented lastly is implemented with a Sliding Window to get the multiple cases of the input image precisely. Therefore, as all cases have different dimensions, finding a Sliding Window that works for all the cases is hard. Then, it is implemented the third algorithm that works with image pyramids [47]. This allows to find objects at different scales in images, by changing the size of the image, as seen in Figure 6.5. Hence, applying the code of [48] which embed Sliding Window with image pyramids, without changing the size of the window it is possible to get the cases of different dimensions.

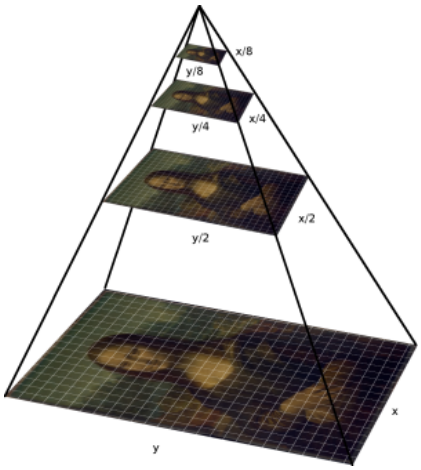

Figure 6.5. Example of an image pyramid. Source: [47].

6.2. Final approach

Finally, here the final approach implemented will be commented step by step. To begin with the code in Appendix D, the necessary libraries are imported, and also auxiliary functions that take care of the Sliding Window and the image pyramids. These two functions are exactly the same as provided in [48]. Then, the input image is cropped to avoid some wrong detections, which can be done precisely because the position of the camera and the pallet are always the same. Once all the initial configurations are made, the list of cases images in the dataset is iterated, and for each of them, all the faces of the cases are iterated too.

In this way, the algorithms presented above are applied for each face of each case. Then, with the corresponding input image read, all the resized images obtained by the pyramid function are iterated, as well as all the windows obtained by the Sliding Window. Most of the parameters of these two iteration functions are obtained by experimentation, to get that ones that fit better to the problem. The window size is set to a value similar to the dimensions of most of the cases. Each window is resized to the dimension of the database image with a linear interpolation, to be able to detect the point features and the matches correctly.

To continue, the ORB detector is initialized by setting the maximum number of features to be obtained at 2000, because the default value was not enough in some cases. Then, keypoints and descriptors for the input image and for each window are found with the ORB detector. Afterwards, using the matcher described above, matches are obtained through the descriptors. All the matches are iterated to get the best ones. With them, if the total of matches is higher than the minimum number of matches, set to 110 by trial and error, then the *findHomography* function is applied. With it and the *perspectiveTransform* function, it is possible to get the bounding box for each case. At this point, both images compared, this is, the one corresponding from the dataset and the window of the input image, are shown with the matches and the bounding box, as can be seen in Figure 6.6.

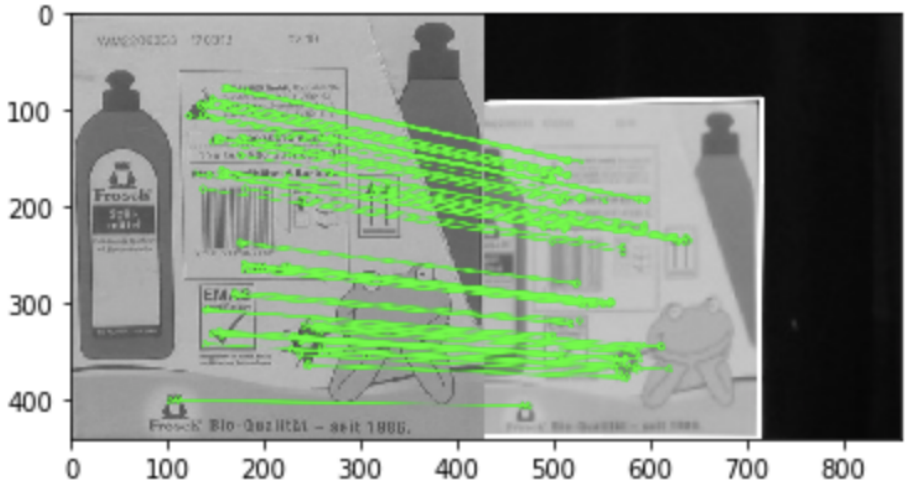

Figure 6.6. On the left, the dataset image. On the right, the resized window considered at the moment with the bounding box obtained. In green, the matches found. Source: own.

As shown in Figure 6.6, despite the fact that the case found is not completely in the window, the bounding box is obtained precisely because the relative distances are maintained. Consequently, the position of the bounding box is resized and saved, and also rescaled if needed.

In addition, as the input image is iterated several times, the same case in the same position may be detected multiple times too. For this reason, a list is created to keep track of those positions already detected. Then, for each “new” case detected, it is checked if it is already on the list. If it is not, the position is appended to the list and also is stored in a dictionary whose keys are the coordinates of the top left point, and the values are the number of matches and the corresponding face of the corresponding case. If the case is already in the list, which means that the new position has the same coordinates approximately (with a small margin of error of 20 pixels for both directions), then the number of matches are compared. The margin is because the bounding box may vary a bit for each case, but it is assumed that with more matches, the bounding box obtained is better. As a result of this, if the new detection has more matches, the corresponding values in the list and also in the dictionary are replaced for the new ones.

Finally, a filter function is applied to the list and the dictionary of detections, to delete some strange detections that may occur, such as a bounding box defined by a single point. Then, the list and the dictionary are returned, and the image with the bounding boxes of the cases detected is printed, such as in Figure 6.7. As can be seen, the algorithm developed does successfully the Object Detection required at the beginning of the project. To check if this algorithm performs better than the one built by the Prisma Lab is difficult because of the reduced number of images available. However, the results obtained by applying the algorithm to these images are attached in Appendix E. It is important emphasize that when the algorithm is applied multiple times to the same image the result may vary a bit due to the point features extracted, that influence the bounding boxes obtained.

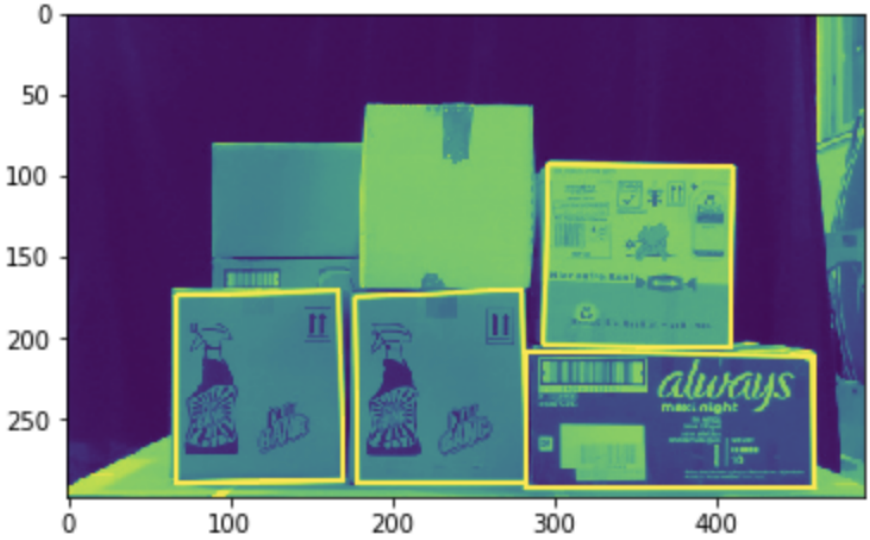

Figure 6.7. Image returned by the algorithm with the bounding boxes around those cases detected. Source: own.

7. Challenges encountered

One of the main problems, already mentioned, that came across during this project was the lack of a wide dataset, and the impossibility to get more data due to the quarantine period. This has influenced the final approach developed. For example, if there had been more photos like those used as input of the algorithm, a Deep Learning algorithm could have been developed to take advantage of the commented benefits.

