Computer Vision techniques and Deep Learning for performance improvement in an UR3 robot

Author: Maria Pilar Hernández Bas
Director: Philippe Juehel

***Abstract*—Taking advantage of the needs of industries to optimise their process, this project proposes an Artificial Intelligence solution that can help the performance of a industrial robotic arm, a UR3 robot. The implementation of the project is focused on Computer Vision , techniques for understanding the information of images. Afterwards, the features extracted from the environment pictures are fed into a Reinforcement Learning algorithm. The robot learns from the images and the coordinates to perform a Bin-Picking task. This solution implements of a proof of concept which yields satisfactory results and sets the bases and the architecture for continuance and improvement.**

Keywords: Artificial Intelligence, Computer Vision, Features, Feature extraction, CNN, Transfer Learning, ROS, UR3.

I. Introduction

Industrial robots are used nowadays to automatize processes and perform tasks in a more efficient manner. With that in mind, the goal is to maximize the utility of the machine while we decrease the amount of time that an action or a task will take. Techniques such as Machine Learning, Artificial Intelligence or Data processing must be applied to achieve a robot performance which could be capable of replacing the human efficiency.

Therefore, data is essential for all *Machine Learning* transformation process. Data is interpreted by machines to extract useful information. One of the most important tasks is "feature extraction" or "feature engineering". The objective is to collect all the knowledge and special qualities of the data son it can be converted into something useful for the problem that should be solved.

One of the most interesting data types are images. As inputs in a model, a picture can provide useful information of the environment. This information can be fed into our Machine Learning models so we can predict for example, if an action will be successful or what the next movements of our robot should be.

These images need be processed and prepared to be fed into models. Models expect the images standardized with some common characteristics. Afterwards, the models will use the images and inputs to extract the most relevant characteristics to be used in real environments.

II. Objectives

The goal for this project is to be able to teach the robot how to perform a task commonly known as *Bin-Picking*. Bin-picking is a type of Pick and Place task for robots.The robot will perform a set actions with the objective to collect objects from an environment, such as a box.

The robot should try to pick all the pieces in the less amount of time possible.To achieve the most efficient approach, *Artificial Intelligence* is added to improve the performance and obtain the most knowledge possible from the environment.

The robot will collect images taken from the environment. Based on the images of the environment the robot will make decisions to try to picking the objects displayed successfully and then, leave them in the desired place.

That is the main idea behind the project. Using the techniques and resources available to improve the performance of the *cobot*. This 'Bin-Picking' project, is composed by two main Machine Learning techniques: *Computer Vision* and *Reinforcement Learning* . This project will focus the attention into the *Computer Vision* techniques and how it affects the performance and how the *Image Recognition* and *Image Processing* play an important role in this problem. The outputs extracted from the *Computer Vision* algorithms will be used in a *Reinforcement Learning* algorithm and the whole performance of the robot will be evaluated.

III. System Description

A. Overview

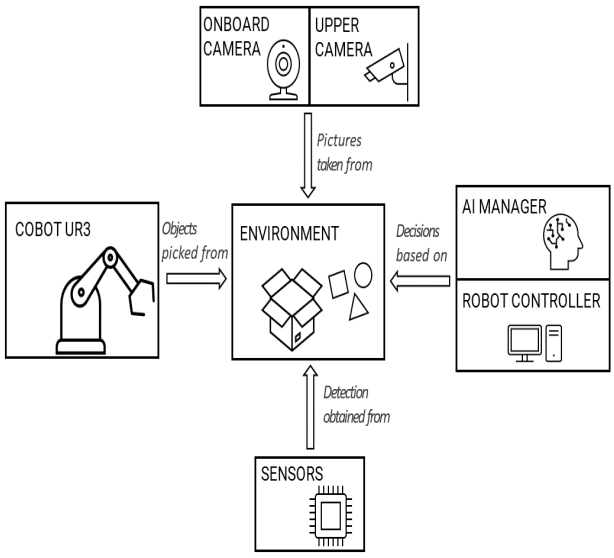

Fig. 1: Complete architecture

The system architecture was designed to adapt to the needs of the systems. Using *ROS*, several interconnected nodes, each

specialized in a task, were created to achieve the project objectives. All the components are specifically designed to have a better understanding of the environment.

B. Architecture

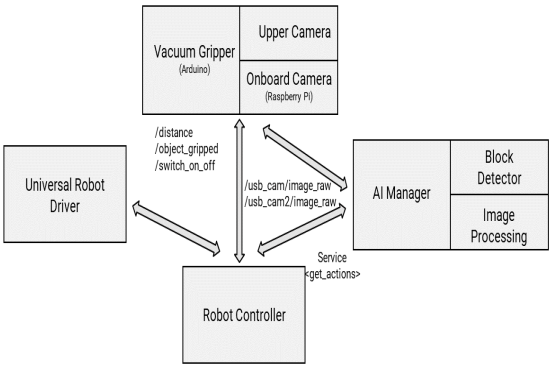

Fig. 2: ROS architecture

All the nodes are connected and they need to send and receive messages. By default, all the messages will be send by *ROS topics*, however, more critical information, like the actions for the robot, need to be confirmed so no information is lost. The **AI Manager** node uses the outputs of the **Image Processing** node and decides with the *Reinforcement Learning* algorithm, the actions that the robot should perform. This actions are send to the **Robot Controller** node, in charge of controlling and communicating with the robot.

There is also a important node, in charge of determining the coordinates where the robot must place itself at the end of a successful pick or when the robot goes out of limits. This node avoids the random movements and helps increasing the accuracy of the robot. The node is called Block Detector and uses traditional *Computer Vision* techniques to recognize the points of more probabilities for picking pieces.

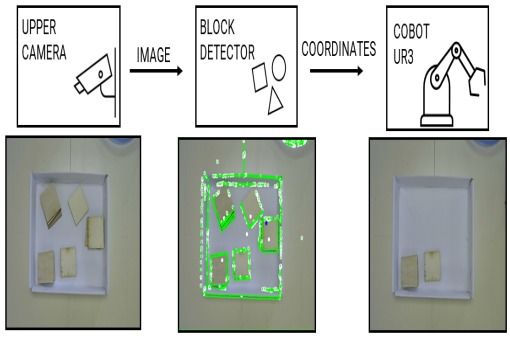

Fig. 3: Diagram of How Block Detector works

Finally, the main focus on the project is the **Image Processing node**.In this node, the images are taken from the onboard camera of the robot. This images are pre-processed and standardized, so the model expects always the same type of images with the same characteristics like size, colour and appearance. Once the images are ready to be used in the model, the images are divided into training and validation datasets. The data are trained with a neural network model, **CNN**. The model extracts the features of the image (what makes the image unique) and the classification of the image. The model predicts if the image will be a 'success' or a 'failure' so the next step can have the more information as possible to feed the *Reinforcement Learning* algorithm.

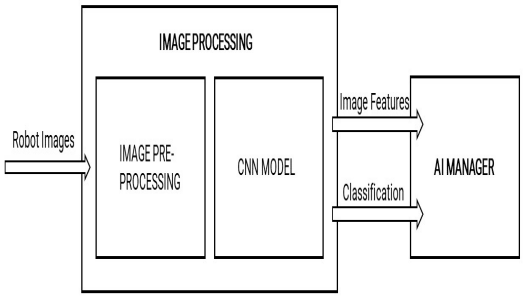

Fig. 4: Image Processing steps

These features, along with the coordinates and model prediction, will serve as inputs to the reinforcement learning model that will serve to determine the robot's movements.

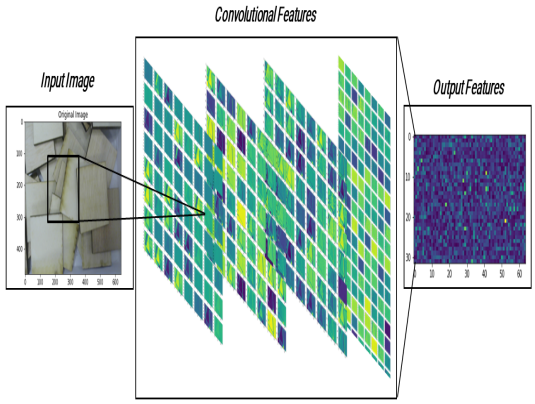

Fig. 5: Features extracted from the input image

IV. Computer Vision

Every *Computer Vision* system needs to requirements, a sensing device and a interpreting device. The sensing device will take the inputs (images) and they will be processed to extract information.

A. General Overview

Computer Vision applications can vary, but most of the steps for processing images follow a similar pipeline. In figure 6 it is represented the standard pipeline from the acquisition of images, to finally the classification or prediction, based on the extracted information from the pictures.

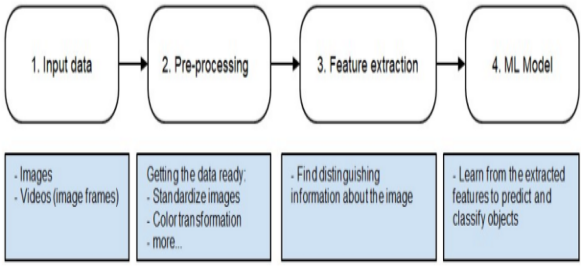

Fig. 6: Computer Vision Pipeline [1]

B. Camera Calibration

Before using a camera it is necessary to consider the parameters and characteristics of the camera used for the deployment.Image calibration can be crucial to get the correct results when dealing with 2d and 3d coordinates.

Camera calibration is the process of estimating the camera parameters. By using these coefficients it is possible to establish a relation between the 3D world, the real world, and the 2D projection(pixels).

