
Master in Telecommunications Engineering

Master's final project

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

Author
María Pilar Hernández Bas

Supervised by
Philippe Juhel

Madrid
February 2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título "Computer Vision and Deep Learning for performance improvement in an UR3 robot"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2020-2021 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: María Pilar Hernández Bas Fecha: 10/ 02/ 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Philippe Juhel Fecha: 10/ 02/ 2020

**AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO**

***1º. Declaración de la autoría y acreditación de la misma.***

El autor D. María Pilar Hernández Bas

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: "Computer Vision and Deep Learning for performance improvement in an UR3 robot" que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

***2º. Objeto y fines de la cesión.***

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

***3º. Condiciones de la cesión y acceso***

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

***4º. Derechos del autor.***

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

***5º. Deberes del autor.***

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

To my family, for all the unconditionally love and support
To Philippe, for the guidance in this project
To my friends, for growing together
To Álvaro, for accompanying me all along the way

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

**Author: Maria Pilar Hernandez Bas**

Director: Philippe Juhel

## 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.

**Key words:** Artificial Intelligence, Computer Vision, Features, Feature extraction, CNN, Transfer Learning, ROS, UR3

## 1. 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.

# 2. Project Definition

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.

# 3. System Description

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.

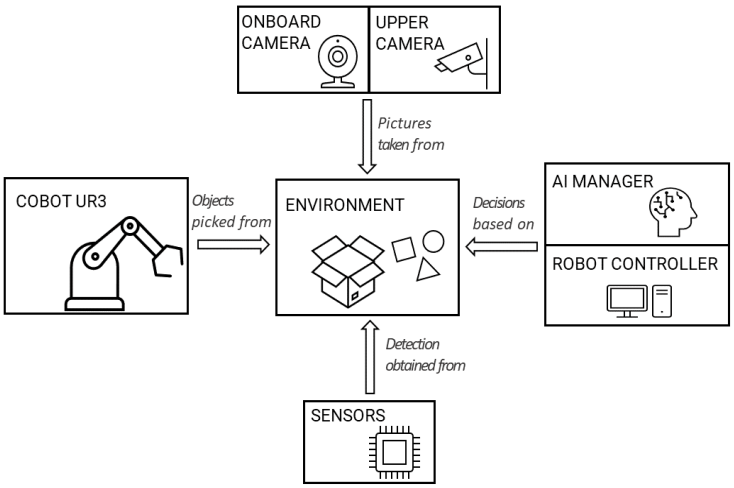



Figure 1: Complete architecture

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.

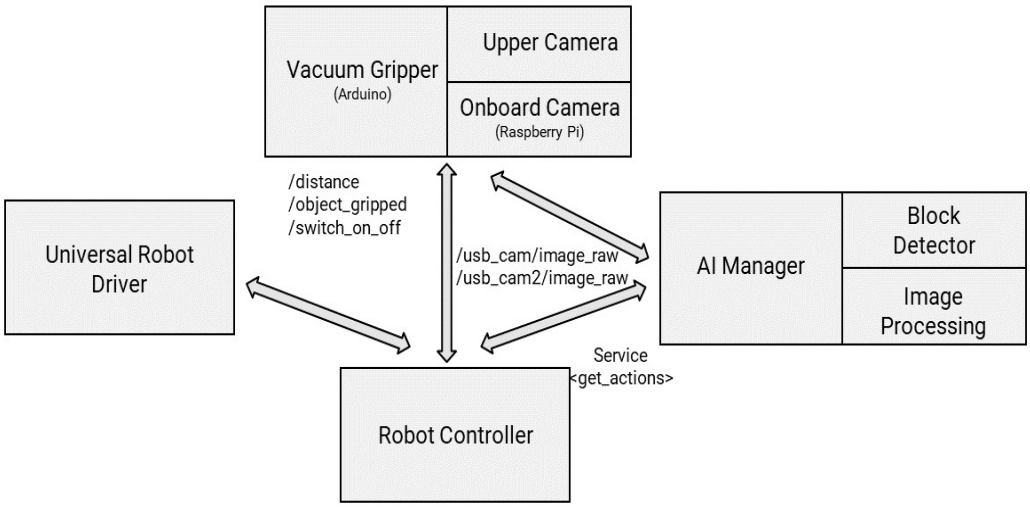



Figure 2: ROS architecture

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.

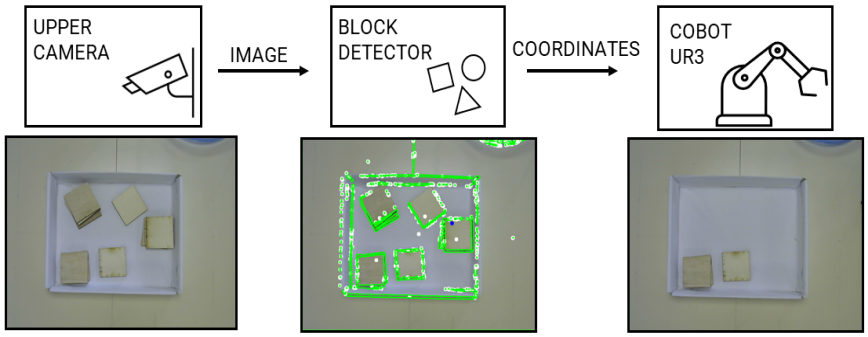



Figure 3: Diagram of How Block Detector works

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.

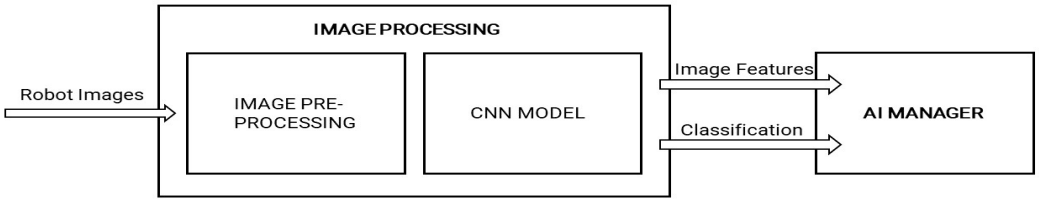



Figure 4: Image Processing steps

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.

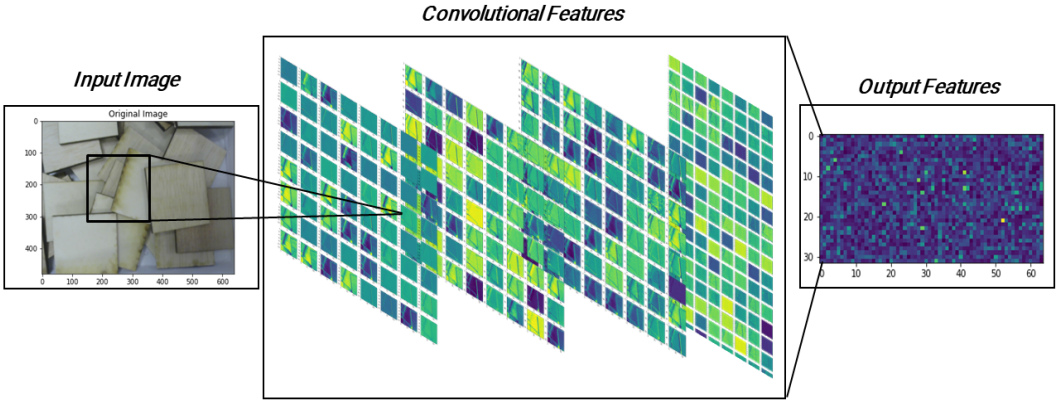



Figure 5: Features extracted from the input image

# 4. 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.

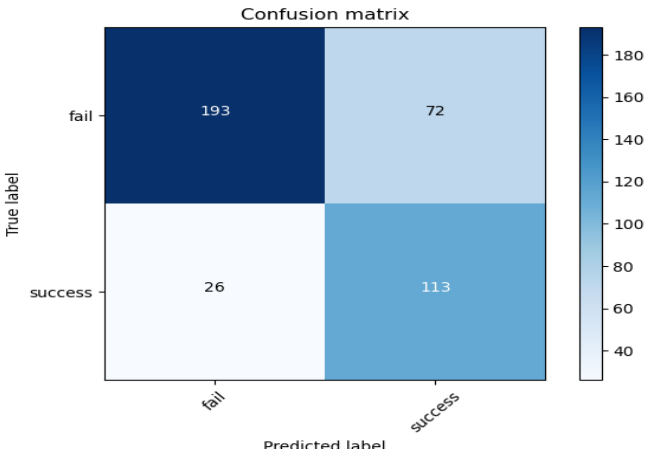



Figure 6: Confusion Matrix

Due to the nature of the project, our dataset has more signs of failure than success. When evaluating the model, we must take this into account and use metrics that focus on predicting the positive values, success in this case.

Once the model has been validated with the metrics, it is time to see the performance with real inputs.The model works correctly for 76% of the cases but it can get confused in some scenarios, where even humans can get confused about the prediction.

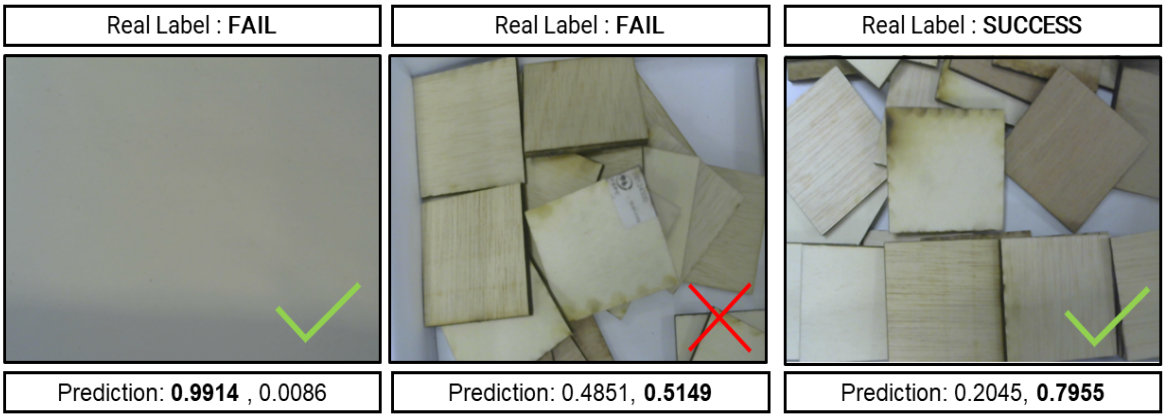
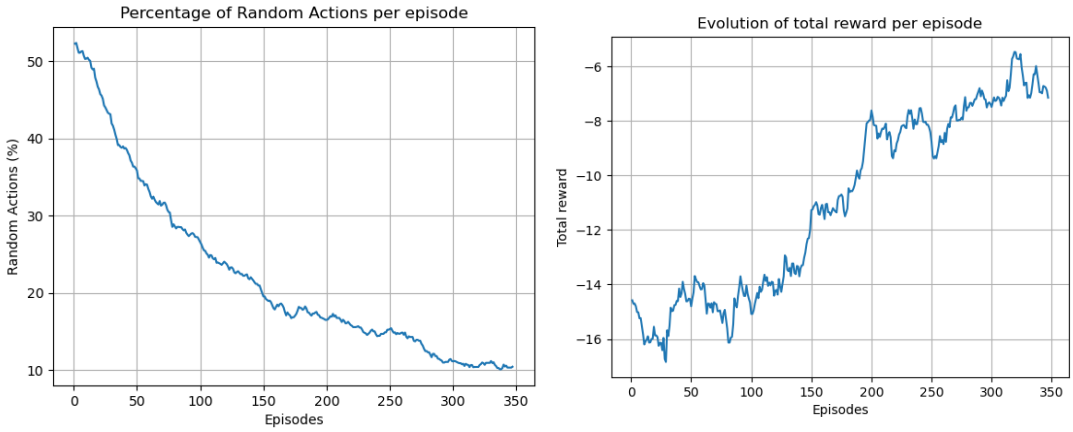



Figure 7: Predictions from the model based on different inputs

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.




(a) Random actions per episode

(b) Evolution of rewards per episode

Figure 8: Is the robot learning?

At the beginning the robot could spent hours trying to empty the box. Before *Computer Vision* techniques, the robot could not identified all the pieces and got locked in the same place without being able to recover. *Computer Vision* techniques and the CNN model are able to extract features and predict a classification value for the images. This specific models are trained only for the images, so the model can adapt to the specific features of the environment images. Besides, 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.

# 5. 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.

# 6. References


1. Universal Robot. *Universal Robots UR3* URL:
   $https://www.universal-robots.com/es/productos/robot-ur3/$

2. Open Robotics. *About ROS* URL:
   $https://www.ros.org/about-ros/$

3. Robotics Online. *Pick and Place Robots: What Are They Used For and How Do They Benefit Manufacturers?* URL:
   $https://www.robotics.org/blog-article.cfm/Pick-and-Place-Robots$

4. SAS.*Computer Vision, What it is and why it matters* URL:
   $https://www.sas.com/es_es/insights/analytics/computer-vision.html$

# Técnicas de Computer Vision y Deep Learning para la mejora de rendimiento de un robot UR3

**Autor: Maria Pilar Hernandez Bas**

Director: Philippe Juhel

## Resumen

Aprovechando la necesidad del sector industrial para optimizar sus procesos, se propone una solución de Inteligencia Artificial para un brazo robótico industrial. Las técnicas utilizadas serán de *Computer Vision*, referido al conjunto de técnicas que permiten a un ordenador extraer información de imágenes. El objetivo de nuestro proyecto es desarrollar una solución de Inteligencia Artificial que pueda ayudar al rendimiento de un brazo robótico industrial, un robot UR3. La implementación del proyecto se centra en estas técnicas de extración de información de las imágenes. A continuación, las características extraídas de las imágenes del entorno se introducen en un algoritmo de Aprenidizaje por refuerzo. El robot aprende de las imágenes y las coordenadas para decidir los mejores movimientos para su tarea de recolección conocida en inglés como *Bin-Picking*. Esta solución implementa una prueba de concepto que arroja resultados satisfactorios y sienta las bases y la arquitectura para la continuación y mejora del proyecto.