Another problem encountered was that with the algorithm developed, some cases that are not in the front part are detected as well. This can be seen in Figure 7.1, where the top right case of the product called '*ariel*' is detected, not the one on its left, of the '*bang*' product. As explained at the beginning, these cases cannot be picked by the robot. In fact, that might not be considered as a problem, but just a thing to take into account if the algorithm is implemented.

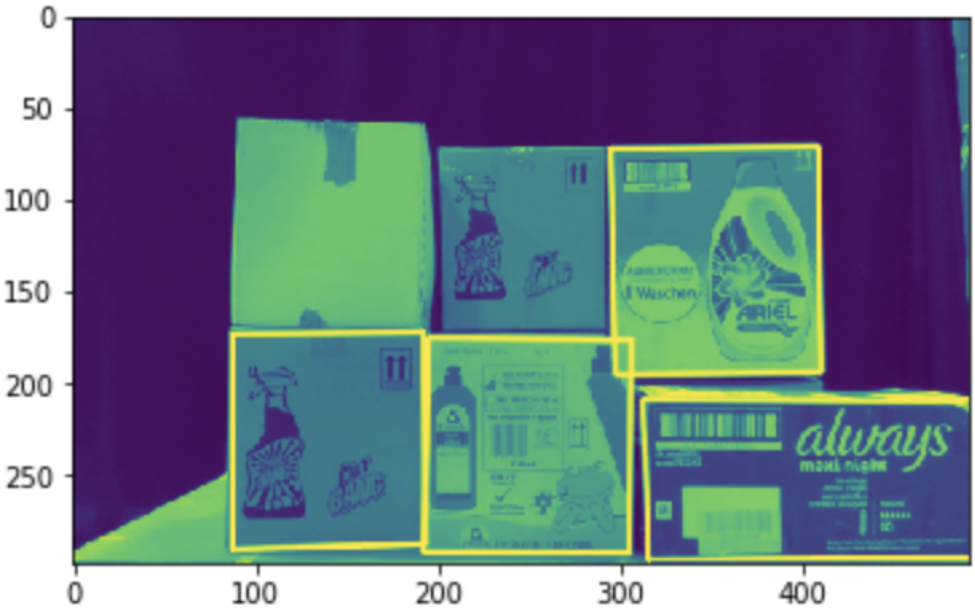

Figure 7.1. Output image where the algorithm has detected a case located at the back of the pallet. Source: own.

The different kinds of bounding boxes obtained depending on the matches found is another issue faced during the development of the final approach. It has been solved in a simple but quite effective way, using the maximum number of matches, but future work may be done to improve it more.

8. Future work

The first improvement that could be done with the same dataset would be detecting those cases that are not inside the dataset. This is, to use some algorithm able to do the same as the one achieved, and also detect those unknown cases. In fact, some of them can be inferred from those cases recognized correctly. As the position of the camera and the pallet are always the same, if a case is located much above from the bottom part of the pallet and no case is recognized underneath, there might be a case not existing in the dataset.

Another point that may be changed is about recognizing cases without drawings in the external faces. As commented in the outline of the project, these cases are ignored. In fact, all of the images in the database have some drawings. Hence, the algorithm developed might not work for this kind of cases, but it has been impossible to test it due to the lack of images.

Moreover, improving the velocity of the algorithm would be another important issue. Indeed, as it has been done, iterating over all the windows obtained by the Sliding Window method and over all the cases of the dataset is not the most optimal way to reach the goal desired, although it works pretty well.

As it has been stated several times, having a larger dataset would have made easier to work on the algorithm, to check the accuracy of the one obtained and also to make it applicable to wider types of cases, situations and even objects. So, like in many Machine Learning problems, acquiring enough data is a crucial step and, in this project, a great improvement could be accomplished with it.

Another improvement that could be developed would be getting the best bounding box among all those obtained from the same case. This is, as commented when the final approach is explained, as the same case can be detected multiple times, the bounding box selected is the one with more matches, because it is assumed to be the best one. But this is not always the case, as it can be seen in Figure 8.1. In this example, the '*always*' case has been detected many times, but the one with more matches has not given the best bounding box. So, implementing a more complex logic, maybe a weighted average of all the bounding boxes, would be more useful.

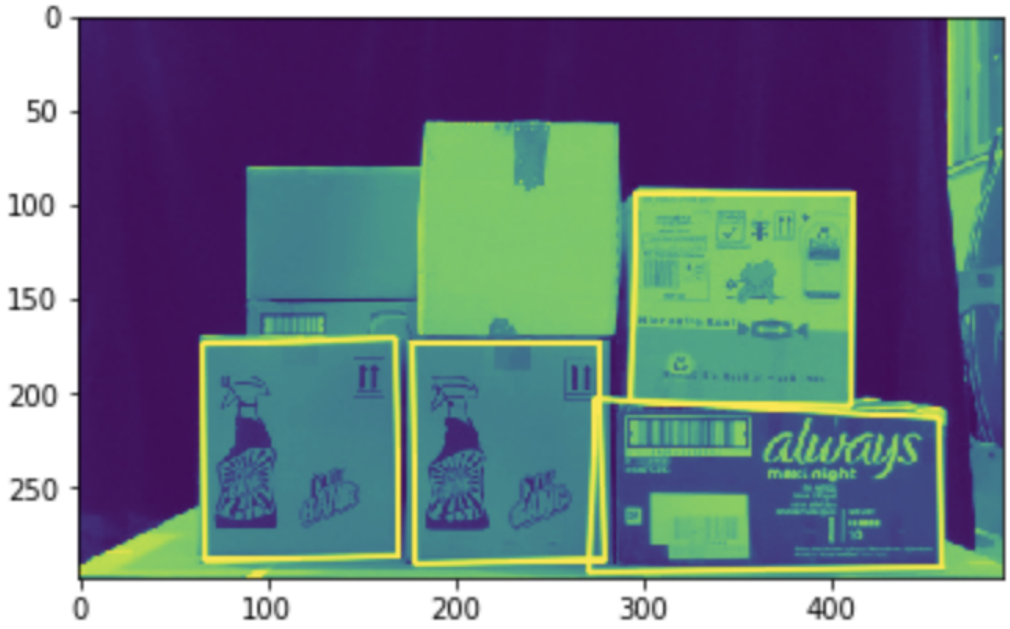

Figure 8.1. Output image where the algorithm has picked a bounding box that is not the best of all those obtained. Source: own.

In addition, the Deep Learning algorithm developed to classify the cases could be embedded with the Sliding Window method to see how it would work. Yet, it would probably be more complex and slow to get the exact position of the cases, and it might still require some other technique.

Taking advantage of the unique positions of the camera and the pallet, a different size of Sliding Window for each case may be implemented. Doing so, a bigger number of matches would probably be obtained, so the bounding boxes could be better.

Finally, more drastic would be to change almost all the algorithm to implement a Histogram of Oriented Gradients (HOG) and see if it works better than what it has been achieved. Also, a SVM technique may be suitable to go with it, as commented above.

9. Planning and programming

The current project has been developed in different stages. The initial one consisted on learning about the problem, acquiring more knowledge about Computer Vision and doing a lot of research. Once basic ideas were understood, it was the time to work with the dataset and start to develop some of the algorithms described above. Furthermore, at the same time, a Machine Learning online course provided by Stanford University was attended [20], to acquire more knowledge about possible techniques to apply to the current project. After trying multiple techniques and understanding exactly the problem at hand, the work on the final approach began. As soon as an algorithm capable of solving the task was obtained, even with some more improvements, writing this report started. The Gantt diagram in Figure 9.1 shows more precisely the steps followed and the corresponding time-lines.

2020																					
Month	**February**			**March**				**April**					**May**				**June**				
Week	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Activity	10/02-16/02	17/02-23/02	24/02-01/03	02/03-08/03	09/03-15/03	16/03-22/03	23/03-29/03	30/03-05/04	06/04-12/04	13/04-19/04	20/04-26/04	27/04-03/05	04/05-10/05	11/05-17/05	18/05-24/05	25/05-31/05	01/06-07/06	08/06-14/06	15/06-21/06	22/06-28/06	29/06-05/07
1. Initial research																					
2. Initial approaches																					
3. Final approach																					
4. Improvement of the algorithm																					
5. Writing report																					
6. Machine Learning course																					

Figure 9.1. Gantt diagram showing the steps and time-lines followed. Source: own.