The camera has two type of parameters: [2]

1) **Intrinsic** parameters. Ex: focal length, radial distortion...
2) **Extrinsic** parameters: Orientation of the camera (rotation and translation), with respect to the environment coordinates

In order to understand the camera calibration concepts, it is important to have some knowledge about image formation [3]:

There is a 3D point with coordinates x,y,z and it is necessary to find the coordinates in the image taken by the camera. The real environment is related with the camera environment by the parameters called: rotation and translation. The first step is to define a coordinate system with an origin and several known points with the coordinates x,y,z. This is called "World coordinate system". With this coordinates system we can access to any point in the real environment. Unfortunately, when a camera is added, the coordinate system is different and works in pixels. The idea is to put the camera in the place desired for the project and find the relation between the two coordinate systems. The camera is placed in an arbitrary position in the real word with coordinates cx,cy and cz. The camera is *translated* respect the world-coordinates.

In order to find the projection of 3D coordinates in an image, first, it is necessary to transform 3D to 2D using the extrinsic parameters. Next, using the *intrinsic* parameters, the point is projected onto the image plane. The camera algorithm has a set of images where the 2D points and the real-world points are known. The outputs will be a matrix of dimension 3x3, and a rotation and translation vector of each image.There are several calibration methods, and depending of the type of environment and the control over the process, it is possible to choose. In the case of our project, as there is a complete overview of the environment thanks to the upper camera in the structure, we can use the *Calibration Pattern*.The *Calibration Pattern* uses a checkerboard to perform the calibration.

a) : The calibration for the project is performed following these steps:

1) Define the environment with a checkerboard pattern. The total space of our environment must be represented by the chess board. The board is defined by the number of rows and columns. So, for example, the checkerboad of this project was (5x6)
2) Then, in order to obtained the most accurate parameters as possible, we should take several captures of the checkerboard from different angles. In figure 7 there are two examples of the chessboard from different angles. Approximately a good value of pictures for the calibration can be around 20 samples.

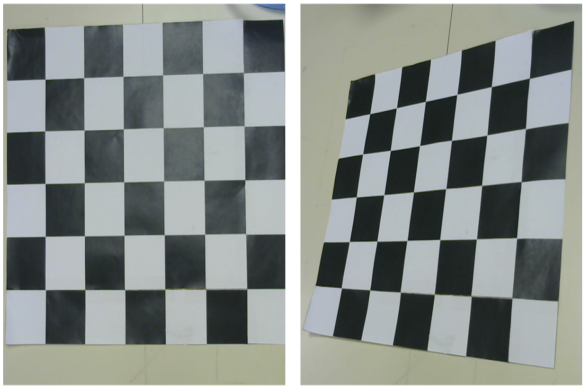

Fig. 7: Chess board for Camera Calibration

3) The camera produces a distortion of the images, and they are not a exact replica on the real world. It is important to remove the distortion to improve the calibration of the camera.
4) Using the chess board and some pre-processing techniques such as thresholds and image normalization, we detect the corners and the features of our environment.In figure 8 the chessboard is detected and prepared for extracting the camera parameters.
5) Then, it is time to calibrate the camera. After the calibration, the parameters of the camera will be saved, so they can be used in the next step, where it is necessary to extract coordinates from images. 8
6) Then, the coordinate system must be defined so it is possible to find the 2D coordinates and the relation between our coordinate systems. We will use the pa-

rameters obtained from the camera calibration in order to achieve it.

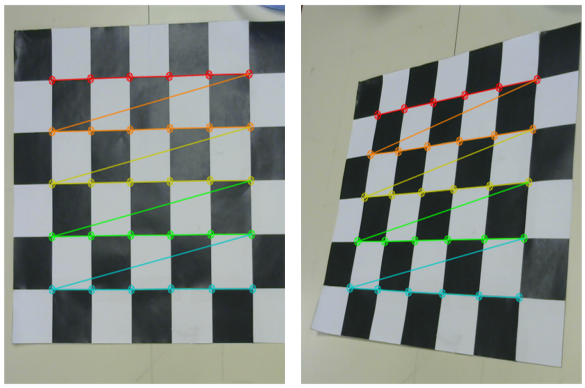

Fig. 8: Camera already calibrated

C. Data Gathering

Images are the most important input for the project. Images have the information about the environment. This information will be used by the algorithms to chose the next action for the robot to make. The development of the project started from zero and the environment and the architecture were designed exclusively for the project. The main problem when dealing with *Machine Learning* problems is the data. Before designing any model, it is important to answer a few questions about the data recollection.

1) Are there any initial images?
2) How the images will be collected?
3) What type of images are necessary?

The dataset of images must be created from scratch. The camera on board the robot is placed in the most optimal place possible to capture the view of the environment as similar as the vacuum griper "sees" the box. The images are collected when the robot tries to perform a pick action, based on the result the image will be classified as "success" or "fail" and it will be saved in the corresponding folder. Before training, it was necessary to have some images in advanced. The first images were taken by letting the robot made random movements all around the environment and trying different picks. The initial dataset was built in that way and when there were enough data to start modelling, the dataset keeps increasing while it is training by the *Reinforcement Learning* algorithm.

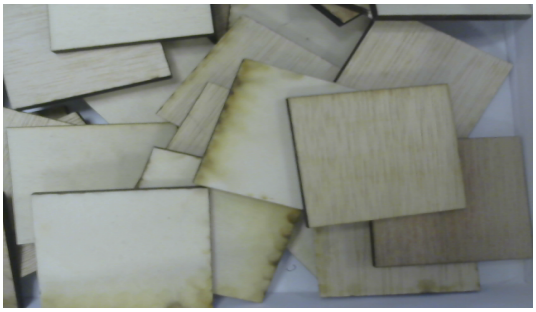

Fig. 9: Example of image taken from the onboard camera

In figure 9 there is an example of one of the pictures taken from the camera and classified as "success". The robot takes a picture of environment underneath the vacuum gripper. The centre of the image corresponds approximately with the position of the pump. Images must be in some way standardized. This process is called "pre-processing" and will be explained in the next section. They will be the inputs of the model and due to the conditions of the lab, the factor that can affect the most in the recollection is the light conditions, due to the amount of hours that the algorithms need to train.

D. Understanding the image

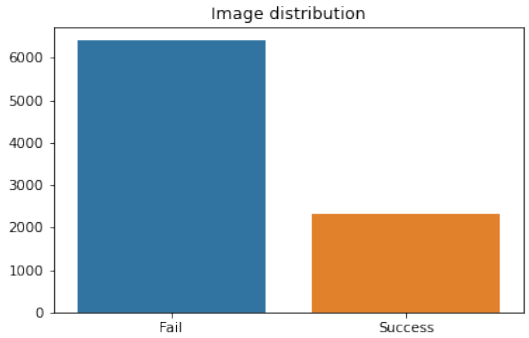

Fig. 10: Distribution of dataset

Before start using the dataset, it is important to explore the data in order to understand the complexity of the project and how the data must be treated to fit the models.

The first question must be about how the images are distributed in the dataset . In figure 10 there is a representation of the different classes and how the data is organized. Given the nature of the project itself it is normal to find more fail pictures than success images. This dataset has been built from the different trainings. This means that these pictures are taken not only from the model starts to learn, but also from the random phases of exploiting the environment whenever the robot is training.

This uneven distribution of data affect how the model performs. With a approximately ratio on 3:1 of failure over successful picks, the mode will tend to predict most of the times a fail pick, even when it is not true. This will be explain better in the metrics section V-C

a) How the unbalanced data can be handle?: The unbalanced data is a problem in the training data fed in the model. The model will get used to the majority class. Then, when the model is put on production, when other classes are presented, the model will not know how to generalized and the predictions will not be correct.

As the project is develop using Pytorch, there is a method called *WeightedRandomSampler*. The idea is to balanced the dataset using the weights of each class.

First, the images will divided in **train, validation and test** datasets. The proportion of images distributed in each dataset is **70%-15%-15%** using a RandomSplit(). That way we are forcing the images to shuffle, so there is a more uniformed distribution in the datasets. Only the train dataset needs to be balanced, so the other datasets will remain the same. From the train dataset, the number of samples per class are counted and then the weight of each picture is calculated, so the sampler will balanced each class using the associated weights. Then the number of samples will be calculated, more or less we want to have the most similar balance of data, so the number of samples will be $2 * n\_success\_samples$. As the dataset is large enough this will not reduce the training dataset in excess.

Here there is the implementation of the code in Python and Pytorch:

```
    WeightedRandomSampler(weight_samples,
    len(weight_samples),
    replacement=False,
    generator=torch.Generator().manual_seed
    (42))

```

Snippet 1: Weighted Random Sampler

All the images from the dataset are taken from the same camera in the same position. The images have the same characteristics:

Characteristic	Value
Width	640
Height	480
RGB	Yes
Format	PNG

TABLE I: Characteristics of the image

Besides the properties of the image, it is possible to see the colour distribution of the image in the following image. Colour has a very similar distribution. More or less the colours where there is more density of each colour correspond to the beige palette in RGB, which makes sense because they are the predominant colours in the image. We expect similar distributions but adding white colours and grey, due to the shades and the white box when there are less pieces or the box is empty. Besides, images are very similar to each other, and this can difficult the recognition of the images for the model.

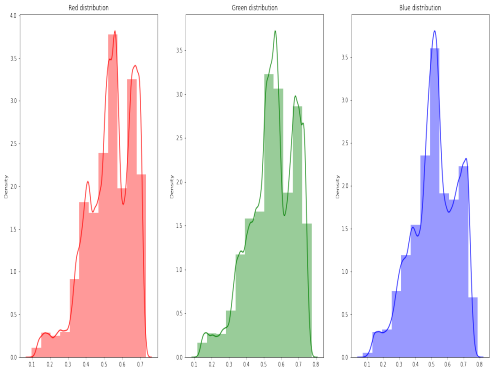

Fig. 11: Colour distribution of the image

E. Image Pre-processing

In this section, we will talk about the previous modifications that images must suffered before being prepared to be fed into the model. During this pre-processing phase, images can be altered to be standardized. The model need images with the same characteristics such as size or colour. In addition, when the dataset is not larger enough a good practice is to increase training dataset with data augmentation. In the next section we will get deeper into these techniques, choosing the pre-processing pipeline more adequate for our bin-picking task.