**Palabras clave:** Computer Vision, Inteligencia Artificial, CNN, ROS, Features, características, Aprendizaje por refuerzo

## 1. Introducción

Los robots industriales se utilizan hoy en día para automatizar procesos y realizar tareas de forma más eficiente. Con esto en mente, el objetivo es maximizar la utilidad de la máquina mientras disminuimos la cantidad de tiempo de ejecución para una tarea. La aplicación de técnicas como el Aprendizaje Automático, la Inteligencia Artificial o el Procesamiento de Datos ayudan a incrementar el rendimiento del robot para que puedan ser capaces de igualar o superar la actuación humana.

Por lo tanto, los datos son esenciales para todos los procesos de transformación del aprendizaje automático. Estos datos serán interpretados por las máquinas para extraer información relevante.

Uno de los tipos de datos más interesantes son las imágenes. Como entradas en un modelo, una imagen puede proporcionar información útil del entorno. Esta información puede introducirse en nuestros modelos de aprendizaje automático para poder predecir, por ejemplo, si una acción tendrá éxito o cuáles deben ser los próximos movimientos de nuestro robot.

Estas imágenes deben ser procesadas y preparadas para ser introducidas en los modelos. Los modelos esperan que las imágenes estén estandarizadas con algunas características comunes. Después, los modelos utilizarán las imágenes y las entradas para extraer las características más relevantes que se utilizarán en un entorno real.

# 2. Definición del proyecto

El objetivo de este proyecto es poder enseñar al robot a realizar una tarea comúnmente conocida como *Bin-Picking*. El robot realizará una serie de acciones con el objetivo de recoger objetos de un entorno, como una caja.

El robot debe tratar de recoger todas las piezas en el menor tiempo posible.Para lograr el enfoque más eficiente, se añade *Artificial Intelligence* para mejorar el rendimiento y obtener el mayor conocimiento posible del entorno.

El robot recogerá imágenes tomadas del entorno. Basándose en las imágenes del entorno, el robot tomará decisiones para intentar recoger los objetos mostrados con éxito y luego, dejarlos en el lugar deseado.

Esta es la idea principal del proyecto. Utilizar las técnicas y recursos disponibles para mejorar el rendimiento del *cobot*. Este proyecto 'Bin-Picking', está compuesto por dos técnicas principales de Machine Learning: *Computer Vision* y *Reinforcement Learning* . Este proyecto centrará la atención en las técnicas *Computer Vision* y cómo afecta al rendimiento y cómo el *Image Recognition* y *Image Processing* juegan un papel importante en este problema. Los resultados extraídos de los algoritmos *Computer Vision* se utilizarán en un algoritmo *Reinforcement Learning* y se evaluará todo el rendimiento del robot.

# 3. Descripción del sistema

La arquitectura del sistema se diseñó para adaptarse a las necesidades de los sistemas. Utilizando *ROS*, se crearon varios nodos interconectados, cada uno especializado en una tarea concreta, para lograr los objetivos del proyecto. Todos los componentes están diseñados específicamente para tener una mejor comprensión del entorno.

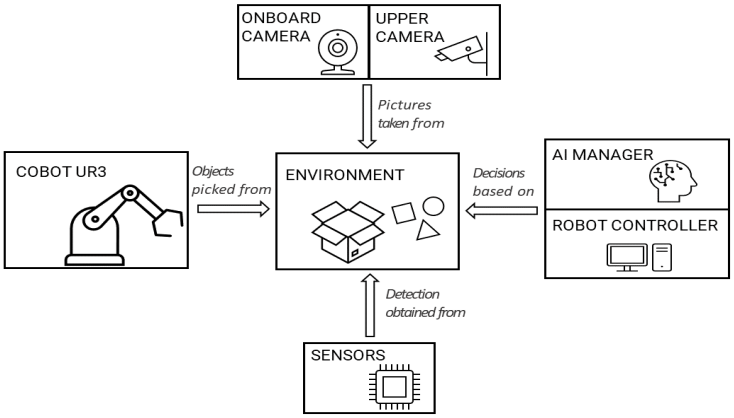



Figure 9: Complete architecture

Todos los nodos están conectados entre sí y necesitan enviar y recibir mensajes. Por defecto, todos los mensajes serán enviados por *ROS topics*, sin embargo, la información más crítica, como las acciones para el robot, necesitan ser confirmadas para que no se pierda información. El nodo **AI Manager** utiliza las salidas del nodo **Image Processing** y decide, con el algoritmo de aprendizaje por refuerzo, las acciones que el robot debe realizar. Estas acciones son enviadas al nodo **Robot Controller**, encargado de controlar y comunicarse con el robot.

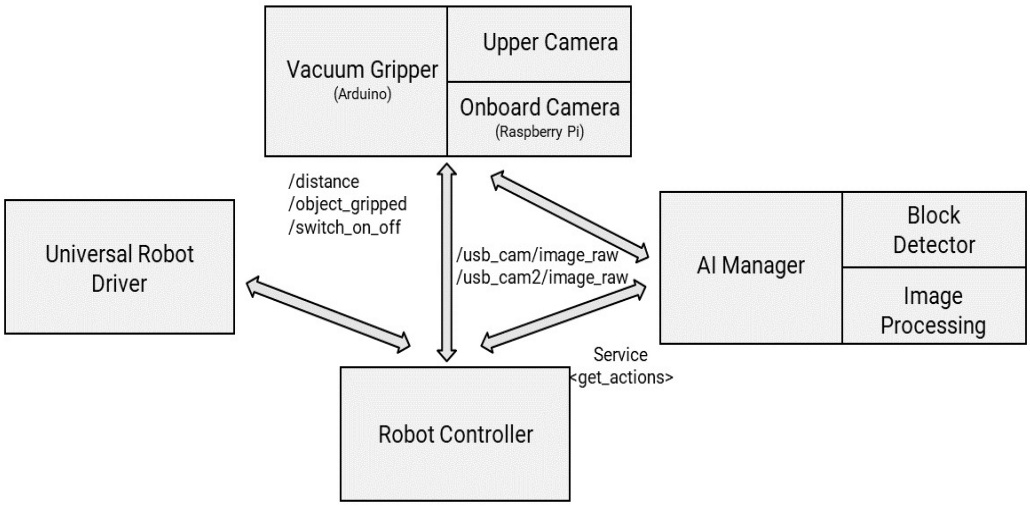



Figure 10: ROS architecture

También hay un nodo importante, encargado de determinar los lugares donde el robot debe colocarse al final de una recogida exitosa o cuando el robot se sale

de los límites. Este nodo, evita los movimientos aleatorios y ayuda a aumentar la precisión del robot. El nodo es **Detector de bloques** y utiliza técnicas *Computer Vision* tradicionales para reconocer los puntos de más probabilidades para recoger piezas.

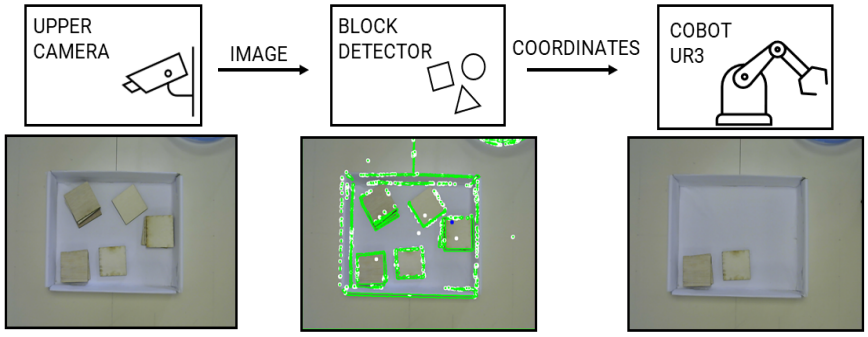



Figure 11: Diagram of How Block Detector works

Finalmente, el foco principal del proyecto es el nodo **Image Processing**. En este nodo, las imágenes se toman de la cámara a bordo del robot. Estas imágenes son pre-procesadas y estandarizadas, por lo que el modelo espera siempre el mismo tipo de imágenes con las mismas características como tamaño, color y apariencia. Una vez que las imágenes están listas para ser utilizadas en el modelo, se dividen en conjuntos de datos de entrenamiento y de validación. Los datos se entrenan con un modelo de red neuronal, **CNN**. El modelo extrae las características de la imagen (lo que la hace única) y la clasificación de la imagen. El modelo predice si la imagen será un "éxito" o un "fracaso" para que el siguiente paso pueda tener la mayor información posible para alimentar el algoritmo de *Aprendizaje por refuerzo*

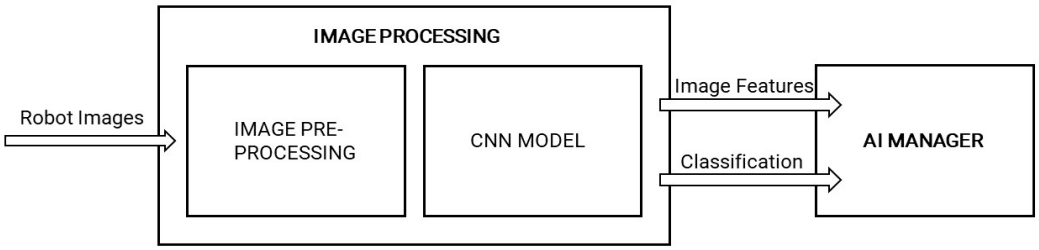



Figure 12: Image Processing steps

Estas características, junto con las coordenadas y la predicción del modelo, servirán de entrada al modelo de aprendizaje por refuerzo que servirá para determinar los movimientos del robot.

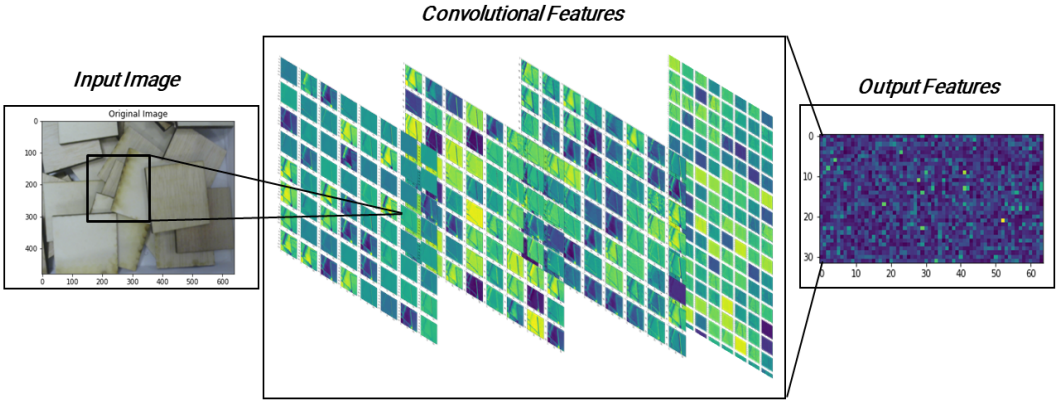



Figure 13: Features extracted from the input image

# 4. Resultados

Los resultados de este proyecto son varias soluciones que implementan técnicas de Visión por Computador con el objetivo final de aumentar el rendimiento del robot. El modelo elegido es una CNN que utiliza el Aprendizaje de Transferencia con un modelo preentrenado para aumentar la precisión del modelo a la hora de predecir y clasificar.

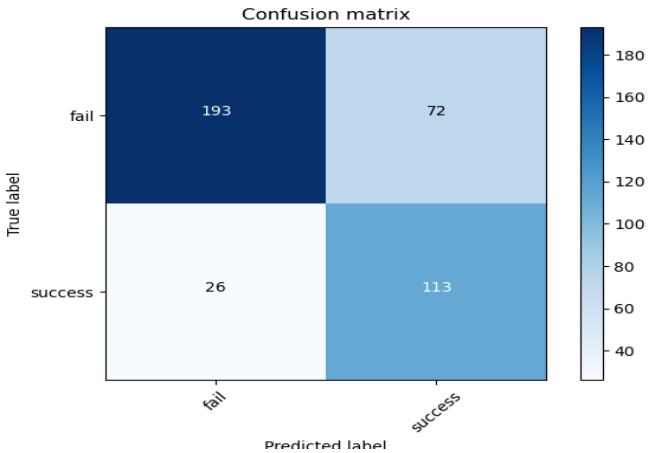



Figure 14: Confusion Matrix

Debido a la naturaleza del proyecto, nuestro dataset tiene más muestras de fracaso que de éxito. A la hora de evaluar el modelo, debemos tener esto en cuenta y utilizar métricas que se centren en la predicción de los valores positivos, de éxito en este caso.

Una vez que el modelo ha sido validado con las métricas, es el momento de ver el rendimiento con entradas reales.El modelo funciona correctamente para el 76% de los casos, pero puede confundirse en algunos escenarios, donde incluso los humanos pueden confundir la predicción.