10. Cost estimation

The current project has few costs associated as it only consists on developing an algorithm using a computer. Hence, it has to take into account the amortization of the computer, the Internet connection and the engineer salary. The cost of electricity consumed is neglected. The detailed costs are shown in the Table 10.1.

	Quantity	Cost/unit	Total
Computer amortization	64 cycles	1,5 €/cycle	96 €
Internet connection	5 months	30 €/month	150 €
Engineer salary	600 h	40 €/h	24.000 €
			24.246 €

Table 10.1. Detailed costs of the project. Source: own.

As the images used were acquired before this project was developed, the camera amortization has not been considered. The computer amortization is calculated as follows. The computer used is a MacBook Pro (13-inch, 2017, Two Thunderbolt 3 ports), which has a maximum battery cycle count of 1000 [49], which at the moment is 280. It costs around 1500 €, and it was bought at September of 2018. Hence, it has 22 months of lifetime. So, working on the current project can be estimated to have spent 64 cycles, having an estimated cost of 96 €. Regarding the Internet connection, the providing company is TIM, which offers a pack that costs around 30 € per month.

11. Environmental impact

There is not a relevant direct environmental impact of this project, as it has been about developing an algorithm. Indirectly, it can be taken into account the impact caused by the energy consumed. In the same way, if this project was implemented, it would make increase the automatization and the efficiency of some processes, which would probably result in a decrease of energy consumption.

Conclusions

Even if it has not been possible to compare the algorithm explained here with the one already developed by the team at Prisma Lab, it can be said that the goal of detecting, recognizing and locating cases of an input image from a database has been reached. Furthermore, in general, the bounding boxes obtained are very precisely, except some particular cases as it has been explained.

Moreover, it is relevant to emphasize the enormous importance of having a good database in a Computer Vision project. As it has been commented many times, the results obtained may vary a lot depending on the initial dataset. Therefore, this a point to take into account for future work.

Additionally, while investigating how to solve the problem considered at the beginning of this project, knowledge about important fields of Artificial Intelligence, such as Computer Vision, Machine Learning and Deep Learning, has been acquired. This is indeed something really useful because of the great future of all these fields and, in general, of Artificial Intelligence, which is expected to make a huge change to our lives.

Finally, in the case of Computer Vision, it has been seen that there is never a single solution. In fact, there are always many approaches to follow that might work well, and the key is to know which is better to implement depending on the problem at hand and the dataset available.

Acknowledgments

I want to express my gratitude to my co-supervisor Riccardo Caccavale from Prisma Lab, who has been very kind during the difficult period of sanitary emergency due to the Covid-19 pandemic, helping me to go on with this project and guiding me to reach the desired goal.

Bibliography

[1] LAURENCE GOASDUFF. *Top Trends on the Gartner Hype Cycle for Artificial Intelligence, 2019.*
[https://www.gartner.com/smarterwithgartner/top-trends-on-the-gartner-hype-cycle-for-artificial-intelligence-2019/, July 1, 2020].

[2] D. KATSOULAS and D. I. KOSMOPOULOS. *An efficient depalletizing system based on 2d range imagery, in Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164),* vol. 1. IEEE, 2001, pp. 305–312.

[3] D. KATSOULAS and B. LOTHAR. *Efficient 3D Vertex Detection in Range Images Acquired with a Laser Sensor, in Proceedings of the 23rd DAGM-Symposium on Pattern Recognition,* vol. 2191, 2001, pp. 116–123.

[4] Y. RAAJ, S. NAIR, and A. KNOLL. *Precise Measurement of Cargo Boxes for Gantry Robot Palletization in Large Scale Workspaces Using Low-Cost RGB-D Sensors, in 2016 Asian Conference on Computer Vision,* 2016, pp. 472–486.

[5] D. HOLZ, A. TOPALIDOU-KYNIAZOPOULOU, J. STÜCKLER, and S. BEHNKE. *Real-time object detection, localization and verification for fast robotic depalletizing, in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2015, pp. 1459–1466.

[6] F. WEICHERT, S. SKIBINSKI, and J. E. A. STENZEL. *Automated detection of euro pallet loads by interpreting pmd camera depth images, Logistics Research,* vol. 6, pp. 99–118. 2013.

[7] I. MOHAMED, A. CAPITANELLI, and F. E. A. MASTROGIOVANNI. *Detection, localisation and tracking of pallets using machine learning techniques and 2d range data, Neural Computing and Applications.* 2019.

[8] H. NAKAMOTO, H. ETO, T. SONOURA, J. TANAKA, and A. OGAWA. *Highspeed and compact depalletizing robot capable of handling packages stacked complicatedly, in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2016, pp. 344–349.

[9] JASON BROWNLEE. *A Gentle Introduction to Computer Vision.*
[https://machinelearningmastery.com/what-is-computer-vision/, July 3, 2020].

[10] ILIJA MIHAJLOVIC. *Everything You Ever Wanted To Know About Computer Vision*. [https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e, July 3, 2020].

[11] VERDICT. *History of computer vision: Timeline.* [https://www.verdict.co.uk/computer-vision-timeline/, July 3, 2020].

[12] ZBIGNIEW ZDZIARSKI. *The Early History of Computer Vision.* [https://zbigatron.com/the-early-history-of-computer-vision/, July 3, 2020].

[13] JIA-BIN HUANG. *Introduction to Computer Vision.* [https://filebox.ece.vt.edu/~jbhuang/teaching/ece5554-4554/fa16/lectures/Lecture_01_Introduction.pdf, July 3, 2020].

[14] ARTHUR OUAKNINE. *Review of Deep Learning Algorithms for Object Detection.* [https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852, July 3, 2020].

[15] SAMSUNG. *Is it available to use pictures to unlock device when it is locked by Face Recognition?.* [https://www.samsung.com/ph/support/mobile-devices/is-it-available-to-use-pictures-to-unlock__device-when-it-is-locked-by-face-recognition/, July 3, 2020].

[16] BEN DICKSON**.** *What Is Computer Vision?.* [https://www.pcmag.com/news/what-is-computer-vision, July 3, 2020].

[17] SAS. *Computer Vision, What it is and why it matters.* [https://www.sas.com/en_us/insights/analytics/computer-vision.html, July 3, 2020].

[18] EXPERT SYSTEM. *What is Machine Learning? A definition.* [https://expertsystem.com/machine-learning-definition/, July 4, 2020].

[19] ANDRÉS GONZÁLEZ**.** *¿Qué es Machine Learning?.* [https://cleverdata.io/que-es-machine-learning-big-data/, July 4, 2020].

[20] ANDREW NG. *Machine Learning.* [https://www.coursera.org/learn/machine-learning, July 4, 2020].

[21] BUILTIN. *Machine Learning. What Is Machine Learning? What Are Machine Learning Algorithms? What Are Popular Machine Learning Examples?.* [https://builtin.com/machine-learning, July 4, 2020].

[22] SAS. *Machine Learning, What it is and why it matters.*

[https://www.sas.com/en_us/insights/analytics/machine-learning.html#machine-learning-importance, July 4, 2020].

[23] MATHWORKS. *What Is Machine Learning? 3 things you need to know.* [https://www.mathworks.com/discovery/machine-learning.html, July 4, 2020].

[24] DEVIN SONI. *Supervised vs. Unsupervised Learning, Understanding the differences between the two main types of machine learning methods.* [https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d, July 3, 2020].

[25] MIRIAM LEESER and HAIQIAN YU. *Computer Vision.* [https://sites.google.com/site/hsi2013logan99/computer-vision/implementation, July 4, 2020].