1) Image Pre-processing: Image pre-processing is that phase in the *Computer Vision* pipeline where the data is prepared to be fed into the model. The idea behind pre-processing the images, is to get the images the most standardized as possible. This would allow a better performance in the model. Additionally, some models can have concrete restrictions about the inputs, so the data needs to be prepared in an specific way. We are going to explore the most common techniques for data pre-processing and will explain the reason behind the use of some of them in the project.

1) **Center Crop**: This type of transformation just cut the image in the center. This technique can be useful when the main information is placed in the center of the image, and the rest is adding noise. In figure 12 there are some examples with different crops applied to one of the images in our dataset. The smaller the value of the crop size, the less information we are capturing from the image.

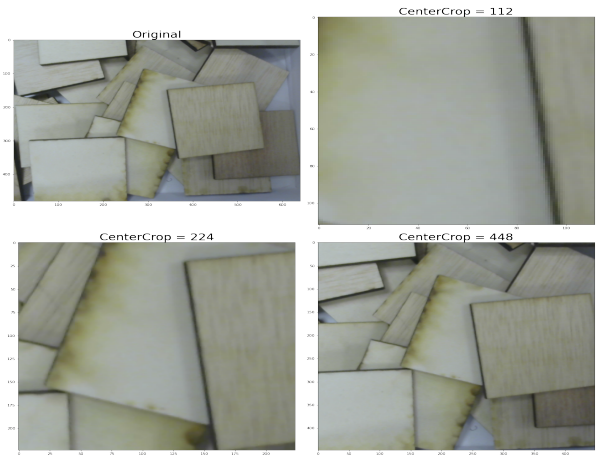

Fig. 12: Center Crop

2) **Resize**: This technique change the size of the image. The difference with the Center Crop is that while Crop just cuts the center of the image with the size specified, Resize adapts the whole original image into the shape desired. In figure 13 there are examples of different shapes of resize. This method can help reducing the computing time and the processing load when there are problems with the memory. The images maintain the same view, with a reduction in dimensionality.

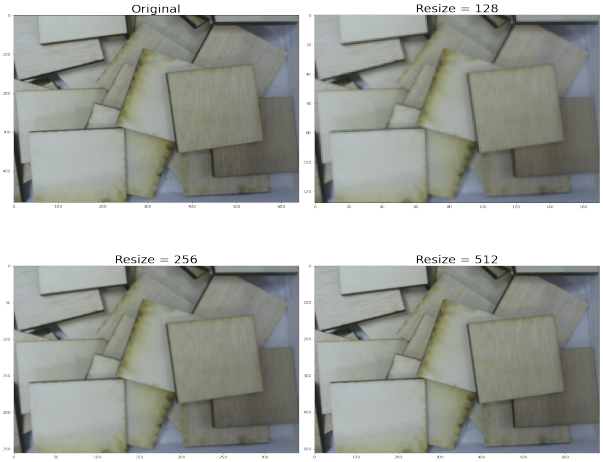

Fig. 13: Resized images

3) **Colour**: Depending on the images, or the tools, images may need a specific colour or colour format. For example, PIL or Matplotlib reads the images in RGB while OpenCV uses BGR. Conversions may be needed. Also, grey images contain less information and they are easier to process. Most of the pre-processing steps include grey transformations when the colour is not an important feature in the process.

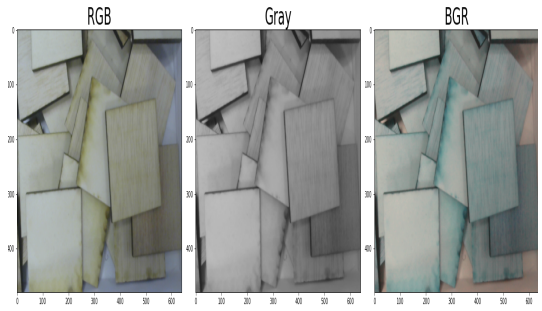

Fig. 14: Blur techniques

4) **Blur**: Technique to reduce the noise in an image with a low-pass filter kernel. Sometimes there are detailed of the images that they are not interesting and the model can focus on small useless details. Blurs allow to remove this details.

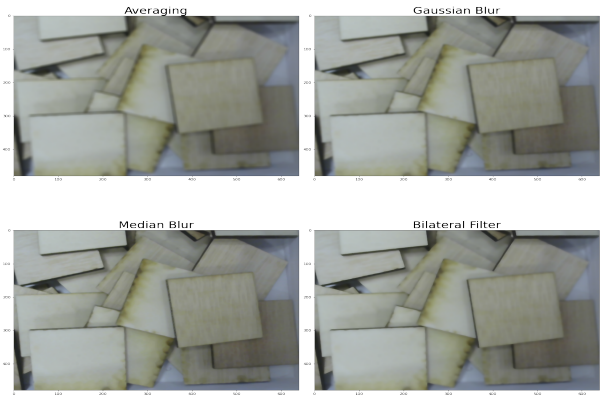

Fig. 15: Blur techniques

5) **Threshold**: Type of image segmentation. The pixels of the image change based on the threshold, so the image is easier to analyse. This technique converts pixels normally in greyscale.

- Regular Thresholds: This type of thresholds perform differently depending on the needs and the characteristics of the images. This techniques are very efficient and quick for processing.
- Adaptative Threshold: The next version of Thresholds. This techniques can adapt better to the characteristics of the image, with better performance but adding more time of processing. In figure 16 it is possible to see how both adaptive thresholds represent the image perfectly, but removing the colour. In this example, a previous blur is a good idea to remove the lines and dots of the surfaces.

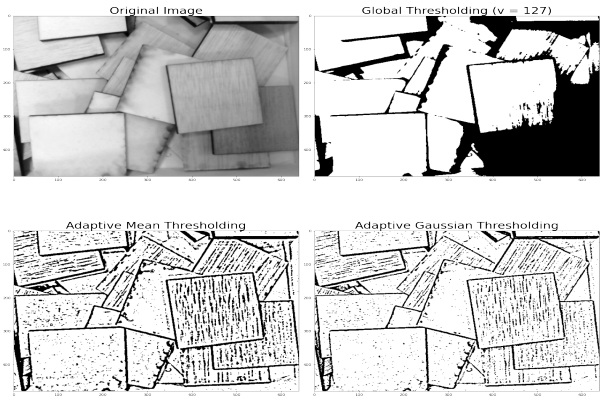

Fig. 16: Threshold techniques

6) **Brightness and contrast**: This techniques tries to reduce the effect of excess or default of light in the pictures. There are several parameters that can be controlled as brightness, the contrasts, saturation or hue. This techniques apply filter to the images, regulating and controlling light. In figure 17 there are some examples of the parameters. These techniques are very helpful in problems were the images are taken at different moments of the day for example.

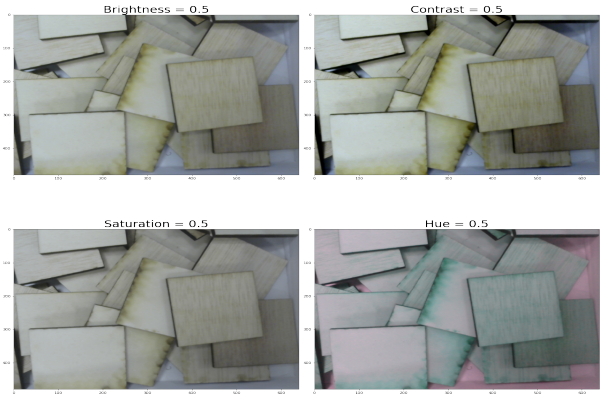

Fig. 17: Brightness and contrast

7) **Normalization**: This technique is mandatory in PyTorch for Transfer Learning algorithms, the models expect the inputs normalized the same way. The values for normalization are:

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

```

Snippet 2: Normalization for tensors

a) What types of image pre-processing will be used?: During this project, some techniques will be used, both for the *Deep Learning* CNN model and for the Block Detector.

- Block Detector: The images taken from the upper camera will be converted into Greyscale, and then for detecting the image better a Bilateral Blurred Filter and a Gaussian Adaptive Threshold.
- CNN Model: The pre-processing is more complex also due to the PyTorch requirements for Transfer Learning models.

1) Greyscale with 3 RGB input channels due to the needs for the Transfer Learning models.
2) Resize with size 256 to reduce computation time and memory with CUDA.
3) Gaussian Blur: to remove irrelevant information in the pictures.
4) ToTensor: In Pytorch is mandaory to convert the images into tensors before entering the model.
5) Normalize as expected in Pytorch.

2) Data Augmentation: Data Augmentation allows to have a larger dataset without taking more pictures from the camera. Similar as pre-processing there are some techniques that modify the image so some characteristics change and the image is a little bit different than before. This is very interesting because it gives the dataset some variety that will help the model with the generalization.

There are multiple techniques to perform data augmentation. In this section the focus will be on the *Pytorch Transform* [4] techniques, because they are the ones that can be used in the Pytorch model.

1) **Rotation**: Rotate the image certain degrees.
2) **Flip**: The image will be completely flipped vertically or horizontally. For data augmentation you give a probability.
3) **Erasing**: Its a random selection of certain regions of the image, where the pixels are removes the values of the pixels.
4) **Random Affine**: Transformation that conserves parallelism and lines, but not necessarily distances or angles

In addition, there are some methods explained in the pre-processing section that can introduce random modifications: Random Center Crop or Random Colour Jitter. In figure 18

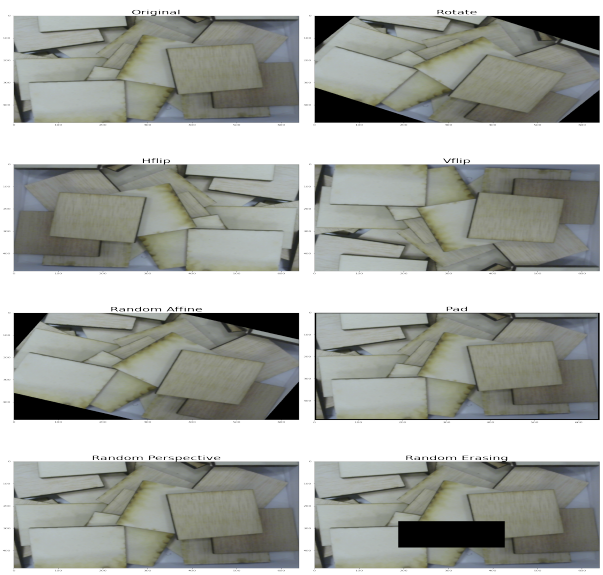

Fig. 18: Different data augmentation techniques

a) What types of data augmentation will be used?: Besides the same basic transformations needed in the pre-processing step, data augmentation can help the model generalize. Adding some type of transformations, the model will receive images with different colours, flips, or rotations. But, for example, the main problem with our model is that we are analysing images with a concrete disposition and this features will be used to take decisions based on coordinates. This is the reason while the data augmentation will be focused on small rotations of the images, and some changes in brightness

F. Feature Extraction

Feature extraction consists in extracting the useful characteristics of an image. Feature is a property that can be measured. These features from the input images, will help the model classify and understand the environment better. The goal of the feature extraction is to be able to identify objects inside the images. There are two ways of obtaining features from images. In the following subsections, the difference between a more traditional approach versus a Deep Learning approach for feature extraction will be described.

1) Manual Feature Extraction: Traditional Feature Extraction has been done in *Machine Learning* for a long time and in the occasions where *Deep Learning* is not recommendable, there are several techniques that allows to extract good features that then, can be fed into *Machine Learning* models, like a Support Vectors Machine algorithm.

In figure 19 there is a representation of an image after a K-Means clustering method. The algorithm tries to separate samples in a number of groups specified called clusters. The algorithm creates centroids, the mean representation of each group. In the figure 19 is possible to see the clustering method applied to one of the images. This technique obtains some features of the image. The different clusters are associated with the range of light and colour. The pieces in the left of the image are more bright and clear, and they are categorized in the same cluster. The darker parts of the box are represented by a very dark blue. This is an example of a more traditional feature extraction.

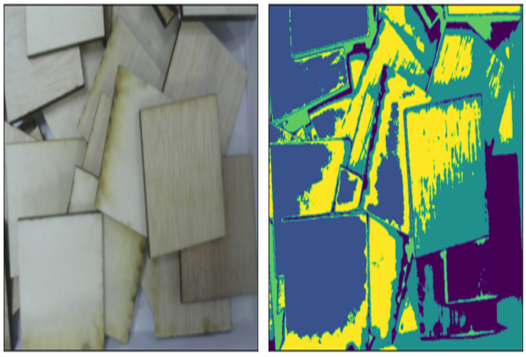

Fig. 19: Original Image vs. Cluster features

Thinking about the type of images in our problem, seems very plausible that the most important features in the images would be edges, contours and centers of shape. In figure 20 we have chosen a contour detector that can almost perfectly reproduce the contours and edges of our image. This techniques of contours will be taken into account when developing the Block Detector system for identifying pieces.

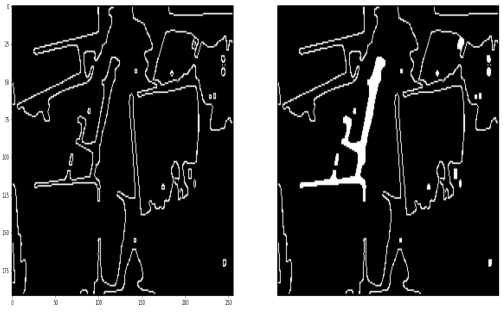

Fig. 20: Canny Edge Detector

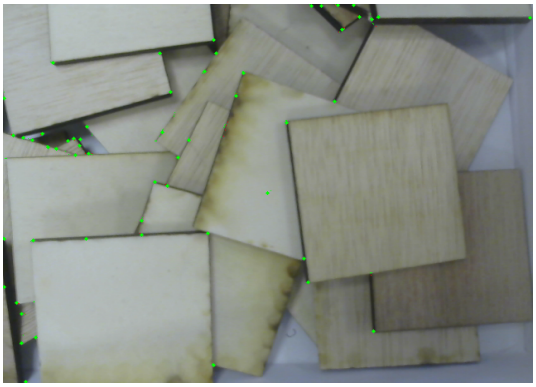

Fig. 21: Harris Corner Detector

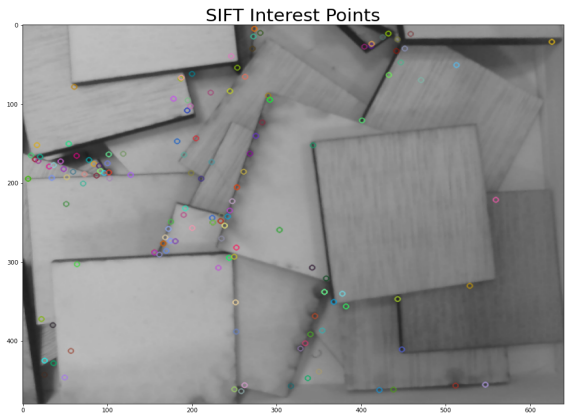

Fig. 22: SIFT method for Feature Extraction

In figure 22 there is an example of SIFT a very known algorithm used in *Computer Vision* and *Artificial Intelligence* to extract the key features in an image, so later can be used in different problems like image recognition. It is possible to observe that this algorithm has extracted practically the corners and contours of the pieces, that confirms the theory that corners and edges are very important features in *Image Recognition* . When the problem increases in complexity, this feature extraction techniques can be scarce. In the following section, a more automatic way to extract features will be presented using *Deep Learning* techniques.

2) Automatic Feature Extraction: The idea behind a feature extractor using *Deep Learning* is adding a structure of layers that will be able to extract the relations between the data and the most relevant characteristic of an image. The most important layers to extract the features are the *Convolutional Layers*. These layers act as filters and subtract the patterns in the image. The higher layers will get the more general features like edges and contours while the down layers will obtain the details and specifics of the images.

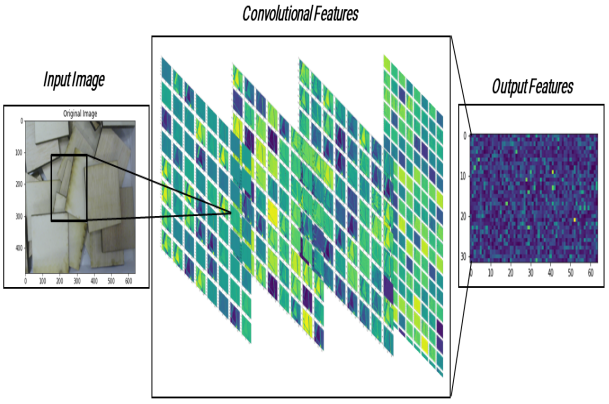

Fig. 23: Features extracted from the input image

In figure 23 there is a diagram showing how a CNN can work with feature extraction. The layers in the model focus on one part of the image and get the characteristics. At the end, the output will be a vector with the most relevant information of the image. Instead of using the complete image, we can use the vector to reduce dimensionality while keeping the important features.

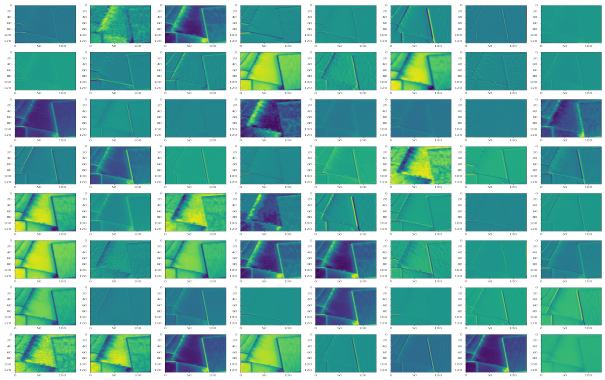

Fig. 24: First Convolutional Layer

The first convolutional layers are high-level filters, so in 24 the patterns are very clear and the filter is focusing in the edges and contours of the images. This layers are focusing in the just the center of the image, so the patterns belongs to the pieces.

The deeper the layer analysed, the patterns and the filters become harder to understand. Figure 25 represents a convolutional layer inside the whole architecture of a ResNet50 model. The patterns are starting to vanish.

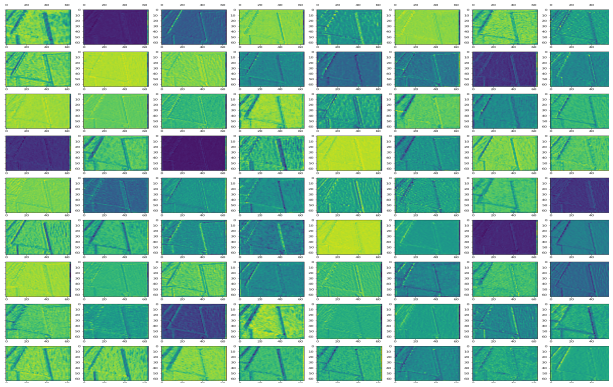

Fig. 25: Deeper Convolutional Layer

This last figure 26 is just a representation of the output feature vector from the model. It is impossible to visualize anything in particular due to the representation of the vector, which is just a flatten vector. However, even it is not possible to guess any pattern, the model can understand this information without any problem, and can be fed directly to the *Reinforcement Learning* algorithm.

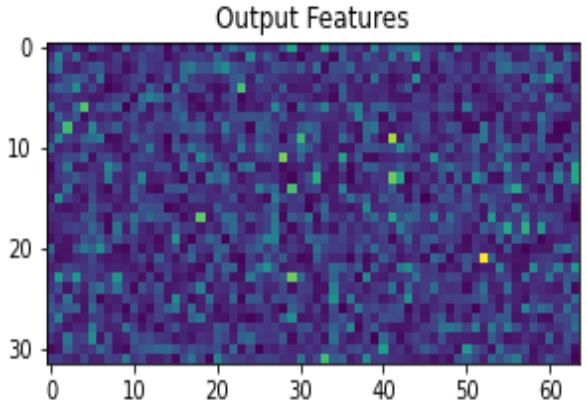

Fig. 26: Feature Vector Output

V. Image Classification

Image Classification is a fundamental task which goal is to comprehend an image as a whole. Classification consists in assigning a label to the image. This is the case of the bin-picking problem. Our image must be understood completely, In contrast with **Object Detection** problems (like the Block Detector) which involves not only classification but also location tasks.

The image enters the model, pre-processed (see IV-E1) and then the image will be evaluated in a trained model. The output of the model will be a prediction of belonging to 'fail' class or 'success' class. Once the inputs are ready to feed the model, it is time to focus on the model. Fitting models is a complex task due to the wide range of possible configurations and the parameters.

Due to this, a suitable solution is called **Transfer Learning**. This technique uses pre-trained networks, already tested and approved.

A. Transfer Learning

As explain in the state-of-art section Transfer Learning is a technique very recommended when working with images.

Transfer Learning is a technique very recommended when the dataset is small and getting new images can get difficult. For example, in this project, an image is taken only when the robot decides to make a pick movement. The dataset will increase only during the training of the robot.

As a result, Transfer Learning avoids the need of designing a model and training a model from scratch. This would require significant number of images, time and resources. Using the pre-trained models as feature extractors will simpler and more efficient.

a) ImageNet Dataset:: This dataset is one of the larger image dataset available. Most of pre-trained models are trained using this dataset. In the following table II there are most of the PyTorch available models for Transfer Learning. The Top 1 Error over the *ImagineNet* dataset will gives us some clues about which pre-trained models are better to start testing.

Model	Top-1 error
AlexNet	43.45
VGG16	28.41
VGG19	27.62
SqueezeNet	41.90
ResNet50	23.85
ResNet101	22.63
DenseNet169	24.00
Inception v3	22.55
GoogleNet	30.22
ShuffleNet v2	30.64
MobileNet v2	28.12
ResNetXt	20.69
Wide ResNet	21.49
MNASNet	26.49

TABLE II: Budget

From the results in table II the ResNet family of models are a good choice for starting the testing.

B. CNN

CNN is the standard *Deep Learning* technique for *Computer Vision* problems. In this section the common configuration for all the testing of models will be explained. This configuration remains the same but the Transfer Learning model and the tuning of the model can change. The configuration of the CNN model is the following:

1) Loss function: **Cross-Entropy**: "measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0." [5]
2) Optimizers: Algorithms or methods designed to change the parameters or attributes of a neural network. Some of these parameters are: learning rate or model weights. The main goal is to help reducing the loss function. Without optimizers, the internal parameters should be modified manually. The optimizer chosen for this project is **SGD**

The idea is to test the most algorithms as possible, so an extra layer is added to extract the features from the feature extractor easily in a flatten vector of one dimension. The last layer will be an Adaptive Average Pool with just one dimension. This will force all the features to have the same structure, so it can be easily adapted to the *Reinforcement Learning* algorithm.