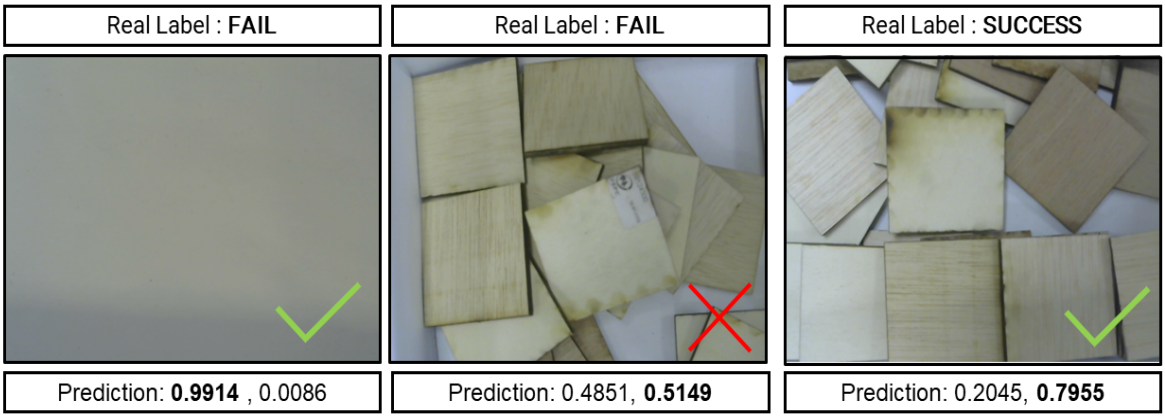
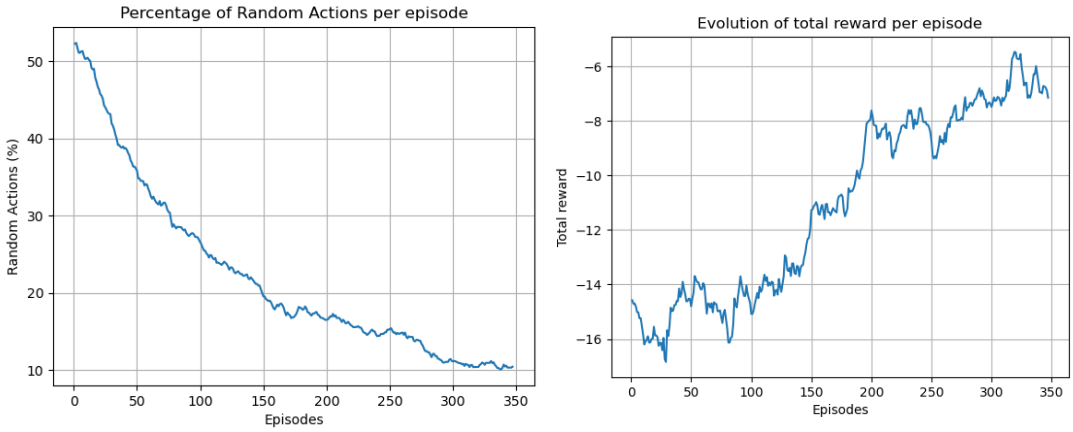



Figure 15: Predictions from the model based on different inputs

En *Aprendizaje por Refuerzo* la mejora del modelo puede verse a lo largo del tiempo, el algoritmo va aprendiendo en línea, con cada nuevo movimiento. La entrada de este algoritmo se toma del conocimiento del modelo *Computer Vision* . Esto significa que si la entrada para el algoritmo *Reinforcement Learning* es correcta, el modelo debería estar aprendiendo y mejorando con el tiempo.Los buenos resultados en el proyecto global también indican un buen rendimiento en los algoritmos *Computer Vision* y *Aprendizaje por refuerzo*. En este caso, en la imagen podemos ver que el modelo acumulando mayores recompensas según avanzan los episodios, el principal objetivo de estos algoritmos. Confirma que el robot va aprendiendo y optimizando los movimientos.




(a) Random actions per episode

(b) Evolution of rewards per episode

Figure 16: Is the robot learning?

Al principio, el robot podía pasar horas intentando vaciar la caja. Antes de las técnicas de *Computer Vision* , el robot no podía identificar todas las piezas y se quedaba encerrado en el mismo sitio sin poder recuperarse. Las técnicas de *Computer Vision* y el modelo CNN son capaces de extraer características y predecir un valor de clasificación para las imágenes. Estos modelos específicos se entrenan sólo para las imágenes, por lo que el modelo puede adaptarse a las características específicas de las imágenes del entorno. Además, no sólo se reducen las dimensiones de la imagen, sino que también se prepara la información para detectar con la mayor exactitud posible el movimiento del pico. En la actualidad, con la misma cantidad de piezas, se puede vaciar toda la caja en menos de 30 minutos.

# 5. Conclusiones

Este proyecto ha sentado las bases para la mejora y la continuidad del problema de recolección de objetos gracias al uso de *Artificial Intelligence* y las técnicas *Computer Vision* y *Aprendizaje por Refuerzo*. Sin embargo, todavía hay un amplio margen de mejora. Los proyectos relacionados con *Inteligencia Artificial* son complejos y requieren mucho tiempo de entrenamiento y recolección de datos. Además, cada día se presentan nuevas técnicas, modelos y soluciones innovadoras. Esto significa que la mayoría de los problemas de *Machine Learning* tienen muchas probabilidades de mejorar sus resultados. Un aumento en el rendimiento o en la velocidad de cálculo, puede afectar muy positivamente en un modelo en producción, especialmente en robótica y procesos industrializados.

El ajuste de los parámetros en *Deep Learning* puede ser una de las tareas más complejas a la hora de entrenar un modelo. La configuración de los hiperparámetros afectan al modelo, a su convergencia y al rendimiento. El modelo escogido para clasificación y extracción de features de *Computer Vision* tiene un buen rendimiento y funciona correctamente para identificar las imágenes de éxito o de fracaso. Sin embargo, debería usarse como una buena aproximación para entrenar nuevos modelos, afinando la configuración y probando nuevas soluciones que debido al limitado tiempo no ha sido posible probar.

Debido a la naturaleza del proyecto el conjunto de datos estaba desbalanceado. El desequilibrio de los datos es uno de los problemas más comunes en *Machine Learning* . Esto provoca errores en las predicciones y en la evalucación, Esto debe corregirse antes de entrenar los datos y se deben elegir las métricas correctas en función del problema. Existen diferentes técnicas para equilibrar los conjuntos de datos de forma que los datos de entrenamiento estén preparados para evitar un mal rendimiento.

Otra de las conlusiones extraídas de este proyecto es la tendencia a pensar que los problemas de *Computer Vision* deben resolverse con técnicas complejas, sin embargo, veces enfoques más sencillos pueden tener resultados más rápidos y con un rendimiento adecuado dependiendo del problema al que se enfrenten. Existen técnicas muy potentes para el procesamiento de imágenes que están validadas y han demostrado su eficacia en multitud de proyectos en los últimos años. Las soluciones basadas en redes neuronales son perfectas para problemas en las que nos debemos adaptar al entorno o las condiciones cambian, por lo que quizá la mejor práctica sea una combinación entre las técnicas tradicionales y el *Deep Learning* .

# 6. Referencias


1. Universal Robot. *Universal Robots UR3* URL:
   $https://www.universal-robots.com/es/productos/robot-ur3/$

2. Open Robotics. *About ROS* URL:
   $https://www.ros.org/about-ros/$

3. Robotics Online. *Pick and Place Robots: What Are They Used For and How Do They Benefit Manufacturers?* URL:
   $https://www.robotics.org/blog-article.cfm/Pick-and-Place-Robots$

4. SAS.*Computer Vision, What it is and why it matters* URL:
   $https://www.sas.com/es_es/insights/analytics/computer-vision.html$

# Contents


**1 Introduction** **1**

**2 Technology Description** **3**
2.1 Universal Robots . . . . . 4
2.2 ROS . . . . . 4
2.2.1 Components . . . . . 5
2.2.2 Features . . . . . 6
2.2.3 Integration . . . . . 7
2.3 Python . . . . . 7
2.4 OpenCV . . . . . 8
2.5 PyTorch . . . . . 8
2.5.1 Pytorch vs. Tensorflow . . . . . 9
2.5.2 Pytorch Lightning . . . . . 10
2.5.3 Tensorboard . . . . . 11
2.6 Libraries and tools . . . . . 13

**3 State of the art** **15**
3.1 Robotics . . . . . 15
3.2 Artificial Intelligence . . . . . 16
3.3 Computer Vision . . . . . 17
3.4 Machine Learning . . . . . 18
3.5 Deep Learning . . . . . 21
3.5.1 CNN . . . . . 23
3.5.2 Transfer Learning . . . . . 24
3.6 Reinforcement Learning . . . . . 26
3.6.1 DQ-Learning . . . . . 29
3.7 Feature Engineering . . . . . 29

**4 System Description** **32**
4.1 Motivation . . . . . 32
4.2 Objectives . . . . . 33

4.3 Methodology . . . 34
4.4 Planning . . . 35
4.5 Resources . . . 37

**5 Solution** **38**
5.1 Project Overview . . . 38
5.2 System Architecture . . . 40
5.2.1 The environment . . . 44
5.2.2 Vacuum Gripper . . . 44
5.2.3 Robot Controller . . . 45
5.2.4 Image Processing . . . 46
5.2.5 Block Detector . . . 47
5.2.6 AI Manager . . . 48

**6 Computer Vision** **50**
6.1 General Overview . . . 50
6.2 Camera Calibration . . . 51
6.3 Data Gathering . . . 54
6.4 Understanding the image . . . 56
6.5 Image Pre-processing . . . 58
6.5.1 Image Pre-processing . . . 58
6.5.2 Data Augmentation . . . 65
6.6 Feature Extraction . . . 66
6.6.1 Manual Feature Extraction . . . 67
6.6.2 Automatic Feature Extraction . . . 69
6.7 Image Classification . . . 72
6.7.1 Transfer Learning . . . 72
6.7.2 CNN . . . 73
6.8 Metrics and Evaluation . . . 74

**7 Results** **79**
7.1 Image Recognition . . . 79
7.2 Image Classification . . . 81
7.3 Bin-Picking . . . 88

**8 Conclusions** **92**
8.1 The problem of unbalanced data . . . 92
8.2 Tuning a model . . . 93
8.3 Is Deep Learning always the best option? . . . 93
8.4 From theory to practice . . . 94
8.5 When is enough? . . . 94

8.6 Complex structures and points of failure . . . . . 95

**9 Future work** **96**
9.1 Tune the models . . . . . 96
9.2 Implement new techniques . . . . . 96
9.3 Try new solutions . . . . . 97

**10 Aligment with the Sustainable Development Goals** **98**

**Bibliography** **100**

# List of Figures


1 Complete architecture . . . . . vii
6 Confusion Matrix . . . . . ix
9 Complete architecture . . . . . xv
14 Confusion Matrix . . . . . xvii

2.1 Universal Robots UR3 cobot . . . . . 4
2.2 ROS Components (Source: [4]) . . . . . 5
2.3 Tensor . . . . . 9
2.4 Use of *Machine Learning* frameworks in the industry [13] . . . . . 10
2.5 Pytorch vs Pytorch Lightning . . . . . 11
2.6 Example of Tensorboard . . . . . 12
2.7 Tensorboard in the industry [13] . . . . . 13

3.1 Hype Cycle for Emerging Technologies, 2020 by Gartner . . . . . 17
3.2 Artificial Intelligence relationships . . . . . 19
3.3 Machine Learning adoption in enterprises . . . . . 20
3.4 Most common methods and algorithms . . . . . 20
3.5 An example of Neuron . . . . . 21
3.6 CNN architecture . . . . . 23
3.7 Convolutional Layers . . . . . 24
3.8 Pooling Layers . . . . . 24
3.9 Image Classification on ImageNet . . . . . 25
3.10 How to fine-tune in Transfer Learning? . . . . . 26
3.11 Types of *Reinforcement Learning* algorithms [33] . . . . . 27
3.12 Markov Decision Process [38] . . . . . 28
3.13 Percentage of time for Data Science tasks [42] . . . . . 30

4.1 Gathered images as input in Deep Learning Model . . . . . 33
4.2 Hardware activities . . . . . 36
4.3 AI Manager activities . . . . . 36
4.4 Robot Controller activities . . . . . 37

5.1 Real view of the project environment . . . 38
5.2 Complete architecture . . . 39
5.3 Environment from different perspectives . . . 40
5.4 ROS architecture . . . 41
5.5 Vacuum Gripper for the robot . . . 45
5.6 Image Processing steps . . . 47
5.7 Diagram of How Block Detector works . . . 48

6.1 Computer Vision Pipeline [45] . . . 50
6.2 Chess board for Camera Calibration . . . 53
6.3 Camera already calibrated . . . 54
6.4 Example of image taken from the onboard camera . . . 55
6.5 Distribution of dataset . . . 56
6.6 Colour distribution of the image . . . 58
6.7 Center Crop . . . 59
6.8 Resized images . . . 60
6.9 Blur techniques . . . 60
6.10 Blur techniques . . . 61
6.11 Threshold techniques . . . 62
6.12 Threshold techniques . . . 63
6.13 Brightness and contrast . . . 64
6.14 Different data augmentation techniques . . . 66
6.15 Original Image vs. Cluster features . . . 67
6.16 Canny Edge Detector . . . 68
6.17 Harris Corner Detector . . . 68
6.18 SIFT method for Feature Extraction . . . 69
6.19 Features extracted from the input image . . . 70
6.20 First Convolutional Layer . . . 70
6.21 Deeper Convolutional Layer . . . 71
6.22 Feature Vector Output . . . 71
6.23 Confusion Matrix [50] . . . 75
6.24 Confusion Matrix [52] . . . 77
6.25 Confusion Matrix [50] . . . 78

7.1 Original raw pictures . . . 80
7.2 Results after the contour detection . . . 80
7.3 F1 Comparison between VGG and ResNet models . . . 81
7.4 Loss Comparison between VGG and ResNet models . . . 82
7.5 F1 Comparison between ResNet models . . . 82
7.6 Loss Comparison between ResNet models . . . 83
7.7 F1 Metric for chosen model: ResNet50 . . . 83

7.8 Model architecture . . . . . 84
7.9 Confusion Matrix . . . . . 85
7.10 Precision-Recall Curve . . . . . 86
7.11 AUC-ROC Curve . . . . . 86
7.12 Predictions from the model based on different inputs . . . . . 87
7.13 Is the robot learning? . . . . . 89
7.14 Is the model improving? . . . . . 90

# List of Tables

4.1 Budget . . . . . 37

5.1 Environment parameters . . . . . 44
5.2 AI Manager rewards . . . . . 49

6.1 Characteristics of the image . . . . . 57
6.2 Budget . . . . . 73
6.3 F$\beta$ score [51] . . . . . 77

7.1 First training . . . . . 90

# List of Snippets


5.1 Gripper communication: switch on/off topic . . . . . 42
5.2 Gripper communication: distance topic . . . . . 42
5.3 Gripper communication: object gripped topic . . . . . 42
5.4 Upper Camera topic . . . . . 43
5.5 Robot Controller service . . . . . 43
5.6 AI Manager service server . . . . . 43
5.7 Onboard camera topic . . . . . 44
6.1 Weighted Random Sampler . . . . . 57
6.2 Classification Layer . . . . . 74
6.3 Adaptive Layer . . . . . 74

# Chapter 1

# 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.

If we want machines to perform as humans or even better, it is necessary to program them efficiently so they can obtained data and use it to choose the best decisions automatically. This would require a transformation process, in which, the result can be called as an "intelligent machine". 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 so it can be converted into something useful for the problem that should be solved.