[26] SEROKELL. *Artificial Intelligence vs. Machine Learning vs. Deep Learning: What's the Difference.* [https://medium.com/ai-in-plain-english/artificial-intelligence-vs-machine-learning-vs-deep-learning-whats-the-difference-dccce18efe7f, July 4, 2020]

[27] ANDREW NG. *What data scientists should know about deep learning.* [https://www.slideshare.net/ExtractConf, July 4, 2020].

[28] TONY YIU. *Understanding Neural Networks, We Explore How Neural Networks Function in Order to Build an Intuitive Understanding of Deep Learning.* [https://towardsdatascience.com/understanding-neural-networks-19020b758230, July 4, 2020].

[29] MATHWORKS. *What Is Deep Learning? 3 things you need to know.* [https://www.mathworks.com/discovery/deep-learning.html, July 4, 2020].

[30] MATHWORKS. *What Is Object Detection? 3 things you need to know.* [https://www.mathworks.com/discovery/object-detection.html, July 4, 2020].

[31] DISHASHREE GUPTA. *Architecture of Convolutional Neural Networks (CNNs) demystified.* [https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/, July 4, 2020].

[32] MATHWORKS. *Convolutional Neural Network, 3 things you need to know.* [https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html, July 4, 2020].

[33] PULKIT SHARMA. *A Step-by-Step Introduction to the Basic Object Detection Algorithms (Part 1).*
[https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/, July 4, 2020].

[34] EDDIE FORSON. *Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning.*
[https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab, July 4, 2020].

[35] YOLO. *YOLO: Real-Time Object Detection.*
[https://pjreddie.com/darknet/yolo/, July 4, 2020].

[36] OPENCV. *OpenCV modules.*
[https://docs.opencv.org/master/index.html, July 4, 2020].

[37] MOSES OLAFENWA. *Object Detection with 10 lines of code.*
[https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606, July 4, 2020].

[38] KERAS. *Keras, Simple. Flexible. Powerful.*
[https://keras.io, July 4, 2020].

[39] TENSORFLOW. *An end-to-end open source machine learning platform.*
[https://www.tensorflow.org, July 4, 2020].

[40] MOSES OLAFENWA & JOHN OLAFENWA. *ImageAI, State-of-the-art Recognition and Detection AI with few lines of code.*
[http://www.imageai.org, July 4, 2020].

[41] MOSES OLAFENWA**.** *Train Object Detection AI with 6 lines of code.*
[https://medium.com/deepquestai/train-object-detection-ai-with-6-lines-of-code-6d087063f6ff, July 4, 2020].

[42] KERAS. *Mnist CNN.*
[https://keras.io/examples/mnist_cnn/, June 19, 2020].

[43] OPENCV. *ORB (Oriented FAST and Rotated BRIEF).*
[https://docs.opencv.org/master/d1/d89/tutorial_py_orb.html, July 4, 2020].

[44] OPENCV. *Feature Matching + Homography to find Objects.*

[https://docs.opencv.org/master/d1/de0/tutorial_py_feature_homography.html, July 4, 2020].

[45] OPENCV. *Feature Matching.*
[https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html, July 4, 2020].

[46] OPENCV. *Camera Calibration and 3D Reconstruction.*
[https://docs.opencv.org/master/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780, July 4, 2020].

[47] ADRIAN ROSEBROCK**.** *Image Pyramids with Python and OpenCV.*
[https://www.pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/, July 4, 2020].

[48] ADRIAN ROSEBROCK**.** *Sliding Windows for Object Detection with Python and OpenCV.*
[https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/, July 4, 2020].

[49] APPLE. *Determine battery cycle count for Mac notebooks.*
[https://support.apple.com/en-us/HT201585, July 5, 2020].

Appendix

A. First algorithm applied (Python code)

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path ,
"resnet50_coco_best_v2.0.1.h5"))
detector.loadModel()
detections =
detector.detectObjectsFromImage(input_image=os.path.join(execution
_path , "input_image_7.jpg"),
output_image_path=os.path.join(execution_path ,
"imagenew.jpg"),minimum_percentage_probability=15)

for eachObject in detections:
    print(eachObject["name"] , " : " ,
eachObject["percentage_probability"] )
```

B. Object Detection algorithm by transfer learning (Python code)

```
!pip3 install tensorflow-gpu==1.13.1
!pip3 install imageai --upgrade
!wget https://github.com/OlafenwaMoses/ImageAI/releases/download/
essential-v4/pretrained-yolov3.h5

from imageai.Detection.Custom import DetectionModelTrainer
trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="drive/My Drive/
headsets2")
trainer.setTrainConfig(object_names_array=["frosch","ariel","alway
s"], batch_size=4, num_experiments=5,
train_from_pretrained_model="pretrained-yolov3.h5")
trainer.trainModel()

from imageai.Detection.Custom import DetectionModelTrainer
trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="drive/My Drive/
headsets2")
trainer.evaluateModel(model_path="drive/My Drive/headsets2/
models", json_path="drive/My Drive/headsets2/json/
detection_config.json", iou_threshold=0.5, object_threshold=0.3,
nms_threshold=0.5)

from imageai.Detection.Custom import CustomObjectDetection
detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("drive/My Drive/headsets2/models/
detection_model-ex-001--loss-0009.142.h5")
detector.setJsonPath("drive/My Drive/headsets2/json/
detection_config.json")
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image="drive/My
Drive/input_image_3.jpg",
output_image_path="caixes_detectades.jpg",minimum_percentage_proba
bility=60)
for detection in detections:
     print(detection["name"], " : ",
detection["percentage_probability"], " : ",
detection["box_points"])
```

C. Convolutional Neural Network algorithm (Python code)

```
############
# IMPORTS #
############

# BASIC
import os
import sys
import math
import time
import datetime

# DS
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
%matplotlib inline

# DL
import tensorflow as tf
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, Sequential, load_model
#from tensorflow.keras.applications.inception_v3 import
InceptionV3
#from tensorflow.keras.applications.resnet50 import ResNet50,
preprocess_input
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout,
Flatten, Dense
#from tensorflow.keras.layers import Input,
GlobalAveragePooling2D, GlobalMaxPooling2D, Flatten, Dense,
Dropout, LSTM, GRU, SpatialDropout1D, Embedding, TimeDistributed,
Bidirectional, concatenate
from tensorflow.keras.optimizers import RMSprop

PATH_TRAIN = "dataset/train"
PATH_TRAIN_AUGMENTED = "dataset/train_augmented"
PATH_VAL = "dataset/validation"
PATH_TEST = "dataset/test"
FN_MODEL_FORMAT = 'model_{}.h5'

IMG_SIZE = 100
BATCH_SIZE = 32
```

```
# more data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    PATH_TRAIN,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    save_to_dir=PATH_TRAIN_AUGMENTED
)
#validation_generator = test_datagen.flow_from_directory(
validation_generator = train_datagen.flow_from_directory(
    PATH_VAL,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
)

label_to_idx = train_generator.class_indices

NUM_CLASSES = len(label_to_idx)
EPOCHS = 12
STEPS = 10

INPUT_SHAPE = (IMG_SIZE, IMG_SIZE, 3)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=INPUT_SHAPE))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(NUM_CLASSES, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(learning_rate=0.001),
              metrics=['accuracy'])

history = model.fit_generator(
    train_generator,
    steps_per_epoch=STEPS,
    validation_data=validation_generator,
```

```
    validation_steps=STEPS,
    epochs=EPOCHS,
    verbose=1
)

def plot_history(history):
    """ Build convergence history plot. """
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(acc))
    width_per_plot = 7
    height_per_plot = 5
    array_width = 2
    array_height = 1
    fig = plt.figure(figsize=(width_per_plot*array_width,
height_per_plot*array_height))
    fig.subplots_adjust(hspace=0.5, wspace=0.4)
    ax = fig.add_subplot(array_height, array_width,1)
    plt.plot(epochs, acc, label="train")
    plt.plot(epochs, val_acc, label="val")
    ax.set(
        xlabel="Epoch",
        ylabel="Accuracy",
    )
    ax.legend(loc='lower right')
    ax = fig.add_subplot(array_height, array_width, 2)
    plt.plot(epochs, loss, label="train")
    plt.plot(epochs, val_loss, label="val")
    ax.set(
        xlabel="Epoch",
        ylabel="Loss",
    )
    ax.legend(loc='upper right')
    return fig

fig = plot_history(history)