```
    # 2. Adaptive layer:
    self.adaptive_layer = torch.nn.
    AdaptiveAvgPool2d(output_size=(1, 1))
    self.feature_extractor.add_module("
    adaptive_layer", self.adaptive_layer)
    # print(self.feature_extractor)

```

Snippet 3: Classification Layer

Following the guidelines of the previous section for tuning pre-trained models , it is necessary to configure the classifier

layer for our models. In classification problems, the last layers need to be a fully-connected layers.

```
    # classes are two: success or failure
    num_target_classes = 2

    n_sizes = self._get_conv_output(self.dim)

    # Classifier Layers
    _fc_layers = [torch.nn.Linear(n_sizes,
    256),
    torch.nn.Linear(256, 32),
    torch.nn.Linear(32, num_target_classes)]
    self.fc = torch.nn.Sequential(*_fc_layers
    )

```

Snippet 4: Adaptive Layer

This configuration of the model, adapts the last layer of the feature extractor and forces the output to be the same as the number of classes, in this case 'fail' and 'success'.

C. Metrics and Evaluation

In a *Machine Learning* project, it is very important to know how to answer the following question:

Is the model working?

To be able to answer the question it is essential to completely understand the problem. Models need to be checked, they need to be coherent. Metrics can be applied to follow the performance of a model. There are multiple metrics, but not all of them are adequate for all the problems. First, of all, it is important to understand the type of error that our classification model can make. There are two types of error:

- **Type I error** or **False positive**: Rejection of a true null hypothesis.
- **Type II error** or **False negative**: Failure to reject a false null hypothesis.

The *Null Hypothesis* in this problem is: "the robot failed". So, our *False Positive* error, will be when the robot chosed to pick the object thinking it will success but end up failing. On the other hand, a *False negative* error will occur when the robot decides that the action is a failure, even when it is successful.

The confusion matrix can be very useful tool to understand how the rates are distributed for every model:

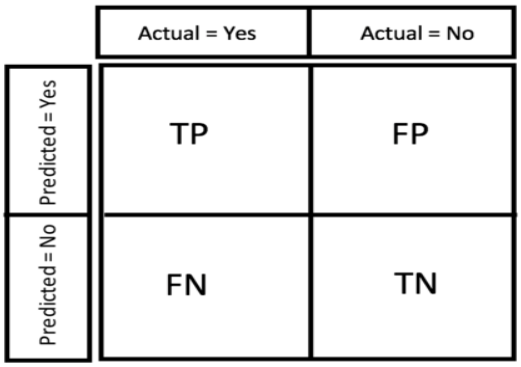

Fig. 27: Confusion Matrix [6]

The confusion matrix allows us to check how the model is performing in terms of classification, in a quick way and allows to decide if our model needs to focus on minimizing some type of error.

- **Accuracy**: The total number of items correctly identified. This metric represents the relation between the true results (true positives and tru negatives) and the total results.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Precision**: Number of items correctly classified as positive out of the total items identified as positive.

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

- **Recall**:Number of items correctly classified as positive out of the truly positive total items.

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

Accuracy, can seem like the metric more suitable for the classification problem. However, as mentioned in IV-D, the images in te dataset are not balanced. This means that the model will have a high accuracy but not a good performance. There are more images of "fail" in the dataset and that means that the model will predict most of the outputs as "fail". The model needs to predict successful output with higher accuracy. There are three options:

1) Balance the dataset
2) Select the importance of each class during traing
3) Focus on other metrics more representative. In this case, the precision can be a very interesting metric to follow, due to the necessity of detecting the true positives correctly trying to reduce the false positives. However, recall is also important, we would also like to detect more precisely the real successful picks. Remember that with the unbalanced dataset, the trend is to predict most of the outputs as 'failure'.

In our model, the penalty is bigger if the robot goes for a pick action and fails. This is the worst reward possible (See table of rewards for the *Reinforcement Learning* algorithm **??**). The scenario to avoid is the one where the prediction is successful but the robot fails in the task of picking the piece. The robot will be rewarded with a negative reward and this will decrease the overall performance of the project.

It is important to understand the trade-off between *Precision* and *Recall*. Besides, this metrics, there are some others metrics that can be useful to detect trends and to perform a follow-up of the developing of the models.

- **F1 Score**: is the Harmonic Mean of Precision and Recall.This measures

$$F1 = 2\frac{Precision * Recall}{Precision + Recall}$$

- **Fbeta Score**:

$$Fbeta = (1 + \beta^2)\frac{Precision * Recall}{(\beta^2 Precision) + Recall}$$

Measure	Beta	Description
F0.5	0.5	More weight on precision, less weight on recall.
F1	1	Balance the weight on precision and recall.
F2	2	Less weight on precision, more weight on recall

TABLE III: Fβ score [7]

- **Precision-Recall Curve**: Plot of the precision recall trade-off for different thresholds. Precision-Recall curve should only be used when specificity (True Negative Rate = 1- FPR) is no concern for the classifier. This is the case of our project, due to the problem of the unbalanced data.

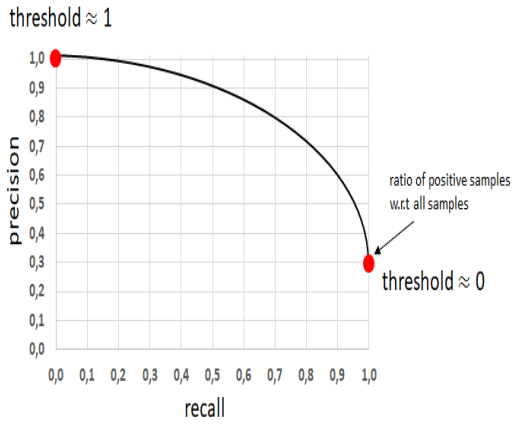

Fig. 28: Confusion Matrix [8]

- **AUC-ROC Curve**: AUC represents how good the model is distinguishing the classes. The higher the AUC, the more capable of identifying correctly the classes. Mathematically is calculated as the area under the curve of True Positive Rate vs. False Negative Rate. AUC value can go from 0 to 1. A random classifier would have a ROC-AUC value of 0.5 This score is independent of threshold set for classification, only considers the rank of each prediction and not the absolute value. [9]

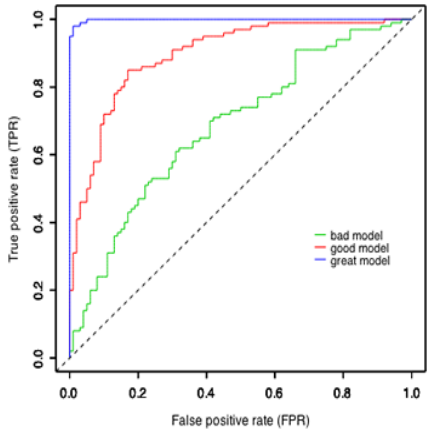

Fig. 29: Confusion Matrix [6]

In small datasets, sometimes the ROC-AUC Curve can present a more optimistic scenario, so F1 score and the complete Precision-Recall Curves can be useful to compare the performing of the model. For this project, the metrics to analyze the performance of the model will be the F1-score, the Precision-Recall Curve and the AUC-ROC Curve. All the metrics will be evaluated so the chosen model will be a model with a good performance in several metrics, rather than just one

VI. Results

The results of this project are several solutions that implement Computer Vision techniques with the ultimate goal of increasing robot performance. The model chosen is a CNN using Transfer Learning with a pre-trained model to increase the accuracy of the model when predicting and classifying.

A. Image Recognition