There are different types of data that can be fed into models. Almost everything can be considered data. This information can be categorize depending on how it is presented. Structure data, for example, refers to all the information that can be presented into a table format. This data can be structured into a relational database. The information will be represented as labels, attributes and relationships. However, the major amount of information nowadays is presented in an unstructured way. Typically unstructured data includes: text files, images or audio. 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.

The goal for this project is to be able to teach the robot how to perform a task commonly known as *Bin-Picking*. The robot will perform a set actions with

the objective to collect objects from an environment, such as a box. Based on the images of the environment the robot will make decisions to succeed in picking objects and leave them in the desired place. The robot should try to pick all the pieces in an efficient way, incorporating and improving with *Reinforcement Learning* techniques, similar to a human being. Is in the process of learning and understanding the environment when *Artificial Intelligence* is added to the equation.

The quality of the data is one of the main concerns in every project. The model should learn from all the inputs received. It is important to build the best dataset that can improve the performance and the utility. With no data or no good data it is very complicated to get a model to work as it would be desired.

Computer Vision techniques such as *Image Recognition* and *Image Processing* can be applied to multiple scenarios, not only to industrial robots. Images can be key inputs in an enormous variety of problems. For example, anomaly detection systems or medical image recognition for medical scans. Images can improve predictions and performances. It is vital to collect all the knowledge, best practices, and techniques to put images at the service of innovation. 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.

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* . Due to the complexity of the problem, this project described here 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.

During the following chapters, the main goal will be to understand and evaluate different techniques and tools for processing and understanding the information that images provide while integrating the knowledge obtained with the whole architecture and the *Reinforcement Learning* module.

# Chapter 2

# Technology Description

In this chapter, the technologies used for the project will be described. The selection of the different tools is considered based on the constrictions of the hardware necessary for the project. It is important to understand the restrictions and the initial situation of the problem. The project will be developed in a real environment, a robotic industrial arm, so before designing a solution, it is important to understand how the connection with the robot should be done.

Once the interaction with the main component of the project is understood, the rest of the technologies and tools will be chosen based on the advantages, the facility of implementation and how can interact with the requirements on the hardware.

The project is focused on a bin-picking, the robot should be able to pick and place objects from one box to another. The goal is to achieve a good performance using Computer Vision and Reinforcement Learning. It can be appreciated, that in this development, there are software and hardware components.

With these in mind, the principal decisions about the technologies for our design will be:

- Software to interact with the robot and the complete architecture.
- Programming language
- Machine Learning tools and libraries

Before explaining the reasons behind the election of technologies, it is important to understand how the robot works and what are the possibilities for interacting with it. The other technologies will be described afterwards.

## 2.1 Universal Robots

As explained before, the main component in the project is a robotic arm. More specifically, the model is a UR3 robot by Universal Robots. The UR3 is an industrial, lightweight and small robot, ideal to set in small spaces or in work tables. The robot weights 11 kg and it is capable of loading 3 kg. It is composed by 6 motors, with a rotation on 360 grades in each articulation. [1]

Figure 2.1: Universal Robots UR3 cobot

This type of robots are known as **cobot** (*Collaborative Robots*). What is the difference between a *cobot* and an *industrial robot*?. The *traditional robots* are bigger, the production is massive and normally they are placed in a fixed position. Cobots are smaller and can be easily placed anywhere. Bigger robots can be harmful and should be installed with barriers to ensure security. These robots lack of load sensors that would allow the machine to control the environment and stop in case of necessity or risk. On the other side, *cobots* are equipped with sensors which allow them to stop if there is any type of obstructions. In conclusion, the mission of *cobot* is to remove the barriers so they can be integrated in any type of production chain. [2]

## 2.2 ROS

*Robot Operating System* also known as **ROS** *ROS* is a collaborative project for robotics software development. *ROS* is a flexible framework composed of a set of

libraries, tools and conventions with the task of helping develop robot behaviours in multiple platforms [3].

*ROS* will allow us to build an independent architecture in which every node can use different libraries, configurations. Each node will communicate with the rest using *ROS services* or *ROS topics*. This provides an ideal environment to develop a complex architecture, where all the nodes can stayed connected, receiving all the necessary information, while keeping the independence of tasks. Also, ROS will provide a flexible environment that can be adapted if necessary with just a simple change in configuration.

In the following subsections, a more detailed of *ROS* will be provided in order to get a better overview of the components and functioning.

### 2.2.1 Components

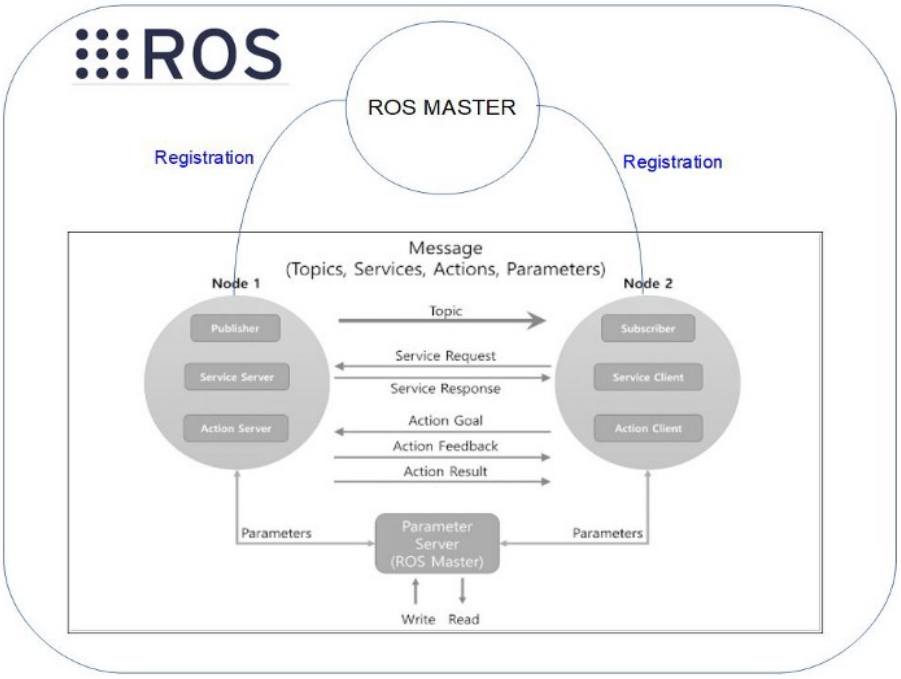



Figure 2.2: ROS Components (Source: [4])

1. **ROS Master**: Name service for ROS. Helps nodes to find the rest. Responsible of name administration and service registration in ROS. Nodes are supervised by the master, it is in charge of communication between the nodes.

2. **ROS Node**: A node is an executable file inside a ROS package. The nodes communicate with other nodes using a ROS client library. There are two types of nodes:
   - Publisher node: Node in charge of writing information in a topic.
   - Subscriber node: Node which receives the messages associate to one or several topics.
3. **ROS Topics**: Once there are defined the publisher and subscriber nodes, both must communicate over a *ROS topic*. Topics are channels where nodes interchange messages.Nodes can publish messages or subscribe to a topic to receive messages.

   Communication between nodes happens by sending ROS messages on topics. Publishers and subscribers must send and receive the exact type of message. Every topic is defined by the type of message published on it.
4. **ROS Services**: Another way that nodes can communicate. The most significant difference with *ROS topics* is that services allows nodes to send requests and receive responses. Services are useful when it is necessary to receive confirmations. They are used for specific tasks and the service-server is in charge of answer to demands.

### 2.2.2 Features

Some of the most important features which make *ROS* ideal for robotics are the following [5]:

- **Communication infrastructure**: ROS offers a middleware with communication tools. ROS is something similar as a "low-level framework" using an existing operating system. **Ubuntu** is the main supported operating system. Before using ROS, a *catkin-workspace* [6] must created, and then ROS can be used as desired.
- **Distributed System**: Ability to separate the code into *packages* containing executable code (*nodes*). Applications can be break into several packages that will contain independent nodes and will interact using the communication middleware.
- **Robot Description Language**: ROS is language *agnostic*. This means that programs can be written in any language. Different languages can be mixed in different nodes or packages. ROS uses TCP/IP to perform communications between nodes.

### 2.2.3 Integration

*ROS* provides integration with other popular open source tools. As a robotic oriented project, it is prepared to work in addition other libraries or programs. With ROS the message passing can be easily done, from sensors or log files. One of this tools that will be used in our project is **MoveIt**

**MoveIt**: is a motion planning library [7]. The movement implementations are well-tested and can be used on a variety of robots. ROS is prepared to integrate with MoveIt, that means, it can be used with any ROS-supported robot [8]

## 2.3 Python

The election of a programming language is the most critical in software development. The rest of libraries and tools used in the project will be influenced by the language chosen. Before making a decision, it is important to answer several questions:

1. Is the coding language compatible with ROS?
2. Are there enough tools and libraries for the requirements of the project?
3. Is the language very difficult to understand or implement?

Following the questions, first of all, ROS supports *C++*, *Python* and *Lisp*.*C++* is a general-purpose language based in *C* language. *Python* on the other hand is a general-purpose language also but it's high-level language, the goal is to be easy and understandable. Nowadays *Python* is the leading language for treating data and machine learning while *C++* is a better option for larger developments. Even there are still languages based on Lisp is very restricted, so immediately is discarded.

It seems that the question might be between *C++* and *Python* . As for the *Computer Vision* and *Machine Learning* tools, *Python* has no rival nowadays. Even it is possible to use *C++* in *Artificial Intelligence* it is not necessary to have such a powerful language. Most of the libraries are highly optimized and they are built in languages such as *C++* or *C*. Python is ideal for the description of the *Machine Learning* environment.

It is possible and very common to have hybrid systems where the high intensity tasks are done in C++ while Python is used for higher level functions. With all the information in mind, the language chosen is *Python* which will allow an agile and easier development, also due to the wide community behind.

## 2.4 OpenCV

*OpenCV* or *Open Source Computer Vision Library* is an open source library for computer vision and machine learning. OpenCV was created to provide a common infrastructure for computer vision applications so they could translate into commercial products easily. This library contains more than 2500 algorithms which include both classic and state-of-the-art computer vision and also, machine learning algorithms. [9]. OpenCV can be used in multiple scenarios such as face detection, camera calibrations, identifying objects...

It is written in *C++*, *Python* , *Java* and *Matlab* and it is supported in most operating systems. This characteristics make the library widely used and very appropriate to interact in environments with different architectures.

*OpenCV* will help with the processing of the images and will help extracting the features and the information of the environment.

## 2.5 PyTorch

**PyTorch** is an open source machine learning framework which goal is to accelerate the process from research and prototyping to production and deployment.[10]. Pytorch provides a package for Python for high-level tensor computation and specially optimised with scientific tools like Numpy. [11]*Pytorch* is a library optimized for *Deep Learning* and neural netoworks using CPUs and GPUs.

Some of the key advantages of *Pytorch* are: [10]

- Production Ready and higher productivity. Simple interface but with a powerful API.
- Distributed Training and performance optimization is possible and enabled with the distributed backend.
- Simplicity, easy to learn. It uses a syntax similar to Python.
- Great API, rich ecosystem of libraries and tools for *NLP* , *Computer Vision* , audio and more.
- Cloud Support on major platforms, easy for scaling developments.

Pytorch works with tensors. **Tensors** are multi-dimensional arrays. This tensors can run on CPU or GPU. A tensor does not know anything about *Deep Learning* or gradients, it is just an array used for numeric computations. Using Pytorch the data needs to be presented as tensors, this is how graph computation and Pytorch works.

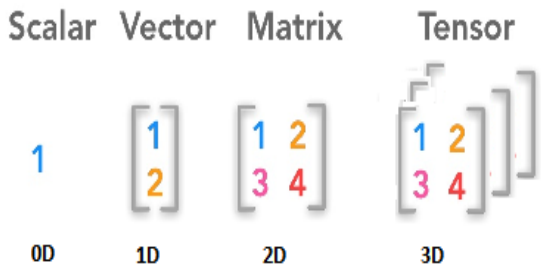



Figure 2.3: Tensor

### 2.5.1 Pytorch vs. Tensorflow

Why *Pytorch* and no *Tensorflow*? [12]. Booth of Pytorch and Tensorflow are the most used open-source tools in academic research and in production. These Python libraries use graphs to compute numerical data. [12]

- Tensorflow was developed by Google and has a very tight connection with the Google Cloud Platform. Keras is a framework based on Tensorflow that makes easier the deployment and configuration of models. One of the biggest advantage of Tensorflow is the large set of tools of libraries for *Machine Learning* .
- Pytorch was developed by Facebook a year later, and the goal was similar to Tensorflow but simplifying the creation of models. Pytorch has been used mostly in academic research. Pytorch is basen on **Toch**, a framework to fast computation in C. One of the best features of Pytorch is the integration and the low level, optimized for Python. For Python programmers Pytorch seems more natural. Pytorch can be higly customizable and has a better compatibility with the multi-dimsensional library *NUmpy*. For Cloud connection, Pytorch can be implemented with Amazon Web Services.