FN_MODEL =
FN_MODEL_FORMAT.format(datetime.datetime.now().strftime("%Y-%m-
%d_%H%M"))
print("Saving model:", FN_MODEL)
model.save(FN_MODEL)

#FN_MODEL = 'model_2020-04-18_2025.h5'
print("Loading model:", FN_MODEL)
```

```
model = load_model(FN_MODEL)

def evaluate(model, fn):
    print("-"*100)
    print("FN:", fn)
    img = image.load_img(
        os.path.join(PATH_TEST, fn),
        target_size=(IMG_SIZE, IMG_SIZE)
    )
    img = image.img_to_array(img)
    img = img/255.
    plt.imshow(img)
    plt.show()
    y_pred = model.predict(np.array([img]))
    print("Weights:", y_pred)
    print("Prediction:", idx_to_label[np.argmax(y_pred)])
    print("-"*100)
    return

EXTS = ['jpg']
for fn in os.listdir(PATH_TEST):
    if '.' in fn and fn.split('.')[-1] in EXTS:
        evaluate(model, fn)
```

To develop this algorithm and the one in the next section a Virtual Environment has been built and the following packages are installed:

absl-py==0.9.0

appnope==0.1.0

astor==0.8.1

backcall==0.1.0

cffi==1.14.0

click==7.1.1

colorama==0.4.3

cycler==0.10.0

decorator==4.4.2

enum34==1.1.10

gast==0.2.2

google-pasta==0.2.0

grpcio==1.28.1

h5py==2.10.0

ipykernel==5.1.3

ipython==7.8.0

ipython-genutils==0.2.0

jedi==0.17.0

joblib==0.14.1

jupyter-client==5.3.4

jupyter-core==4.6.0

Keras==2.2.4

Keras-Applications==1.0.8

Keras-Preprocessing==1.1.0

kiwisolver==1.2.0
Markdown==3.2.1
matplotlib==3.1.0
numpy==1.17.3
opt-einsum==3.2.1
pandas==0.25.2
parso==0.7.0
pexpect==4.8.0
pickleshare==0.7.5
Pillow==6.2.1
plaidbench==0.7.0
plaidml==0.7.0
plaidml-keras==0.7.0
prompt-toolkit==2.0.10
protobuf==3.11.3
ptyprocess==0.6.0
pycparser==2.20
Pygments==2.6.1
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2019.3
PyYAML==5.3.1
pyzmq==19.0.0
scikit-learn==0.21.3
scipy==1.3.1
six==1.14.0
tensorboard==2.0.0
tensorflow==2.0.0
tensorflow-estimator==2.0.1
termcolor==1.1.0
tornado==6.0.4
tqdm==4.38.0
traitlets==4.3.3
wcwidth==0.1.9
Werkzeug==1.0.1
wrapt==1.12.1

D. Final approach algorithm (Python code)

Here the same Virtual Environment as above is used. The two code files must be in the same path. Also, the dataset images must be placed as follows:

- images
    - inputs
    - train
        - always
        - spee
        - proper
        - pampers
        - bang
        - ariel
        - frosch-mandelmilch
        - viss
        - frisch-blue
        - frisch-black
        - frosch-spulmittel
        - sidolin

Then, the input images are inside the inputs folder. At the same time, for each case folder, there are the faces images inside with the corresponding name.

D.1. helpers.py file

```
# import the necessary packages
import imutils
def pyramid(image, scale=1.5, minSize=(30, 30)):
    # yield the original image
    yield image
    # keep looping over the pyramid
    while True:
        # compute the new dimensions of the image and resize it
        w = int(image.shape[1] / scale)
        image = imutils.resize(image, width=w)
        # if the resized image does not meet the supplied minimum
        # size, then stop constructing the pyramid
```

```
        if image.shape[0] < minSize[1] or image.shape[1] <
minSize[0]:
            break
        # yield the next image in the pyramid
        yield image
def sliding_window(image, stepSize, windowSize):
    # slide a window across the image
    for y in range(0, image.shape[0], stepSize):
        for x in range(0, image.shape[1], stepSize):
            # yield the current window
            yield (x, y, image[y:y + windowSize[1], x:x +
windowSize[0]])
```

D.2. final.py file

```
# import the necessary packages and functions
import helpers
from helpers import pyramid
from helpers import sliding_window
import argparse
import time
import cv2
from matplotlib import pyplot as plt
import numpy as np
from os import listdir
from os.path import isfile, join
import sys; sys.argv=['']; del sys

# path of the images subfolders
path_images='/images/'

# input image path
image=path_images+"inputs/input_image_3.jpg"

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", help="Path to the
image",default=image)
args = vars(ap.parse_args())
# load the image and define the window width and height
image = cv2.imread(args["image"],0)

# crop the image
image=image[0:298,109:600]
```

```
#plt.imshow(image)
#plt.show()

# Sliding Window size
(winW, winH) = (150, 145)

# auxiliary function to get images from subfolders
def ls(path):
    return [arch for arch in listdir(path) if isfile(join(path,
arch))]

# auxiliary function to get subfolders from path
def ls_folders(path):
    return [arch for arch in listdir(path)]

# get a list of cases subfolders from the training folder of the
database
cases_list=ls_folders(path_images+'train/')
cases_list.sort()
cases_list=cases_list[1:]

# initialize list and dictionary returned as output
positions=[]
cases={}

# iterate all the cases from the list of cases
for case in cases_list:
    # get images of the case faces
    faces_list=ls(path_images+'train/'+case)
    # iterate all the faces
    for face in faces_list:
        # the corresponding face image is read
        img1 = cv2.imread(path_images+'train/'+case+'/'+face,0)

        counter=0
        scal=1.5
        # loop over the image pyramid
        for resized in pyramid(image, scale=scal):
            # loop over the sliding window for each layer of the
pyramid
            for (x, y, window) in sliding_window(resized,
stepSize=32, windowSize=(winW, winH)):
                # define the minimum number of matches
                MIN_MATCH_COUNT = 110

                # default values used to resize
                dsize=(img1.shape[1],img1.shape[0])
```

```
                src=window
                fx = dsize[0]/src.shape[1]
                fy = dsize[1]/src.shape[0]

                # resize of the window to get the matches
correctly
                window = cv2.resize(src, dsize,fx,fy,
interpolation = cv2.INTER_LINEAR)

                # initiate ORB detector
                orb = cv2.ORB_create(2000)

                # find the keypoints and descriptors with ORB
                kp1, des1 = orb.detectAndCompute(img1,None)
                kp2, des2 = orb.detectAndCompute(window,None)

                # set up the matcher
                FLANN_INDEX_LSH = 6
                index_params= dict(algorithm = FLANN_INDEX_LSH,
                                   table_number = 6,
                                   key_size = 12,
                                   multi_probe_level = 1)

                search_params = dict(checks = 50)
                flann = cv2.FlannBasedMatcher(index_params,
search_params)
                # condition to avoid some errors
          if(des1 is not None and len(des1)>2 and des2 is not None
and len(des2)>2):
                    # find the matches
                    matches = flann.knnMatch(des1,des2,k=2)

                # store all the good matches as per Lowe's ratio
test.
                good = []
                for match in matches:
                    if len(match)==2:
                        m=match[0]
                        n=match[1]
                        if m.distance < 0.7*n.distance:
                            good.append(m)

                # conditions to find the bounding box
                if len(good)>MIN_MATCH_COUNT and len(kp2)>0 and
len(good) < len(kp2):
                    # find the bounding box
```