With the addition of the upper camera in the project architecture, there was an opportunity to increase the performance of the robot using *Object Detection*. By using this solution, we avoid random movements every time the robot picks a successful piece or when the robot goes off the limits of the environment.Instead, we relocate the robot in one of the places with more probability for picking pieces.

In figure **??** there is a representation of two original pictures, taken from the upper camera. The idea is to send the robot to the place where there are higher chances of picking pieces and, specially, to place it anywhere but the white spaces.

a) : Using the parameters obtained with the camera calibration (see IV-B), it is possible to link the 3D coordinates with the 2D pixel coordinates in the images. With this relationship, the contours of the pieces are detected using *Image Processing* and the points with more probabilities can be calculated. The coordinates of the optimal point are send to the robot for avoiding random or useless movements.

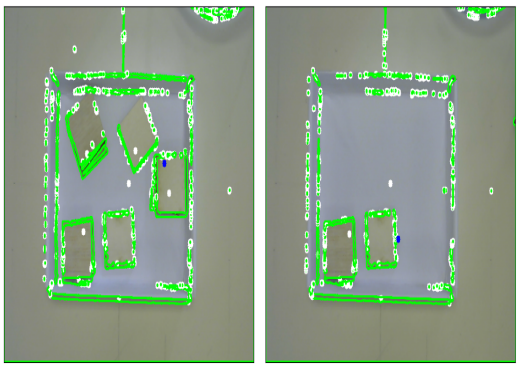

Fig. 30: Results after the contour detection

B. Image Classification

Choosing the model was the first problem when trying to solve the problem. There was not an initial dataset and training the first models was complicated due to the lack of data. The

initial goal was to get the larger dataset as possible, but with *Deep Learning* algorithms, our dataset was not enough.

Because of the need of working with images, the use of Transfer Learning is very common due to the already pre-trained models. These models were trained over huge image datasets such as ImageNet. However, choosing a pre-trained model is not easy due to the large possible selection of models. After several tests with the different pre-trained models, the models with an easier implementation and a better performance were the families of 'VGG' and 'ResNet'.

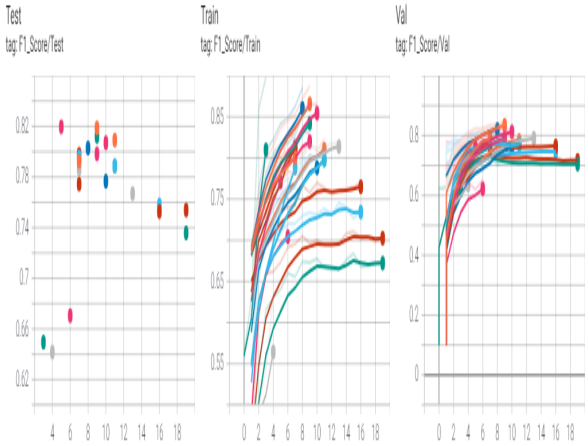

Fig. 31: F1 Comparison between VGG and ResNet models

Just comparing the results in *Tensorboard* where all the metrics from all the trainings performed are stored, the ResNet family has a better overall performance and the convergence is smoother.

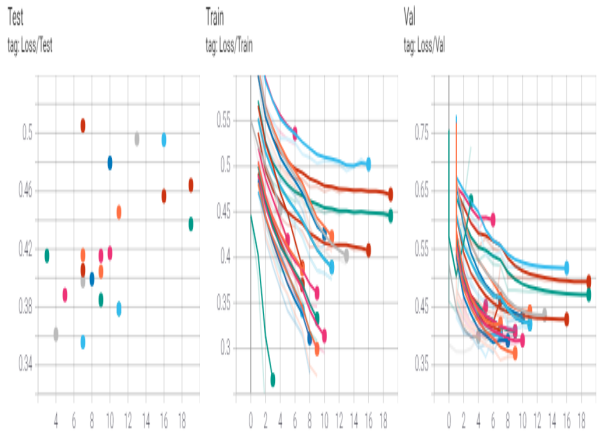

Fig. 32: Loss Comparison between VGG and ResNet models

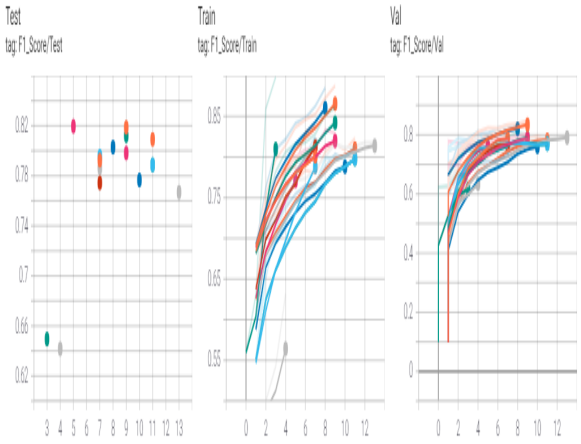

Fig. 33: F1 Comparison between ResNet models

When choosing the model, one of the most important characteristics to avid is a large difference between training, test and validation. If the model performs too well in trainig but it does not behave the same in validation and testing then, the mode is over-fit and it cannot be used.

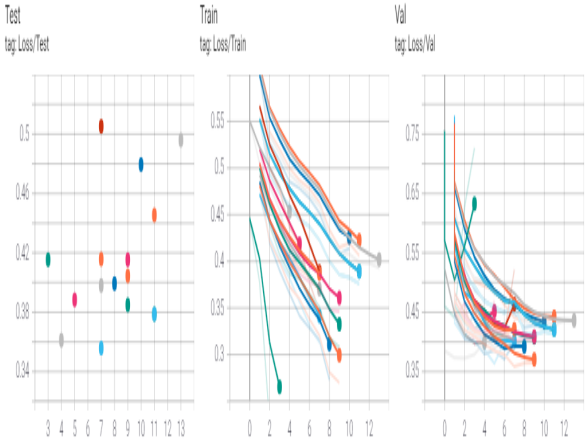

Fig. 34: Loss Comparison between ResNet models

Finally, a good fit for the model is found. The pre-trained model chosen for the CNN is the **ResNet50**. This model combines a good performance with a good training time. Besides a very easy model to understand and adopt. One of state-of-art pre-trained models for image classification. It is true that there are models with better performance, but for this project, the performance is adequate.

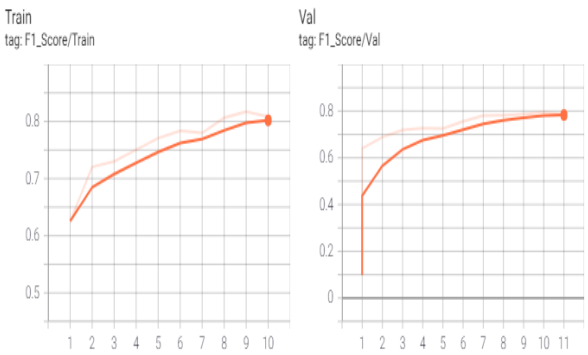

Fig. 35: F1 Metric for chosen model: ResNet50

In figure 36 the architecture of the model is represented. The pre-trained model is used for feature-extraction, with some fine-tuning of the last layer to adapt better to our dataset. Then, the rest of model is configures as mentioned in V-B

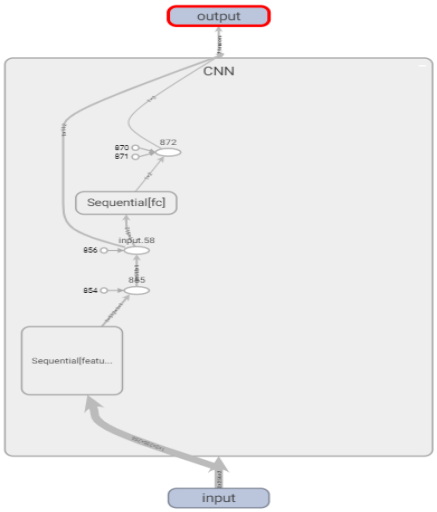

Fig. 36: Model architecture

During the whole development of the project, the main goal of the classifier was to detect the more accurate as possible, the success and failures of the picking movement from the robot. During the training of the *Computer Vision* model, the main goal was to avoid the over-fitting of the model, due to the use of *Deep Learning* with a small dataset, and to avoid the problem of unbalanced data. The *Confusion Matrix* is a very important indicator of how good the model is behaving. Balancing the dataset and controlling the weights of the classes in the training the model was able to detect better the successful picks. There are still False Positives (when the robot decides the movement will success but instead it fails), and less False Negatives (The robot predicts a failure, but ends up being a success). This means that our model predicts very well when the robot will fail. This is a good sign, because with the negative rewards, the model will tend to do pick actions only when the robot is sure about the movement. However, there is still the trade-off between precision and recall as seen previously, that can be improved.

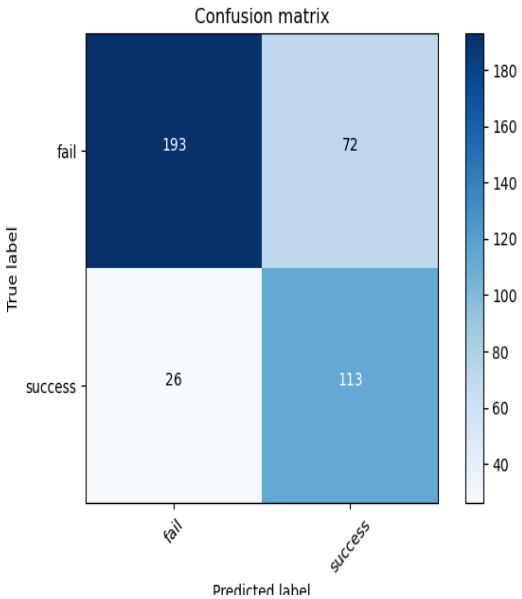

Fig. 37: Confusion Matrix

The trade-off between Precision and Recall has been one of the most difficult tasks to balance. With the imbalance dataset, and as seen in the Confusion Matrix 37 the model knows how to predict True Negatives, so there is no interesting in focusing on how the model predicts the failures. Precision and recall are two metrics with no knowledge of the True Negatives of the model. Precision-Recall Curve represents the plot of these two metrics for different thresholds. Seeing the curve in figure 38, the model is far from a no-skill classifier that cannot discriminate between classes. The no-skill line it is a horizontal line with the value of the positive cases in the dataset. In order to confirm the results, the AUC-ROC will be analysed.

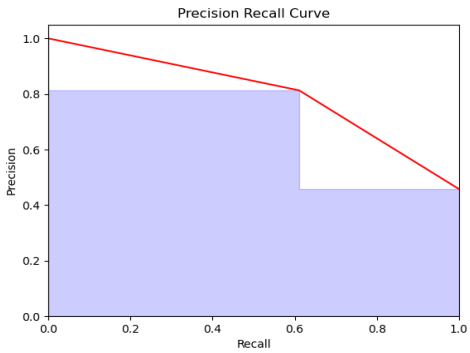

Fig. 38: Precision-Recall Curve

The AUC-ROC metric This metric represents the model's ability to correctly identify a class. This means that our model,

with a 77% probability, is able to identify whether an image is a success or a failure. Although this is not a perfect accuracy, it is a good start, especially since the pieces are very similar to each other. Even a human being might have difficulties to know in certain occasions if the piece could be caught or not by the robot.

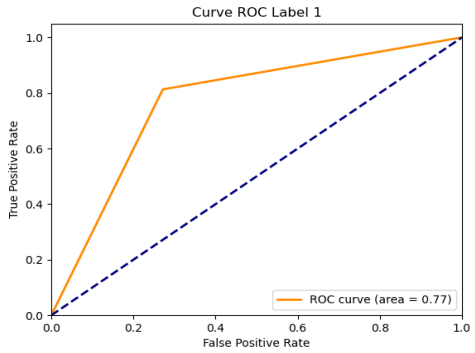

Fig. 39: AUC-ROC Curve

Once the model has been validated with the metrics, it is time to see the performance with real inputs. In 40 three new images have been selected. This images are not from the dataset, they have been selected from images not known by the model. These images are both fail pictures and one success.

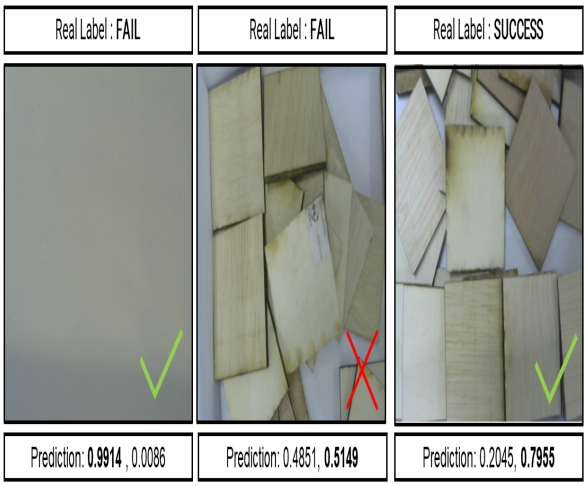

Fig. 40: Predictions from the model based on different inputs

1) The first image corresponds to a clear case of fail. In fact the box is emptied, so clearly the model predicts a failure with a **99%** of probability.
2) The second picture is a failure, but it is a more complex photo, where even humans will have difficulties to understand how the pick movement will develop. The same happens with the model. The prediction is wrong, the model classifies the picture as a success but with only a **51%** of probability, a random choice.