In figure 2.4 there is a comparison between the most used frameworks in the industry. While Skicit-learn library is the most widely used, it is follow by Tensorflow and Keras with Pytorch with a 30% of implementation in business. According to Kaggle, Pytorch has increased its presence in business a 4% in the last year [13]

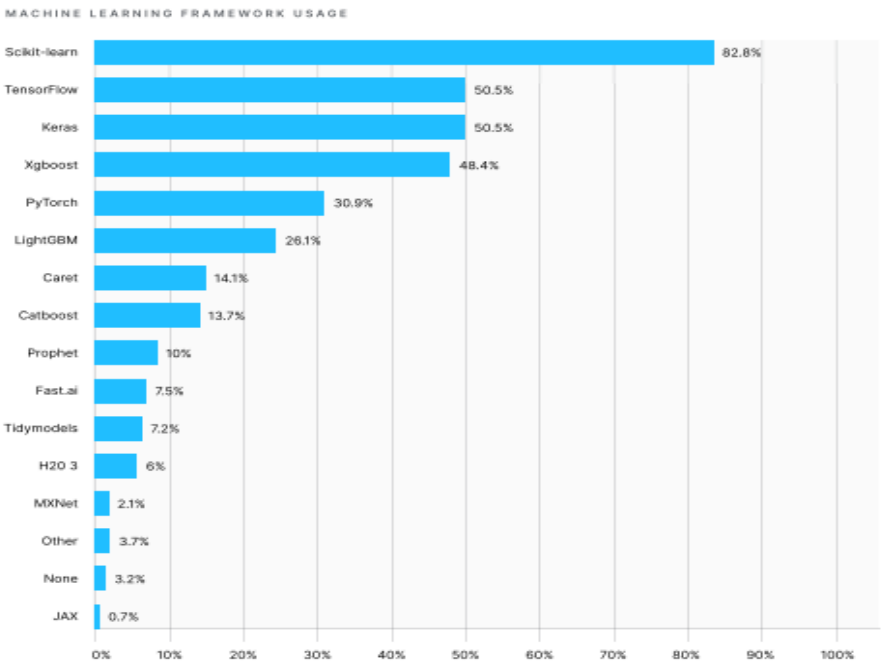



Figure 2.4: Use of *Machine Learning* frameworks in the industry [13]

The choice of a library depends on the data, the model chosen and the goal of the project. Our project is not developed on cloud and there are no more coding languages used. Besides our project is mainly academic so Pytorch is the tool chosen and the easiest to introduce in our code.

### 2.5.2 Pytorch Lightning

*Pytorch Lightning* is a Pytorch extension for Pytorch models. Provides an easier, simpler and more intuitive structure to organize the each of the components in a model. *Pytorch Lightning* is a framework which allows to organize *Pytorch* code in the following way: [14]

1. Provides an easier structure for the models
2. Keeps all the flexibility of *Pytorch*. It is just Pytorch.
3. Reproducibility becomes easier, the structure is always the same
4. Removes redundancies and boilerplate code
5. More readable and understandable

In figure 2.5 the code written in *Pytorch Lightning* is more straight-forward and easy to understand. The structure and the code is cleaner and avoids common coding mistakes. Under the structure, it keeps the same code as in *Pytorch* so, it allows a better reproducibility for production environments.

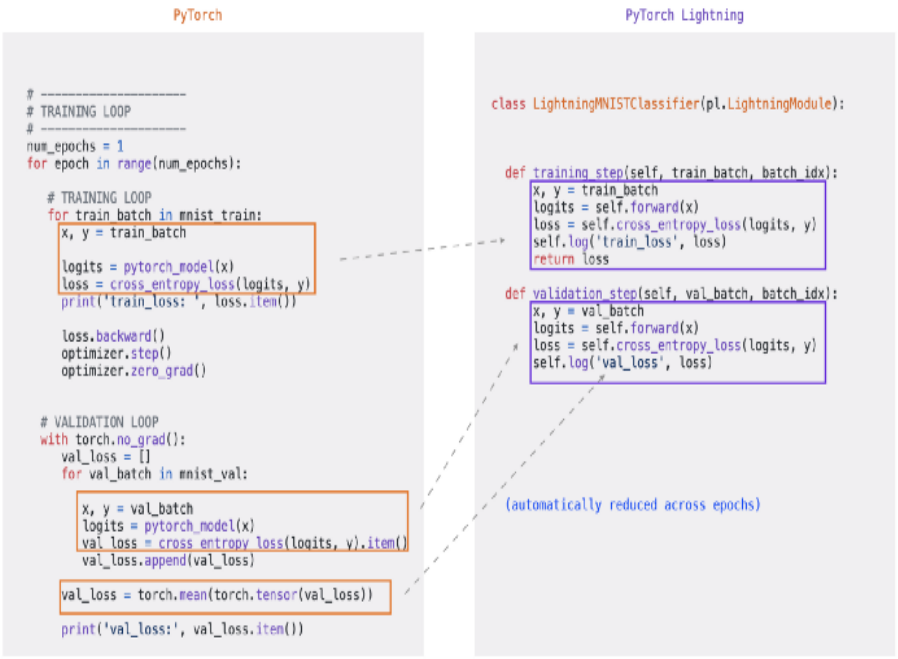



Figure 2.5: Pytorch vs Pytorch Lightning

Metrics can be very easily added to the structure using *Lightning* functions. Also, the personalization and configuration of the models and our won methods can be easily made. As a conclusion, Lightning keeps all the advantages of Pytorch while keeping the code clear, simple and flexible.

### 2.5.3 Tensorboard

*Tensorboard* is the visualization toolkit from TensorFlow. However, it the default board in Pytorch also. Tensorboard provides the tools to visualize what is happening during the process of Automatic Learning in the project [15]. Some of the possibilities of this tool are the following:

- Follow different metrics of the model
- Visualize the model graph and every layer

- Show images, text and audio
- Show histograms, skews and other tensors over time.

The idea behind the use of a tool like *Tensoboard* is to been able to follow exactly what is happening in the model and also it allows the comparison between models. This feature is key when the model is being tuned. It is possible to follow the *hyperparameters* of the model and observe the performance against other types of models, or just between different versions of the same model.

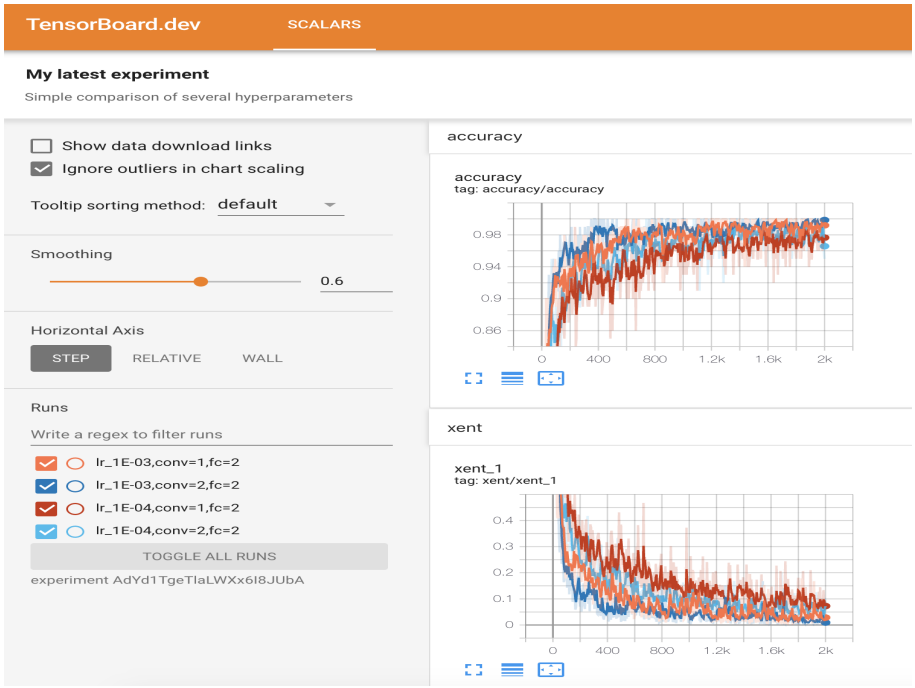



Figure 2.6: Example of Tensorboard

Besides, *Tensorboard* saves the results of every experiment, to make it possible to follow, replicate and share them easily. In addition, Tensorboard is the visualization board more used in business as it is shown in figure 2.7. It is better to implement a tool more popular with a higher support community.

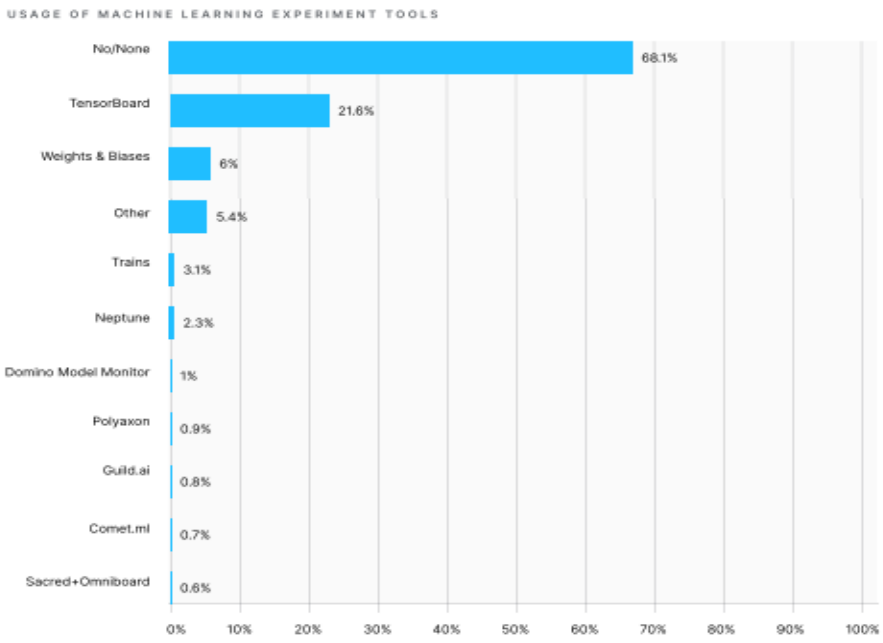



Figure 2.7: Tensorboard in the industry [13]

## 2.6 Libraries and tools

Besides from the libraries previously mentioned. There are some extra tools that will be extra helpful for the development of the project. The idea is to use the most popular open source libraries. They have been tested and approved by the *Machine Learning* community and the support is easier.

As the language chosen is Python (see Sección 2.3), there are

- **Numpy**: One of the most used scientific packages in *Python* . *Numpy* provides multidimensional arrays and methods to operate and manipulate this objects in the most efficient and quick procedure. [16]
- **Scikit-Learn** set of libraries focus on *Machine Learning* . Set of tools for predictive data analysis. Includes a wide variety of implemented models, evaluations and transformations. [17]
- **Pillow** Library for image processing. This tool is designed for a fast access to data store in pixel format. It provides methods for general processing. [18]

- **Matplotlib**: plotting package that provides 2D image visualization with Python.

# Chapter 3

# State of the art

In this chapter, the state of the art for robotics and the technologies for improving performance such as *Artificial Intelligence* and *Machine Learning* technologies will be exposed. Every day new solutions, methods, tools or algorithms are developed based on research and experiments.The last decade had brought multiple breakthroughs. New technologies, applications, algorithms have been discovered, and this is only the beginning. This development have been possible due to two factors:

1. Algorithms require large amounts of data. The increase in databases and data made it possible to feed the algorithms with the necessary information.

2. The development of processors and the increase of computing capacity, GPUs or TPUs, allows *Machine Learning* to train more complex algorithms.

## 3.1 Robotics

Pick and place robots like Universal Robots UR series 2.1 are very common in manufacturing environments. Normally, this robots are used for repetitive, simple and monotonous tasks. This type of robots have several benefits due to the speed and consistency. This type of robots are mostly used for tasks like assembly, packaging, inspection or bin-picking. This last task is the one that this project explores. Bin-picking tasks use vision systems and they can grab objects out of a bin, sometimes even when the objects are randomly mixed. Once the object is grabbed, they can place it anywhere desired. [19]

This knowledge of the environment, so the robot can adapt to the randomly disposed of objects, is due to the advanced technologies like *Artificial Intelligence* . There are several examples to show the possibilities in bin-picking problems. For example, robots can use depth sensors and techniques of *Reinforcement Learning*

and *Deep Learning* to grasp different types of bottles.[20] or using *Computer Vision* for treating food that can be deformable. [21]

In the following sections there are extensive descriptions of the current state for all the techniques and methods available to improve the performance of the robotic pick and place tasks.

## 3.2 Artificial Intelligence

Artificial Intelligence can be defined as a "program in which an arbitrary world will cope not worse than a human". [22] This means that the goal is to imitate the human performance, at least. Based on that idea, industrial machines can be trained to get information and use it to make tasks similar as a person should do it, but automatically and saving time. This is where the idea of this research comes from. The Bin-Picking project is a very common problem in the robotics world. A robot should be able to pick an object and place it where desired. Ideally, the robot should learn about the environment, to avoid random movements that could take a lot of time and resources. So, for this problem, Artificial Intelligence is a perfect tool. The robot should learn in the training process the best ways to solve and pick the objects.

Every year Gartner analyses the technology ecosystem and prepares the Hype Cycle for Emerging Technologies [23]. In the figure 3.1 it is possible to see the important role that Artificial Intelligence is playing in the innovation and technology ecosystem. AI is gaining recognition and popularity and becoming a more and more useful tool in businesses and research.

*Artificial Intelligence* software is being used progressively in businesses and in our daily lives, from the assistants in our smartphones to the system recommendation from our favourite content platform, more and more services are heavily dependant to *Artificial Intelligence* . These new techniques allows machines and products to be more reliable and autonomous. For example: researches show that *Artificial Intelligence* can already outperform radiologist at spotting lung cancer [24] or breast cancer [25]. However, implementation goes slowly due to the expensive costs and the large amount of data required to feed the models.