```
                    src_pts = np.float32([ kp1[m.queryIdx].pt for
m in good ]).reshape(-1,1,2)
                    dst_pts = np.float32([ kp2[m.trainIdx].pt for
m in good ]).reshape(-1,1,2)
                    M, mask = cv2.findHomography(src_pts, dst_pts,
cv2.RANSAC,5.0)
                    matchesMask = mask.ravel().tolist()
                    h,w = img1.shape
                    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],
[w-1,0] ]).reshape(-1,1,2)
                    dst = cv2.perspectiveTransform(pts,M)
                    window = cv2.polylines(window,
[np.int32(dst)],True,255,3, cv2.LINE_AA)
                    print( "OK - {}/{}".format(len(good),
MIN_MATCH_COUNT) )

                    # if it is wanted, show the two images being
dealt at the moment, with the matches and the bounding box
                    draw_params = dict(matchColor = (0,255,0), #
draw matches in green color
                            singlePointColor = None,
                            matchesMask = matchesMask, # draw only
inliers
                            flags = 2)
                    img3 =
cv2.drawMatches(img1,kp1,window,kp2,good,None,**draw_params)
                    #plt.imshow(img3, 'gray')
                    #plt.show()

                    # undo the resize of the window for the
vertices of the bounding box
                    pyramid_scale=counter*scal
                    position=[[x+dst[0][0][0]/fx,y+dst[0][0][1]/
fy],
                              [x+dst[1][0][0]/fx,y+dst[1][0][1]/
fy],
                              [x+dst[2][0][0]/fx,y+dst[2][0][1]/
fy],
                              [x+dst[3][0][0]/fx,y+dst[3][0][1]/
fy]]
          a_position=np.array(position)
                    if counter > 0:
                        a_position=a_position*pyramid_scale
                    position=a_position.tolist()

                    # round and convert to integer the vertices of
the bounding box
```

```
                    for i in position:
                        i[0]=int(round(i[0]))
                        i[1]=int(round(i[1]))

                    # check if the case has been already detected
                    already=False
                    for pos in positions:
                        if pos[0][0] + 20 >= position[0][0] >=
pos[0][0] - 20 and pos[0][1] + 20 >= position[0][1] >= pos[0][1] -
20:
                            already=pos
                    if already != False:
                        # if it has been already detected, compare
the number of matches, and if the new one is higher the
corresponding values are replaced
                        if len(good)>cases[str(already[0][0])
+str(already[0][1])][0]:
                            cases.pop(str(already[0][0])
+str(already[0][1]))
                            cases[str(position[0][0])
+str(position[0][1])]=[len(good),case+'-'+face[:-4]]
                            positions.remove(already)
                            positions.append(position)
                    else:
                        # if it has not been detected yet, the
corresponding values are added
                        positions.append(position)
                        cases[str(position[0][0])+str(position[0]
[1])]=[len(good),case+'-'+face[:-4]]
                else:
          matchesMask = None
             counter+=1

# filter function to delete strange detections
def filt(l,d):
    for c in l:
        if abs(c[0][0]-c[1][0])>10 or abs(c[3][0]-c[2][0])>10 or
abs(c[0][1]-c[3][1])>10 or abs(c[1][1]-c[2][1])>10 or abs(c[0][1]-
c[1][1])<10 or abs(c[3][1]-c[2][1])<10 or abs(c[0][0]-c[3][0])<10
or abs(c[1][0]-c[2][0])<10:
            l.remove(c)
            d.pop(str(c[0][0])+str(c[0][1]))

filt(positions,cases)

# the list and the dictionary of the cases detected are shown
print(positions)
```

```
print(cases)