3) Finally, the last image was labelled as a success and the model predicted it with an almost **80%** of probability. Overall, the model perform correctly, although there is margin for improvement.This good performance will be very useful when adding the evaluation and the feature extracted to the *Reinforcement Learning* algorithm to complete the whole project and evaluate the results.

C. Bin-picking

The Bin-picking problem combines the *Artificial Intelligence* techniques: *Reinforcement Learning* and *Computer Vision* . In this section, the results of the whole project are exposed.

There were two main objectives for the bin-picking problem in the project:

1) Set the complete architecture for the project with the *Reinforcement Learning* and *Computer Vision* algorithms correctly implemented
2) Achieved a mean performance of two minutes for the robot to pick a successful piece in the environment. This means that almost every two minutes, the robot should make a successful pick if there are pieces in the box.

In *Reinforcement Learning* the improvement of the model can be seen over time, the algorithm is learning inline, with every new movement. The input of this algorithm is taken from the knowledge from the *Computer Vision* model. This means that if the input for the *Reinforcement Learning* algorithm is correct, the model should be learning and improving over time.Good results in the overall project also indicates good performance in the *Computer Vision* and the *Reinforcement Learning* algorithms. Good results are the best validation.

1) **Random actions per episode**: *Reinforcement Learning* problems are designed for learning 'inline'. This means that the algorithm uses every new input to learn and actualise the already learned knowledge. The algorithms starts with exploring the environment. At the beginning there is no information about the environment so the agent will have to explore with random movements and decisions, so the algorithm can start understanding the conditions of the environment, in the case of our project, the box with the pieces. However, the more the algorithm trains, the less random actions the robot will make.
2) **Evolution of rewards per episode** This metric can be analyses together with the Random actions per episode metric. When the robot starts using the knowledge of the model, the rewards should be increasing also. The goal of the *Reinforcement Learning* algorithm is to allow the robot to make the best decision, will make the robot acumulate more rewards per episode. If the rewards increase, is due to the improvement in the robot when picking pieces. The success picks increase positively the reward, while the rest of movements rewards a negative amount.
3) **Number of picks per episode**: When the robot is learning correctly, it is obtaining the information from

the images, and the coordinates. The robot knows the location of the gripper and what is underneath the pump. When extracting useful information from the inputs, the robot also learns the best trajectory to pick the pieces, so instead of trying random picks that can be failures, the robot needs less movements to achieve the same results.

4) **Number of steps per episode**: Very similar situation than the number of picks per episode. The robot starts needing a lot of movements to empty the box. Most of the movements are random and there is not enough knowledge to exploit it correctly. When the robot keeps training and the algorithm becomes more aware of the conditions, the robot will need less steps to achieve the same results. This is the ultimate goal of the bin-picking problem.The aim is for the robot to achieve at least a human performance.

In the following pictures, there are the results combining both *Reinforcement Learning* and *Computer Vision* techniques and the learning curve of the algorithms.

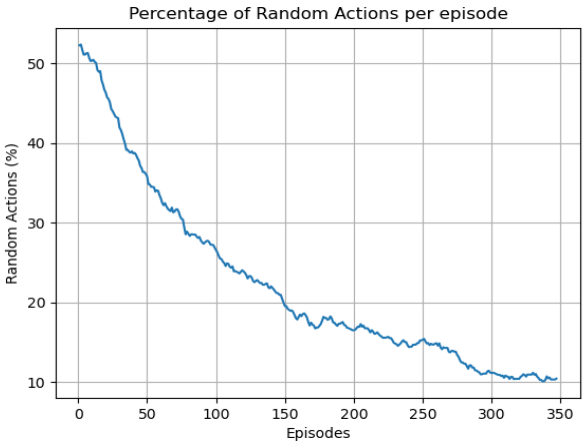

Fig. 41: Random actions per episode

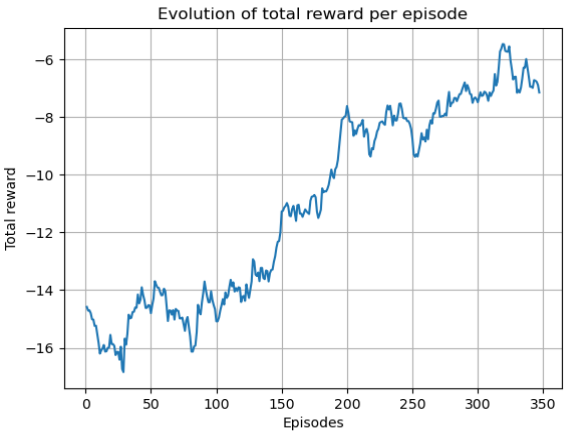

Fig. 42: Evolution of rewards per episode

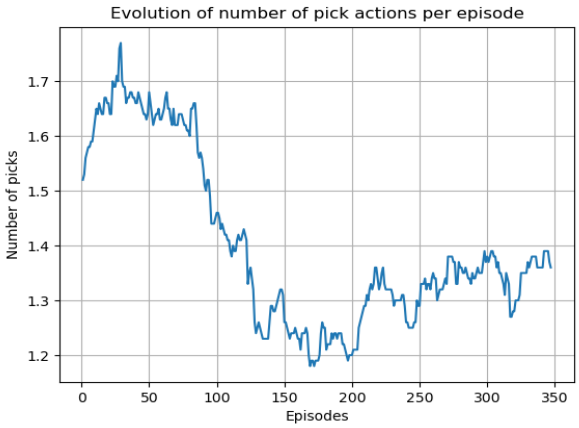

Fig. 43: Evolution of actions per episode

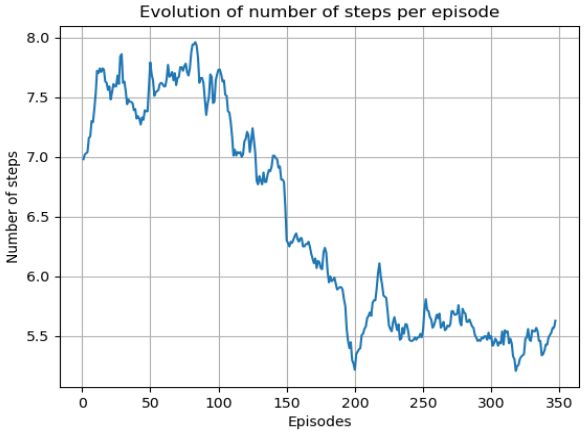

Fig. 44: Evolution of number of steps per episode

Here are the metrics to appreciate the improvement in the project:

Measure	Value
Steps	5044
Episodes	47
Pick actions	321
Objects Picked	30

TABLE IV: First training

As it can be seen in the table below IV the first impressions on the model were not very good. The model could try hundreds of picks without any successful pick. Due to the *Computer Vision* the model improved a lot and set the bases for keeping the improvement.

a) Is Computer Vision necessary for this project?: Maybe someone might think that with a *Deep Learning* algorithm for *Reinforcement Learning* could be enough. Images could be fed in the model almost ”raw” and the model will be able to learn and extract the features to make the best decisions.

This can be explained with results from the model.

- **Before *Computer Vision*** : the robot could spent hours trying to empty the box. There were times when the robot could not identified pieces or got locked in the same place without being able to recover.
- **After *Computer Vision*** : Then *Computer Vision* techniques and the CNN model with the Transfer Solution started to extract features and the classification for the images. The *Computer Vision* model and techniques were only focusing on treating the image and extracting the most useful information. This specific models were trained only for the images so with a good performance of the model, not only the dimensions of the image are reduced, but also the information is prepared to detect as accurate as possible how good the pick movement will be. Now, with the same amount of pieces, the whole box can be emptied in less than 30 minutes.

VII. Conclusions

This project has set the bases for the improvement and continuance of the bin-picking problem with *Artificial Intelligence* using *Computer Vision* and *Reinforcement Learning* techniques. However there is still a lot of margin for improvement. Projects concerning *Artificial Intelligence* are complex and they involved a lot of time. Everyday new techniques, models and new state-of-the art solutions are presented. This means that almost every *Machine Learning* problem has a wide range of improvement. An increase in performance or in speed of computation, can affect very positively in a model in production, especially in robotics and industrialized processes.

Parameter fitting can be one of the most complex tasks when training a model. There hyper-parameter settings affect the model, its convergence and performance. The model chosen for classification and feature extraction for *Computer Vision* has good performance and works correctly in identifying the successful or unsuccessful pictures. Nonetheless, it should be a good approach to train new models, refining the configuration and testing new solutions that due to the limited time it has not been possible to test.