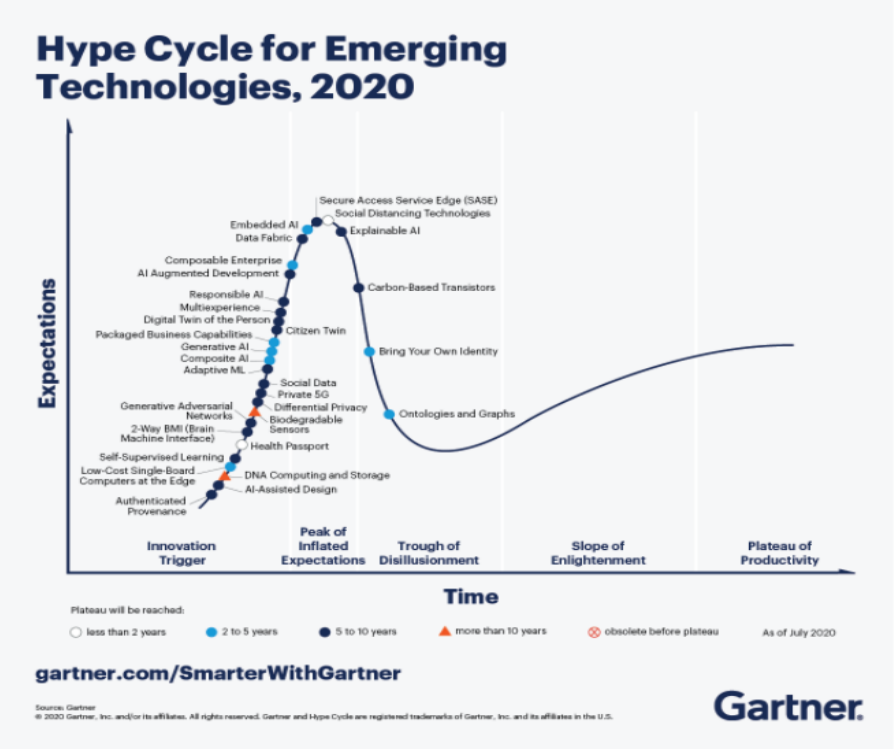



Figure 3.1: Hype Cycle for Emerging Technologies, 2020 by Gartner

## 3.3 Computer Vision

*Computer Vision* can be defined as the set of tools and techniques to understand digital images and videos. Computer Vision aims to help machines process and use this digital content. Most of the times, this type of problems requires to extract information such as the main important features. Features are a representation of the key characteristics from an image.

The first experiments using *Computer Vision* were around 1950 with the first neural networks to detect the contours of squares and circles. In the 90s there were an increased in image analysis and and increase of larger datasets available. Nowadays, there are the perfect conditions to exploit the *Computer Vision* possibilities and some of the systems implemented can be more accurate than humans at detecting visual inputs. [26] The use of *Deep Learning* has brought a better performance to *Computer Vision* problems than traditional approaches.

*Computer Vision* is used in multiple industries, helping automatizing tasks and facilitating processes. For example, in manufacturing industries, most robotic machines need to see the environment to perform the tasks. *Computer Vision* in

robotics can be used to detect irregularities or imperfections.

Here, there is a list of the most used and implemented types of *Computer Vision* used across industries:

1. Image segmentation: partitioning an image into pieces.
2. Object detection: identify different objects in an image
3. Facial recognition: recognize human faces and specific individuals.
4. Edge detection: detect the contours of different objects.
5. Pattern detection: detect repeated shapes, colours or indicators
6. Image classification: label images into categories
7. Feature matching: matches similarities between images.

   In order to achieved this results, *Computer Vision* , as a brach of *Artificial Intelligence* , uses techniques such as *Machine Learning* and *Deep Learning* for resolving complex problems. This two technologies will be described in the following sections.

## 3.4 Machine Learning

*Machine Learning* is a subset of *Artificial Intelligence* . Machine Learning can be defined as the improvement of performance through training and experience on concrete tasks. The primary aim of *Machine Learning* is to allow machines learn automagically.

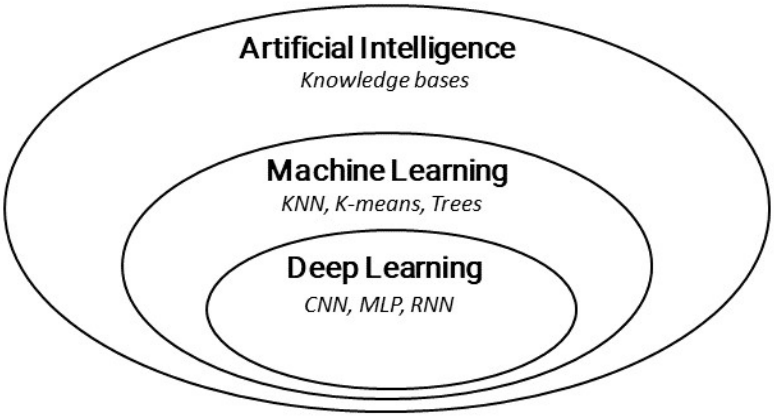



Figure 3.2: Artificial Intelligence relationships

*Machine Learning* models can be categorized in the following way:

- **Supervised models**: Used for labelled or classified data. The system is able to predict the outputs of any new target after enough training.
- **Unsupervised learning**: Used when the information is not labelled. These models explore the data and points out relations and structures.
- **Reinforcement learning**: Learning method that interacts with the environment. This method will be described in deep in 3.6

Kaggle, released its report "State of Machine Learning and Data Science 2020" [13] where it is possible to see the *Machine Learning* adoption in business where the Kaggle scientists work. In figure 3.3 the increase in adoption of *Machine Learning* in business is not very significant in comparison with the two previous years.

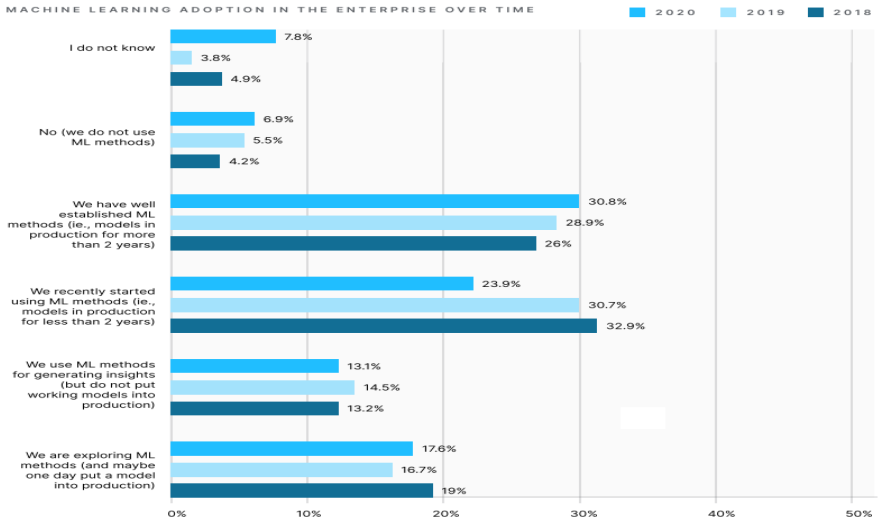



Figure 3.3: Machine Learning adoption in enterprises

Also is curious to analyse that the most used algorithms, are the simpler ones such as Linear or Logistic Regression or Decision Trees in figure 3.4

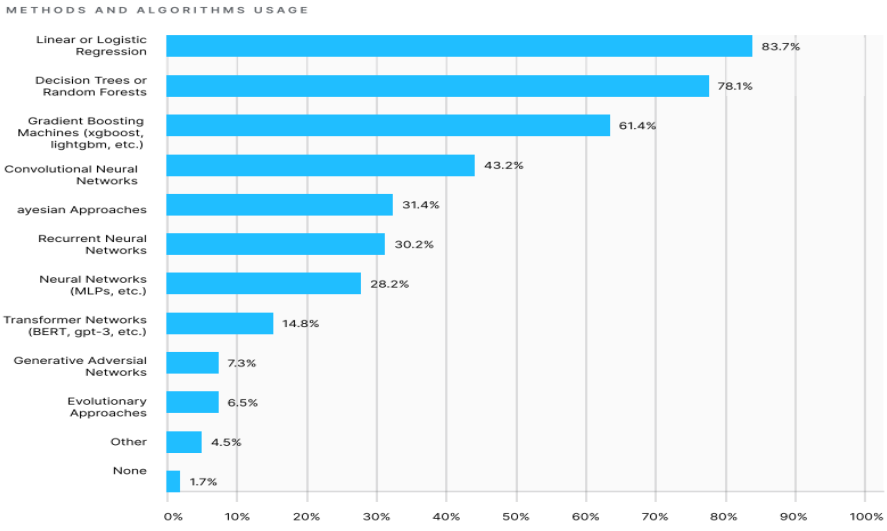



Figure 3.4: Most common methods and algorithms

However, it is also possible to see that *Deep Learning* algorithms are increasing its popularity among the industry. And what exactly is *Deep Learning* ?

## 3.5 Deep Learning

Sometimes, traditional Machine Learning is not enough when dealing with complex and challenging datasets. In that occasions, is when *Deep Learning* might become useful. *Deep Learning* can be defined as an approach to *Machine Learning* but using deep computations. This strong computation ability will allow to detect complex and hidden patterns in the data, that *Machine Learning* standard algorithms, may not detect.

The field of *Deep Learning* is one of the most researched fields in *Artificial Intelligence* . Some of the most impressive innovations in this field have been developed in these recent years. *NLP*, (also know as *Natural Language Processing*), *Image Recognition* or even *Gaming* are examples in where *Deep Learning* have simulated or even improved human-level results. [27]

**Neural Networks** are becoming the standard of *Deep Learning* . This networks are composed by units call *neurons*. A neuron is a computation unit and it is from the union of neurons where the computation advantage of *Deep Learning* come from.

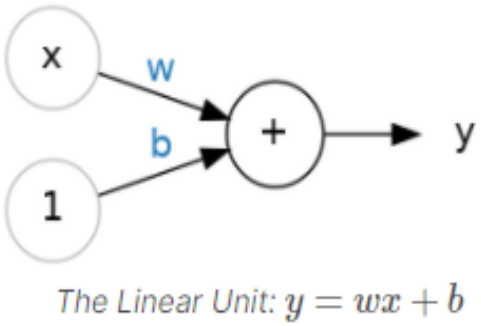



Figure 3.5: An example of Neuron

In figure 3.5 there is an example of a simple neuron. In each neuron, there is an input $x$. The connection to the neuron have an associated weight $w$. When the input flows through the neuron, the weight is multiply by the weight of the neuron. This $w$ will vary as the *Neural Network* learning increases. In addition, there is also this $b$ value called *bias* which its function is to enable the neuron to change its output, with no dependency of the inputs. Finally, $y$ is the output of the *activation function* or the formula $y = w * x + b$

*Neural Network* normally are formed by several layers. Depending on the type of connection between the layers and the neurons, there will be different types of networks. Inputs will be transformed through each of the layers. Besides the

layers it will be necessary to define several parameters that will specify how the network should behave.

1. **Activation function**: Non-linear functions that help the model to learn more complex relationship besides the linear relationships.

2. **Loss function**: What problem must be solved.This function measures the difference between the target true value and the value predicted in the model. Depending on the type of problem, there will be a need to use different loss functions. There is a difference between regression, where the output is a numerical value, or classification problem that predicts an input belonging to a class.

3. **Optimizer**: How to solve the problem. The optimizer adjusts the weights of the network to minimize the loss. All the optimizers in *Deep Learning* belong to the *Stochastic gradient descent* family where every step of the training the weights are adjusted based on the measures of the loss and the direction that will make the loss smaller.

4. **Learning Rate**: Parameter of model that determines the step size at each iteration in training, while moving to a minimum loss function. In other words, the learning rate controls how much the model can change in response to the estimated error each time the weights of the model are updated.

5. **Batch size**: Number of data samples that are fed in the model before updating the weights. At the end of every batch, the model compares the prediction to the real target values and the error is estimated. From this error, the weights of the model are updated with the objective of improving the algorithm.

6. **Epoch**: Number of times that the algorithm will go trough the entire dataset during the training step. An epoch is comprised of n batches.

This configurations would vary depending on the problem to solve. Regression and classification models, need different configurations. There are different types of neural networks like the following:

- **Artificial Neural Network (ANN)**: Group of multiple neurons at each layer. Feed-forward networks.

- **Multi-Layer Perceptron (MLP)**: Neural Network with three or more layers.

- **Recurrent Neural Networks (RNN/LSTM)**: RNN are used for sequential data. Consists on setting looping constrictions on the nodes to ensure the information is captured.
- **Convolutional Neural Networks (CNN)**: This networks uses kernels (filters) to extract the features of the input.

In the next section, we wil focus on **CNN** the state-of-art for *Computer Vision* challenges.

### 3.5.1 CNN

*Convolutional Neural Networks* or *CNN*, are consider as the go-to *Deep Learning* architecture for *Computer Vision* problems like object detection, image recognition or even for video analysis. [28] What is a CNN? It is a powerful neural network used to extract features from images using filters. Convolution refers to a mathematical operation applied to matrices.In case of images, the matrix is formed by pixels. This convolutional operation extracts the information relevant of the images, called features.

The most important advantages of *CNN* is the ability to use the spatial features of an image. This will help in order to recognize objects, locations or relations inside of an image.