# image with the bounding boxes of the cases detected is shown
array_pos=np.array(positions)
for p in array_pos:
    image = cv2.polylines(image,[np.int32(p)],True,255,2,
cv2.LINE_AA)
plt.imshow(image)
plt.show()
```

E. Final approach results

Here are presented the outputs obtained passing the different input images available to the final algorithm. These outputs consist on the images with the corresponding bounding boxes of the cases detected and the list and the dictionary with information about them.

E.1. Input image 1

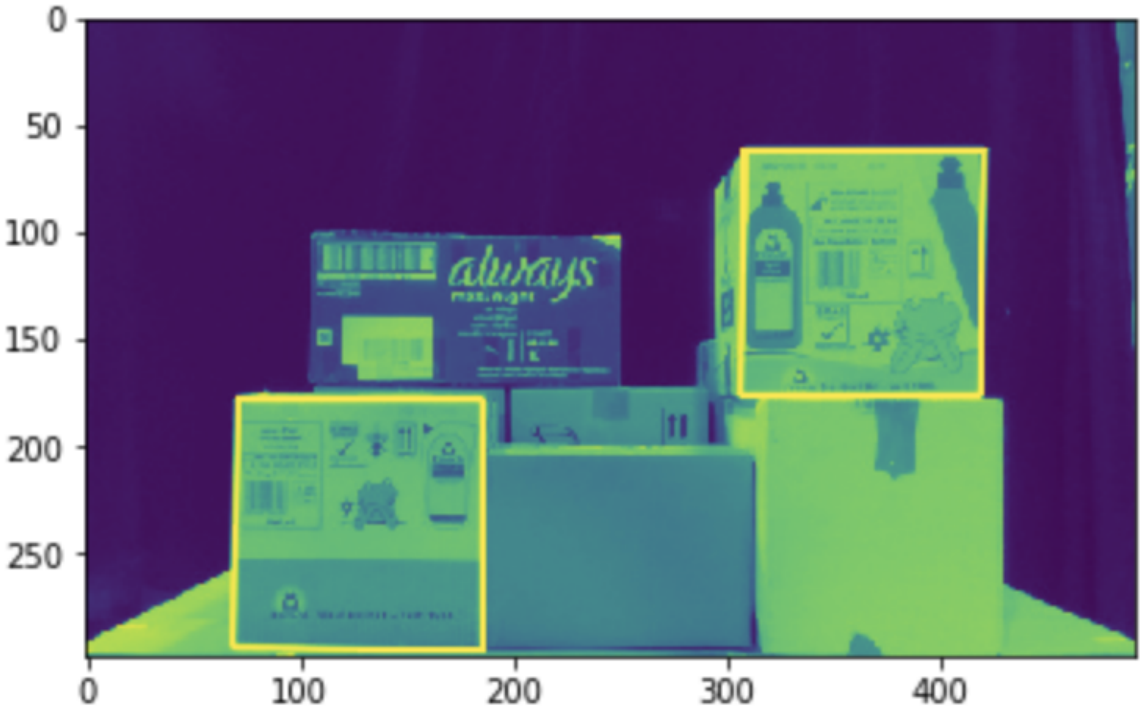

Figure E.1. Image returned by the algorithm with the bounding boxes around those cases detected for input image 1. Source: own.

List: *[[[71, 178], [69, 293], [185, 295], [185, 178]], [[308, 62], [306, 176], [418, 176], [420, 62]]]*

Dictionary: {'71178': [129, 'frosch-mandelmilch-rear'], '30862': [159, ‘frosch-spulmittel-rear']}

E.2. Input image 2

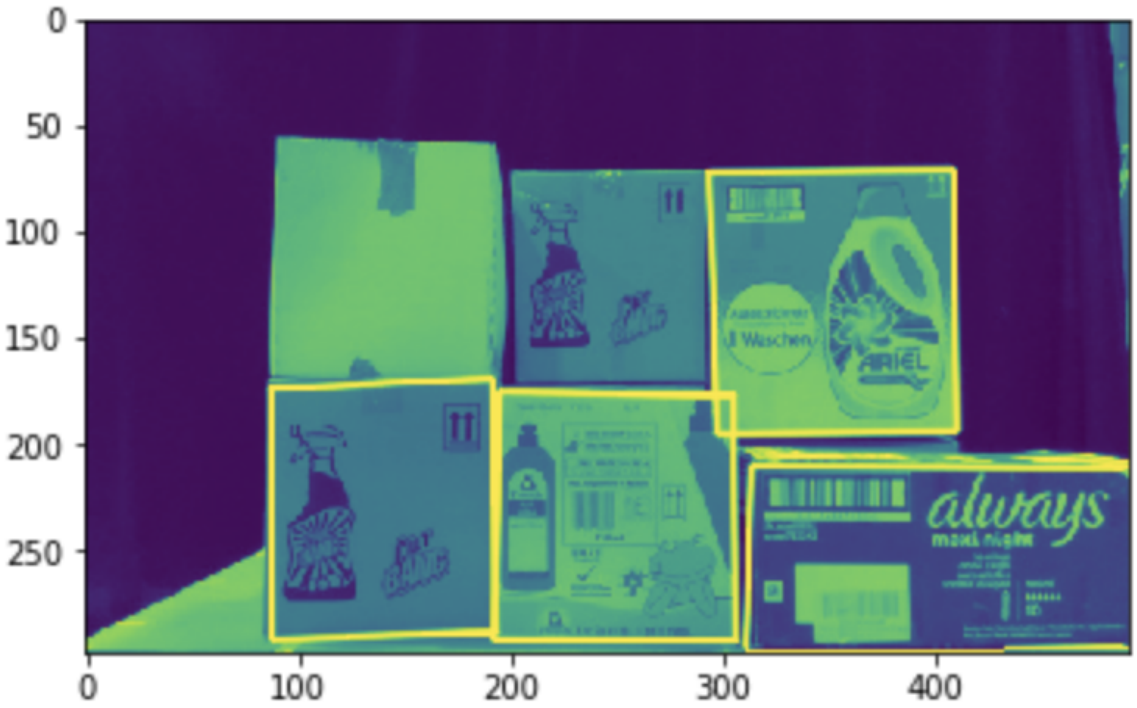

Figure E.2. Image returned by the algorithm with the bounding boxes around those cases detected for input image 2. Source: own.

List: *[[[313, 210], [310, 298], [494, 295], [494, 211]], [[293, 72], [297, 195], [410, 194], [408, 71]], [[87, 174], [88, 291], [193, 288], [191, 169]], [[195, 175], [192, 291], [306, 292], [305, 177]]]*

Dictionary: *{'313210': [182, 'always-forward'], '29372': [232, 'ariel-forward'], '87174': [129, 'bang-forward'], '195175': [177, 'frosch-spulmittel-rear']}*

E.3. Input image 3

Figure E.3. Image returned by the algorithm with the bounding boxes around those cases detected for input image 3. Source: own.

List: *[[[195, 211], [196, 291], [371, 297], [372, 212]], [[196, 54], [195, 203], [326, 207], [334, 59]], [[86, 175], [87, 291], [193, 289], [189, 170]], [[369, 177], [370, 295], [487, 293], [487, 178]]]*

Dictionary: *{'195211': [180, 'always-forward'], '19654': [312, 'ariel-forward'], '86175': [138, 'bang-forward'], '369177': [152, 'frosch-mandelmilch-rear']}*

E.4. Input image 4

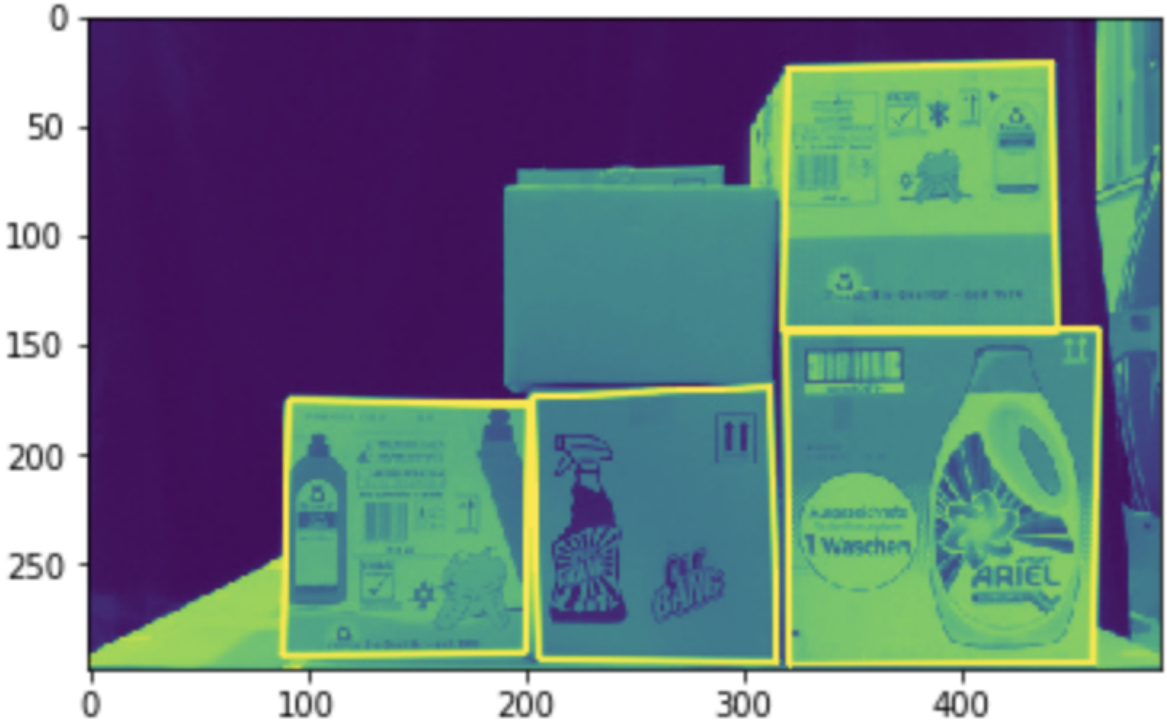

Figure E.4. Image returned by the algorithm with the bounding boxes around those cases detected for input image 4. Source: own.