Due to the nature of the project the dataset was unbalanced. Data imbalance is one of the most common problems in *Machine Learning* . This causes errors in predictions and evaluation and must be corrected before training the data. There are different techniques of balancing the datasets so that the training data is prepared to avoid poor performance.

Another conclusion drawn from this project is that there is a tendency to think that *Computer Vision* problems should be solved with complex techniques. However, sometimes simpler approaches can have faster results and adequate performance depending on the problem faced. There are very powerful techniques for *Image Processing* that are validated and have proven their effectiveness in a multitude of projects in recent years. Solutions based on neural networks are perfect for problems where we must adapt to the environment or conditions changing, therefore, perhaps the best practice is a combination of traditional techniques and neural networks.

Finally, one of the most demanding problems to solve was the integration of the whole architecture. All nodes communicate via TCP/IP, which implies that our structure depends on the network. If a failure occurs when writing on ROS *topics*, the other nodes can continue to wait for a message that will not arrive. These types of problems have been solved by using ROS services or sending a burst of messages so that message loss changes decrease significantly.

VIII. Future work

Projects concerning *Artificial Intelligence* are complex and they involved a lot of time. Everyday new techniques, models and new state-of-the art solutions are presented. This means that almost every *Machine Learning* problem has a wide range of improvement. An increase in performance or in speed of computation, can affect very positively in a production model, especially in robotics and industrialized processes.

Even though this project has set a good base for the *Computer Vision* and *Reinforcement Learning* techniques, there is still a lot of changes and upgrades that can be possibly done. This Bin-picking project was developed using a trial and error iteration.

A. Tune the models

Tuning the parameters in *Deep Learning* models can be one of the most complex tasks when training a model. There are hyper-parameters that affect the model and the model can vary depending on the values chosen. The model for *Computer Vision* has a good performance and it works correctly for classifying the successful or failure pictures. However, it should be a good approach to train new models, fine-tuning the configuration and test new solutions that due to the limited time it has not been possible to prove.

B. Implement new techniques

Due to the complexity of the project and some delays in the hardware, there were not enough time to prove new ideas of image implementation. Some of the examples to improve the information of the images in the model could be the use of a 3D camera or a implementation of a Multi-flash solution.

This Multi-flash solution takes four samples of the same image. The difference between these pictures is that each of them has a different flash lighting the scene. The images will represent the same environment but with different brightening effects. These changes allows to detect better the contours and the depth of the objects in the pictures. Shades and contrasts give an extra information, that can help the model to obtain better features than with a regular image. The implementation of this solution should be easy to add on the software part, the models just need to be adapted to receive the input of four images instead of one. The main difficulty should be the construction of the structure with the leds to recreate the four flashes. There are scientific articles suggesting an improvement in performance using this solution. Some examples can be found in these articles [10] and [11]

C. Try new solutions

The whole idea behind this project was inspired by the problem of Bin-picking in industrial sectors. However the goal was to try to replicate a commercial solution but without a scientific article behind. From the video some concepts such as the use of *Deep Learning* and *Computer Vision* could be extracted, but the solution was designed from scratch proving different techniques and implementing the whole infrastructure. The solution proposed in this project was one of the multiple possibilities that could have been implemented. For future work, maybe some of the variants could be the removal of the onboard camera, and just leave the upper camera with the whole view of the complete environment. Instead of using simple *Image Recognition* to detect the blocks, the dataset of the upper camera should increase and the model could be trained with these images. Other solution, could be just the use of *Computer Vision* to detect the pieces without the *Reinforcement Learning* algorithm. This idea can work to compare which one of the solutions is more optimal. A benchmark of implementations could be done, so it is possible to decide which solution has a better performance in our bin-picking task.

References

[1] Manning Publications. *Computer Vision Pipeline, Part 1: the big picture*. URL: https://manningbooks.medium.com/computer-vision-pipeline-part-1-the-big-picture-5d6a6964913a.

[2] Satya Mallick. *Camera Calibration using OpenCV*. URL: https://learnopencv.com/camera-calibration-using-opencv/.

[3] Satya Mallick. *Geometry of Image Formation*. URL: https://learnopencv.com/geometry-of-image-formation/.

[4] *Torchvision.Transforms*. URL: https://pytorch.org/docs/stable/torchvision/transforms.html.

[5] URL: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

[6] USF-Data Sciene. *Choosing the Right Metric for Evaluating Machine Learning Models — Part 2*. URL: https://medium.com/usf-msds/choosing-the-right-metric-for-evaluating-machine-learning-models-part-2-86d5649a5428.

[7] Jason Brownlee. *A Gentle Introduction to the Fbeta-Measure for Machine Learning*. URL: https://machinelearningmastery.com/fbeta-measure-for-machine-learning/.

[8] Thomas Kurbiel. *Gaining an intuitive understanding of Precision, Recall and Area Under Curve*. 2020. URL: https://towardsdatascience.com/gaining-an-intuitive-understanding-of-precision-and-recall-3b9df37804a7.

[9] Sarang Narkhede. *Understanding AUC - ROC Curve*. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[10] K. Tan R.S. Feris R. Raskar and M. Turk. *Specular reflection reduction with multi-flash imaging*. URL: https://www.researchgate.net/publication/4102103_Specular_reflection_reduction_with_multi-flash_imaging.

[11] Rogerio Feris. *Multi-Flash Stereopsis: Depth Edge Preserving Stereo*. URL: https://www.yumpu.com/en/document/view/5505197/multi-flash-stereopsis-depth-edge-preserving-stereo-citeseer.