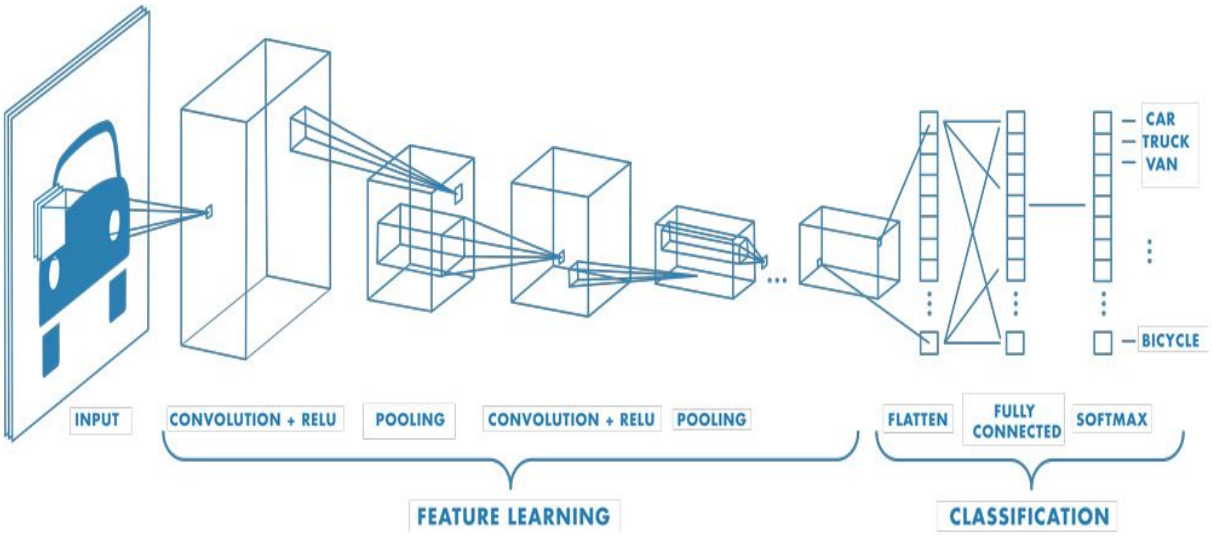



Figure 3.6: CNN architecture

1. Convolutional layers

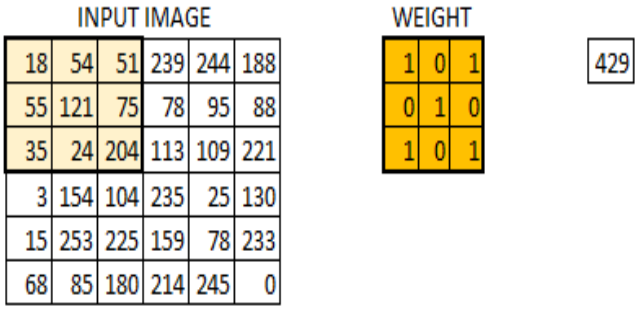



Figure 3.7: Convolutional Layers

2. Pooling layer: Layers used to reduce the number of parameters. Pooling layers are normally added between convolutional layers. This layer reduces the size of the image

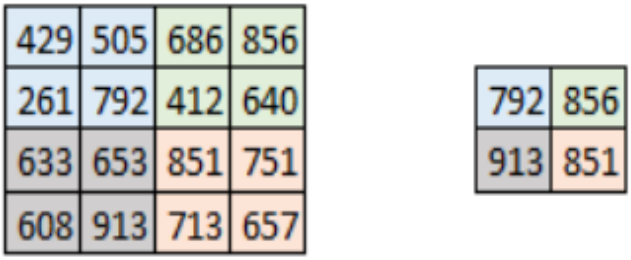



Figure 3.8: Pooling Layers

3. Output Layer: After the different layers, the output of a CNN must be a fully connected layer to generate the number of classes needed.

### 3.5.2 Transfer Learning

Nowadays, most of the CNNs are not trained from scratch. *Deep Learning* in general requires very big datasets with sufficient data. However, in most of the scenarios, increasing a dataset can be complicated and the size may not be enough. This problem is specially common in *Computer Vision* tasks. Instead, the solution is to use a CovNet already pre-trained on a very large dataset (e.g. ImageNet with 1.2 million images and 1000 classes [29]) and then, use this network as desired for the task of interest.[30]

How it is possible to use Transfer Learning:

1. Create a model from scratch. The model is trained and then, the parameters of the model are saved. Then the model can be load and be used in another model.

2. Take an existing trained model and tune the parameters.

There are two types of transfer learning: [31]

1. **Finetuning**: This method uses a pre-trained model, but all the parameters are updated for the new problem. The idea is to retrained the model, but to use the structure already implemented and proved.

2. **Feature Extractor**: The model is pre-trained but in this case only the last layers will be actualized, in order to adapt the more concrete layers to the actual problem.

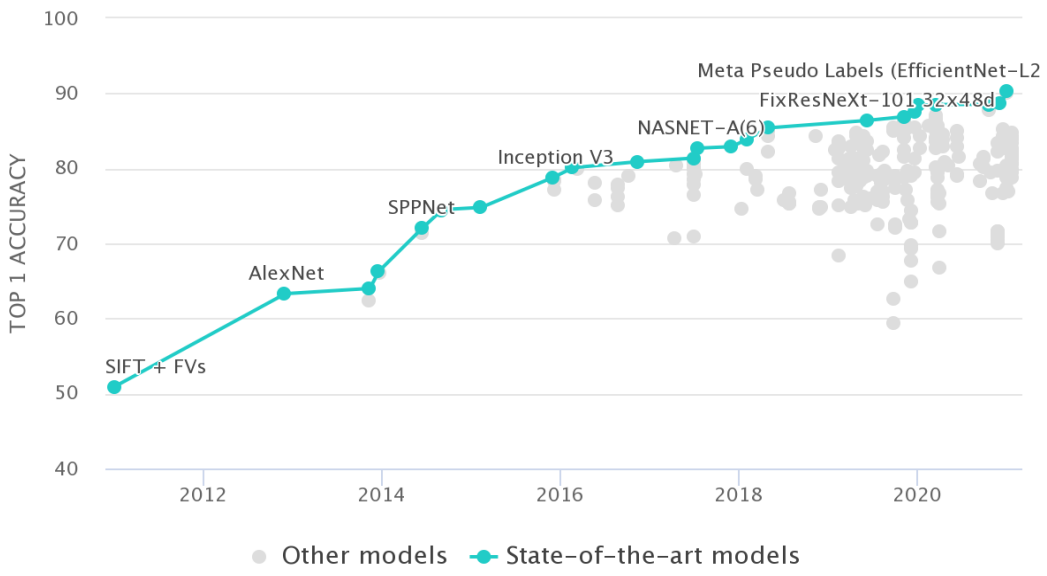



Figure 3.9: Image Classification on ImageNet

**When and how to fine-tune?** This depends basically in the size of the dataset (small or big) and the similarity to the original dataset. There a few guidelines to face the problem correctly: [30]

1. New dataset is small and similar to original dataset. With small data fine-tuning is not a good idea to prevent over-fitting. The best idea is to train a linear classifier.

2. New dataset is large and similar to the original dataset. the best option is to fine-tune, the model will not overfit.

3. New dataset is small but very different from the original dataset. The dataset is small, the best option is to only train the linear classifier. Since the dataset is very different, it is better not to train the classifier from the top of the network.

4. New dataset is large and very different from the original dataset. Train the network from scratch.

| | Similar dataset | Different dataset |
|---|---|---|
| Small dataset | Transfer learning: highest level features + classifier | Transfer learning: lower level features + classifier |
| Large dataset | Fine-tune* | Fine-tune* |

Figure 3.10: How to fine-tune in Transfer Learning?

## 3.6 Reinforcement Learning

*Reinforcement Learning* research has grown very rapidly. There are a wide range of algorithms with the aim to be used in a lot of fields and implementations. The advantage of this type of algorithms is the autonomy and the ability to learn in complex environments. For example, Meta-Learning and Self-learning systems are becoming very popular [32]. *Deep Learning* and *Reinforcement Learning* play and will play a key part of *Artificial Intelligence* developments. To ensure safety in AI, there needs to be strategies and algorithms compatible with this new paradigm.

There is not a standardized *Reinforcement Learning* approach. Most of the information is found in papers or in blogs. Most of the times, implementing RL

algorithms can be very difficult, due to that the information on the papers is not enough, or the code is hard to read. [33]

For researchers already working in the field, there are some repos like garage [34], rllib [35] or Baselines [36] where algorithms are built into frameworks. However, in these repos, some assumptions and trade-off are made, and this information can be really complicated for new practitioners.

The field of *Reinforcement Learning* is very difficult to enter for researchers and for widely implementation in the industry.

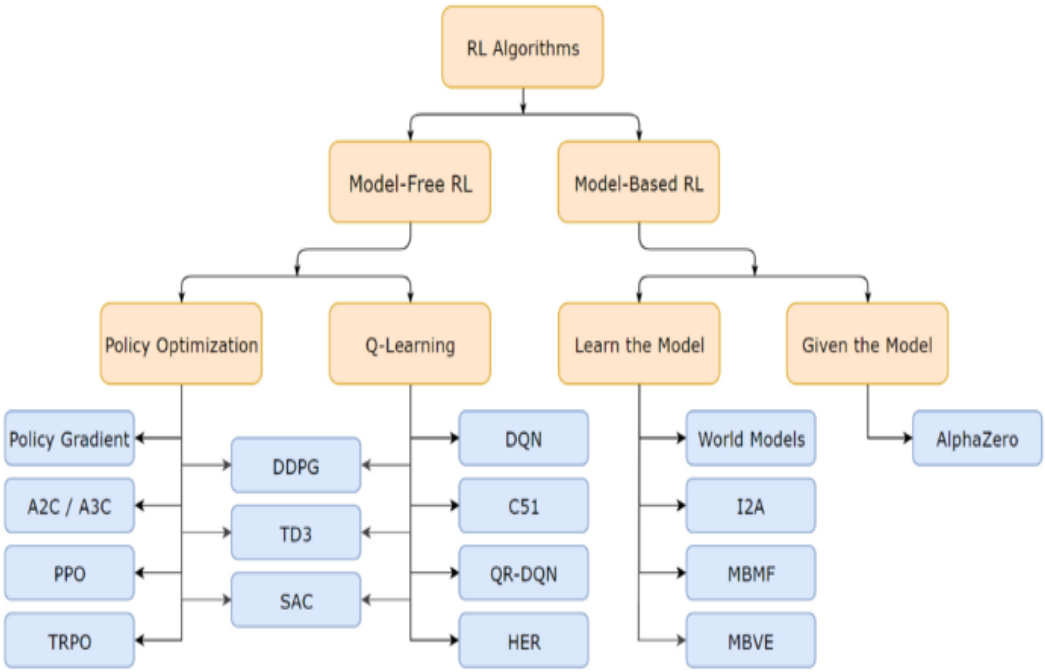



Figure 3.11: Types of *Reinforcement Learning* algorithms [33]

*Reinforcement Learning* is a *Machine Learning* approach where agents are taught to solve a problems or some tasks by trial and error.

*Reinforcement Learning* is an area of *Machine Learning* that focus on the actions an agent can make to maximize the reward received. *Reinforcement Learning* studies the behaviour of subjects in specific environments and learns how to optimize and improve the processes. [37]

Subjects in an environment might take actions. Every task or decision has an associated reward, this reward is a number and can be either positive, negative or just zero. The goal is to maximize this reward.

*Markov Decision Processes* or *MDP* allows to structure sequential decision making. This is the base for *Reinforcement Learning* . The components of a MDP are:

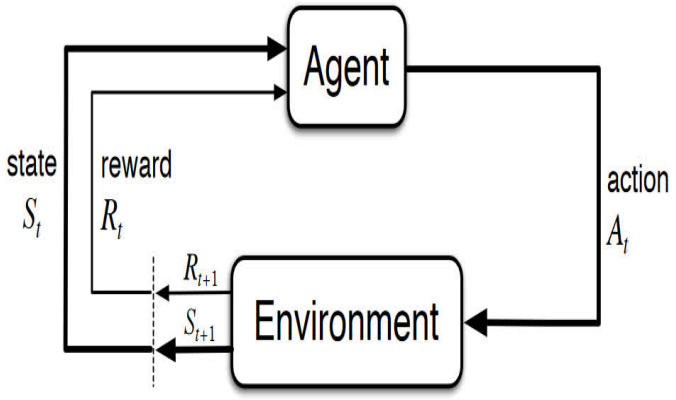



Figure 3.12: Markov Decision Process [38]

- **Agent**:decision maker.
- **Environment**: place where the agent interacts.
- **State**: Current information of the environment
- **Action**: next movement or decision to take
- **Reward**: value associated to a previous action

In MDP there is an agent, interacting with the environment. At each time, the agent will get information about the current state and will select an action. Every transition into a new state is rewarded due to the actions taken. The goal is to maximize the total amount of rewards given over time. [38]

The next step in *Reinforcement Learning* is to find out the probability of the agent to take an action in any state. This is called *policies*. Policies are functions that sees the current state as probabilities depending on the possible actions. Then, it is important to evaluate how good the actions in any state would be. This is called *value function*. These value functions estimate how the agent can perform, depending of the choices. This estimation is always measured with the possible rewards. *Reinforcement Learning* algorithms try to estimate the best routes and decisions for the agent to make. [39]. The goal of *Reinforcement Learning* algorithms is to find the "optimal" policy. The optimal policy has an associated optimal state-value function. Also, there is an optimal action-value function which gives the biggest expected return possibly achieved for each possible combination. Bellman's equation is the responsible of obtaining this expected return. [39]

Depending of the complexity of the environment and the possible choices, traditional *Reinforcement Learning* methods are not enough to solve the problems and improve the performance. In next section, there is a description of the algorithms more suitable for complex problems.

### 3.6.1 DQ-Learning

Q-learning is a *Reinforcement Learning* technique used for learning the optimal policy in a MDP. [40]. The objective of this method is to find the most optimal policy. This means that it looks for the maximum value of the total rewards that is achievable over all the steps. In order to find the optimal policy, Q-learning learns the optimal Q-values for the pair state-action using Bellman's equation and in an iterative process. While exploring the environment can access the learning stored in the Q-table and modify it with the new knowledge. One of the most important concepts in *Reinforcement Learning* is "Exploration versus Exploitation". Exploration refers to the act of the agent exploring the environment in order to find out more information. Exploiting means, the agent uses the information already learned to maximize the return. This trade-off exploration-exploitation needs to be controlled by a parameter the epsilon greedy strategy, so there is an equilibrium.

Q-Learning might be a good job in very small and restricted environments, but when the working space becomes more sophisticated and complex the performance decreases drastically. To avoid this behaviour, there is a solution. Instead of iterating to compute *Q-values* to find the best *optimal Q-function*, there is the possibility of estimating the *Q-function* by using approximations. It is necessary a **Deep Q-Learning algorithm**. [41]

The idea behind DQ-Learning is to use a CNN 3.5.1. The input will be a given states. This states can be represented in images for example, like a stack of frames. The layers of the CNN are the normal layers of this kind of networks. The output layer will be a fully connected layer with the Q-value for each of the actions that can be chosen from the input state.