List: *[[[319, 143], [321, 296], [460, 295], [462, 143]], [[203, 174], [206, 293], [315, 294], [312, 169]], [[320, 24], [318, 144], [443, 144], [441, 22]], [[92, 175], [89, 291], [200, 290], [202, 177]]]*

Dictionary: *{'319143': [313, 'ariel-forward'], '203174': [152, 'bang-forward'], '32024': [179, 'frosch-mandelmilch-rear'], '92175': [129, 'frosch-spulmittel-rear']}*

E.5. Input image 5

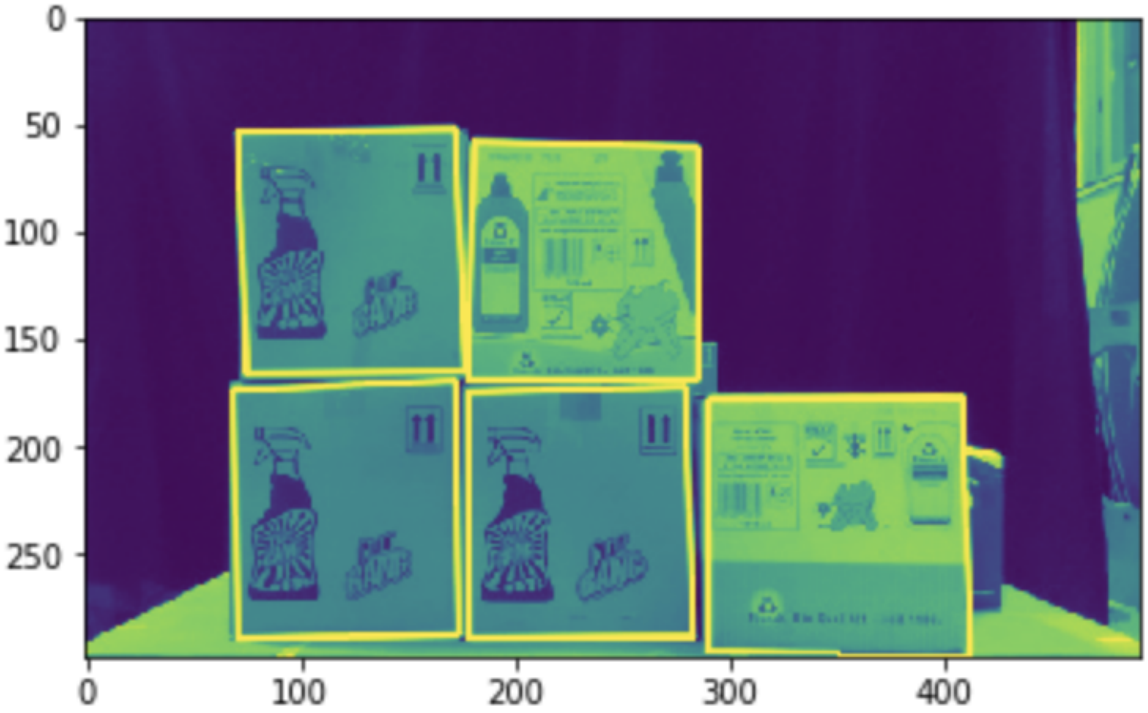

Figure E.5. Image returned by the algorithm with the bounding boxes around those cases detected for input image 5. Source: own.

List: *[[[71, 54], [75, 166], [177, 165], [172, 52]], [[69, 174], [71, 289], [174, 287], [172, 169]], [[178, 175], [179, 289], [283, 288], [279, 172]], [[290, 178], [290, 295], [412, 297], [408, 177]], [[181, 58], [178, 169], [285, 169], [285, 61]]]*

Dictionary: *{'7154': [140, 'bang-forward'], '69174': [157, 'bang-forward'], '178175': [158, 'bang-forward'], '290178': [133, 'frosch-mandelmilch-rear'], '18158': [163, 'frosch-spulmittel-rear']}*

E.6. Input image 6

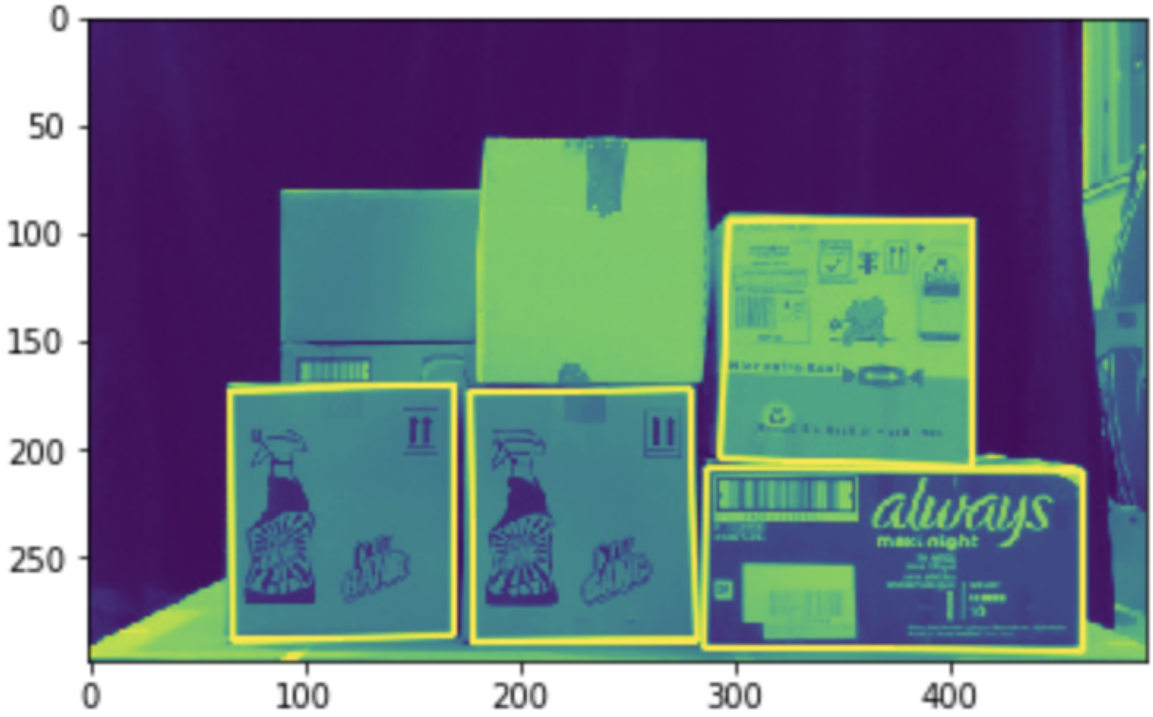

Figure E.6. Image returned by the algorithm with the bounding boxes around those cases detected for input image 6. Source: own.

List: *[[[287, 209], [286, 291], [461, 293], [461, 210]], [[66, 173], [67, 288], [169, 286], [169, 170]], [[177, 174], [179, 289], [282, 288], [279, 172]], [[297, 95], [293, 204], [410, 208], [410, 94]]]*

Dictionary: *{'287209': [146, 'always-forward'], '66173': [156, 'bang-forward'], '177174': [142, 'bang-forward'], '29795': [174, 'frosch-mandelmilch-forward']}*

E.7. Input image 7

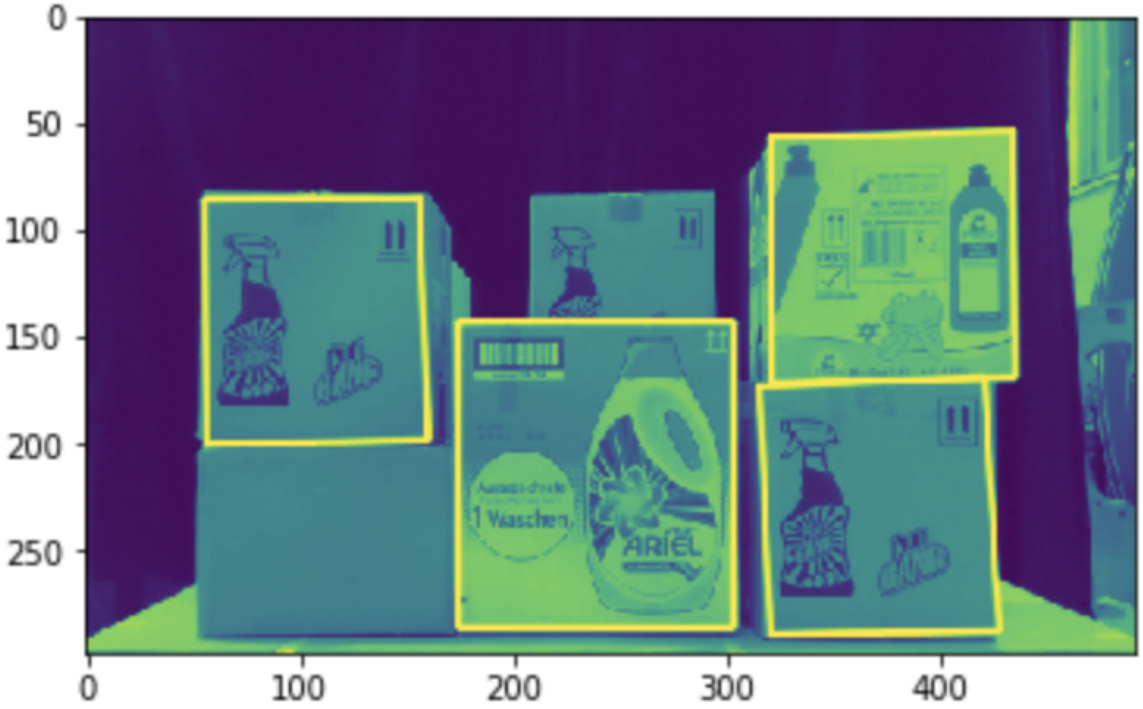

Figure E.7. Image returned by the algorithm with the bounding boxes around those cases detected for input image 7. Source: own.

List: *[[[174, 143], [174, 286], [303, 286], [303, 143]], [[55, 86], [56, 200], [160, 198], [156, 85]], [[315, 173], [320, 289], [427, 287], [420, 169]], [[320, 57], [321, 172], [435, 169], [434, 54]]]*

Dictionary: *{'174143': [292, 'ariel-forward'], '5586': [138, 'bang-forward'], '315173': [149, 'bang-forward'], '32057': [177, 'frosch-spulmittel-forward']}*