## 3.7 Feature Engineering

The most important asset in a *Machine Learning* or *Artificial Intelligence* project is the data. It is impossible to build a good model if the data is not correct or enough. However, not only is important to have the appropriate data, but to explode it correctly to extract the most useful information as possible.

In figure 3.13 it is possible to see that the more time consuming activities in data science are related to data manipulation.

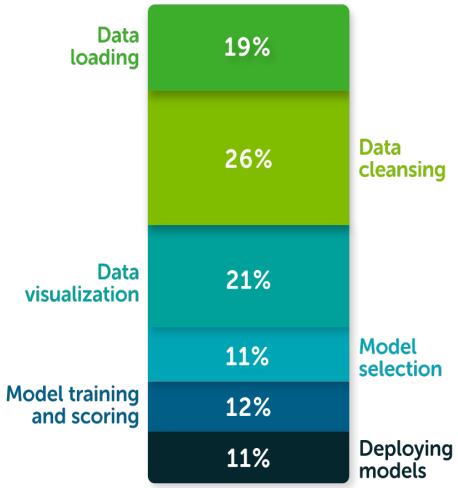



Figure 3.13: Percentage of time for Data Science tasks [42]

The majority of time, data is not prepared to be fed directly into the models. There is a need of treating, cleaning and understanding the information obtained. One of the first things done in a project with data is an exploratory analysis where the distributions of the data, the correlation of variables or any anomaly can be detected.

Nowadays, the data has huge amount of features. Most of the times, working with an exaggerated amount of features can recur into problems and with feature extraction, there are a series of advantages that can help our problem:

- Improvements of accuracy
- Decrease dimensionality of dataset, easy to manipulate.
- Reduction of the risk of over-fitting
- The training of the models is more efficient and fast
- Data visualization can be improved
- Increase in the explainability of the models.

The main goal of *Feature Extraction* is to reduce the number of features of the data, but without losing the important information. The data can be recognised, but it is organized in a more efficient manner, where with less information, data can be represented in the same way. [43]

Traditional *Machine Learning* models, extract patterns from the data analysing the exiting relations. Sometimes, this data must be transformed using the knowledge of experts, or maybe the data is not in the correct format to work with certain algorithms. *Feature engineering* are the set of techniques and knowledge used to extract information from the data. This techniques can be transformations, feature extractions or methods, taking into consideration the type of data and how it should be treated.

# Chapter 4

# System Description

In this chapter, the report describes the idea behind the project and working methodology. First, it is important to understand the motivation for this project. The objectives achieved will be described afterwards and finally, the methodology for the implementation will be presented.

## 4.1 Motivation

The goal for this project is to demonstrate that the use of *Computer Vision* can improve the performance in a *Bin-picking* problem, using *Reinforcement Learning* algorithms.

There are multiple solutions in the market for *Bin-picking* tasks, but in some cases no scientific articles are published. The project is inspired in a commercial solution, where the only information about is a posted *Youtube* video. (See link here [44])

In the video it is possible to see a robotic arm, the robot tries to pick up iron cylinders at random positions. Meanwhile, the robot collects pictures labelled as successful or failed in every attempt.

The collected data is feed into a *Deep Neural Network* to predict the output of the bin-picking from the images as it can be seen in 4.1

The goal of our project is to develop a similar solution adding new techniques and ideas that can be implemented in a real environment. The objective of the project is to implement a Pick and Place task with a robotic arm using *Artificial Intelligence* techniques such as *Computer Vision* and *Reinforcement Learning* .

The difficulty on this task resides in the random distribution of the pieces to pick in a box.The system will need to identify the objects and find an optimal set of movements for completing the task.

For this project, the robot makes the decision using the images of the envi-

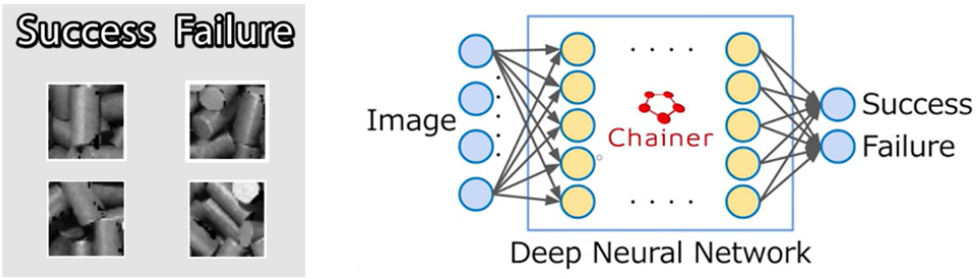



Figure 4.1: Gathered images as input in Deep Learning Model

ronment as inputs and then, the *Reinforcement Learning* algorithm decides its following movement. Image Recognition and Image Processing are the tools selected to improve the knowledge obtained from the algorithms.

As the images taken are key to make the next decision, a good image dataset and a good processing are key to detect the most useful 'features' or characteristics. One of the biggest challenges is to apply all the techniques into a real-world problem. With this project, the results are shown in a real robotic arm, the same one that could be used in industrial factories. The aim of the research is to demonstrate that, not only the models and the information is theoretically correct, and it is possible to see the results, but however that in real-life scenarios the final solution can be implemented, and it is ready to be used. Better knowledge of *Computer Vision* and its application to the industrial world can be achieved with research. Understanding how images work and how they can be exploited it is vital to obtain the best results possible. New techniques and implementations are been develop constantly. It is important to understand how these tools can be used in the development of the problem.

## 4.2 Objectives

The main goal of this research is to improve the performance of a robotic arm. To perform this task, images are going to be the centre of the study. As the project is focused on the images, the inputs for the decision-making algorithm, the proposed objectives are the following:

1. First, is a requirement to have a wide knowledge of all techniques and processes so the best solution can be implemented. As a result, the first part of the project is to study and analyse the upcoming innovations and the current

practices for Image.

2. For the success of the project, we are using Computer Vision and Image Recognition Techniques. The outputs of this models and techniques will be fed into a Reinforcement Learning algorithm.

3. An image pre-processing pipeline will be created. The pipeline will cover all the steps required from obtaining the images, the pre-processing to improve the inputs, obtaining the key features and the best way to exploit the characteristics using Machine Learning and Deep Learning.

4. The solution must be flexible. The objective is to obtain a solution that can be applied to several scenarios, so if the scenario and the circumstances change, the parameters and the possibilities can be adapted.

5. Preparing a benchmark for electing parameters, methods, and techniques. Every decision for processing images must be selected based on evidence and performance. Every step should be understood and must be represented to gain a full knowledge on the feature election by our models.

6. Improving the input images by adding extra information and analysing the effect in the performance and accuracy of our models.

## 4.3 Methodology

The result of the project will be a python coding solution. This code will be implemented as a node which will need to communicate with hardware and software components. (Project architecture 5.2)

For the success of the project, not only the *Computer Vision* solution must be implemented and working, but it must work in the complete architecture. As the processed images will be the input of the *Reinforcement Learning* algorithm, it is important to work closely with the other parts, in order to ensure the compatibility and the necessities of all nodes involved in the project.

The development of the project must be agile and quick. There are many possible configurations, parameters or methods to obtain results. In addition, it will be necessary to adapt the code to the necessities of the environment and the restrictions imposed by the hardware.

With all in mind, the project will be developed in following the next rules:

1. **Experimental**: As the project is inspired in some solutions in the market but with no scientific articles of technical description behind. The project is developed based on research and experimentation. The final solution will

represent a combination of techniques and test, learned and improved along the development of the project.

2. **Incremental**: The idea of an incremental development is to start from the easiest solution to more complex and advances implementations. The reason is to justify the improvements and to understand if more complex techniques, ensures better results.

3. **Iterative**: The project is developed using *Artificial Intelligence* , *Computer Vision* and *Reinforcement Learning* techniques. One of the main difficulties of the *Machine Learning* implementations is the tuning of the models and the wide range of possibilities and configurations. With an iterative approach, it is possible to adapt the algorithms and the methods, based on results and in the requirements of the project.

4. **Independent**: As the project is composed by several nodes, it is important to maintain each node independently so any change or modification does not affect the whole project.

The project will followed an agile methodology. The main goal is to being able to adapt the code and the solution based on the necessities of the problem, and also being able to modify anything without breaking or stopping the whole project.

## 4.4 Planning

As explained in the previous section, this project will be developed in the most agile way possible. This means that some tasks will be developed in parallel. The project will be structured in the following steps:

1. **Hardware**: In this section, all the activities planned are related to the Hardware development and design. These tasks are the most problematic. The design and assemble of the pieces are not a priority in this project and they are provided by a third party. This means, that any delay in the implementation of the final structure, affects directly to the training of the model.

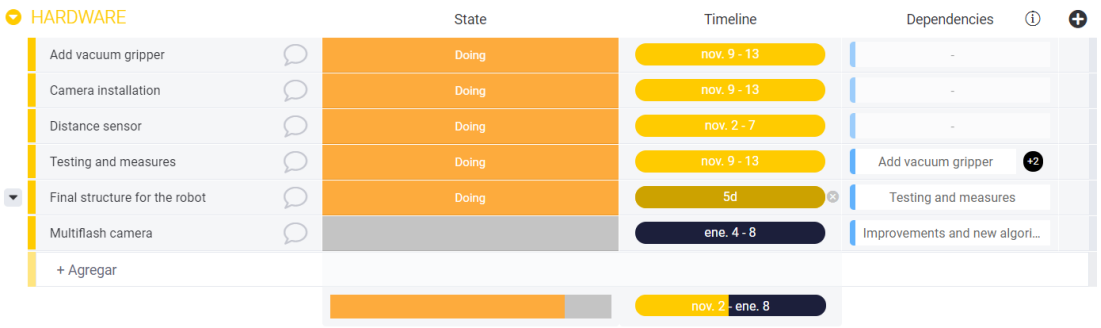



Figure 4.2: Hardware activities

2. **AI Manager**: In this section, the tasks scheduled are related with the programming segments and with the study of technologies, in order to apply them correctly to the project. The first steps consists mainly on research while in the last days of the project, the tasks are focused on testing and iterating. This step is the most complicated. These activities require a lot of knowledge and due to iteration and testing of new algorithms, they need a lot of time.

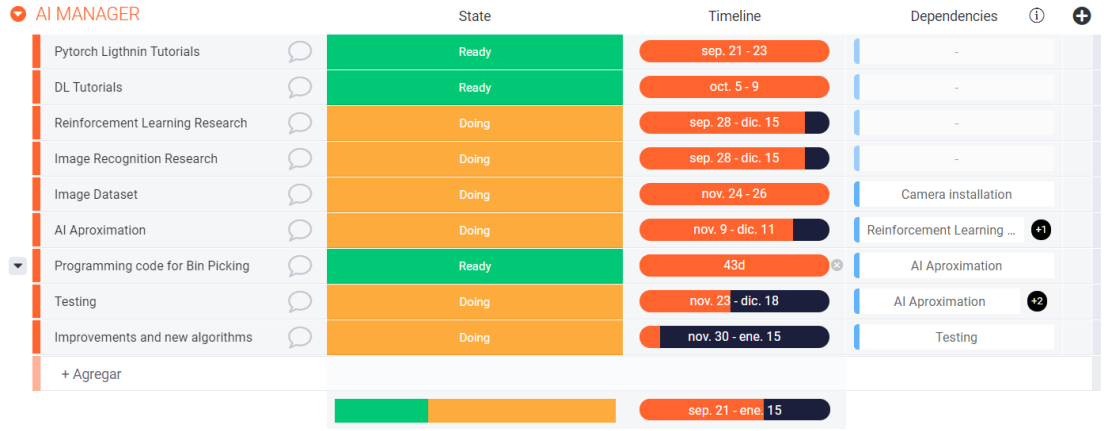



Figure 4.3: AI Manager activities

3. **Robot Controller**: Robot Controller is the easiest step in the project. The main difficulty is learning how to interact with the robot and how to communicate between the nodes. Once the main issues are solved, the code only needs some extra modifications and can be easily reusable.

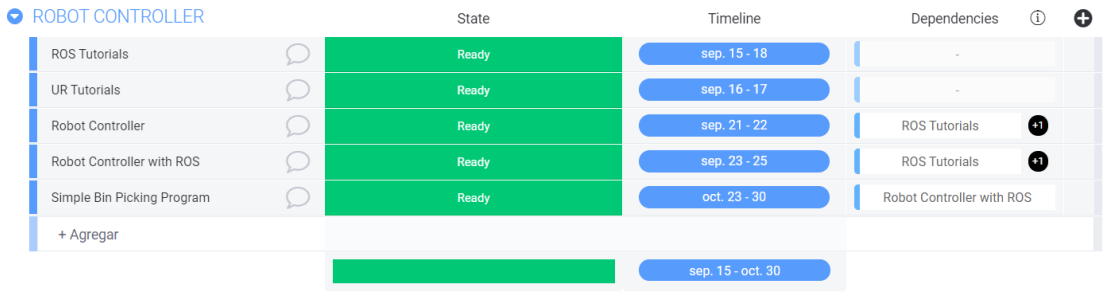



Figure 4.4: Robot Controller activities

## 4.5 Resources

The project was a research in applied *Artificial Intelligence* to a known robotic task. There is no initial budget for the development, and the spendings are relatively small for a complex problem .

In this section it is possible to find all the resources used in the development of the project.

| Item | Quantity | Price |
|---|---|---|
| Nvidia Geoforce GTX | 2 | 395.88 |
| Vacuum Gripper | 1 | 53.50 |
| Raspberry Pi4 | 1 | 59.05 |
| Electronic Parts | (Sensors, transistors..) | 30 |
| Arduino | 1 | 22.64 |
| HDMI -> VGA | 2 | 63.31 |
| QWERTY keyboards | 2 | 36.85 |

Table 4.1: Budget

# Chapter 5

# Solution

In this chapter, a detailed description of the solution implemented is provided. First, there is the project description at a high-level. The following is a more detailed view of each of the components in the project. The objective of this chapter is to understand the complete architecture and the function of each node.

## 5.1 Project Overview

In the motivation section 4.1 there is a detailed description of the main objectives and the reason behind the project. In summary, the main goal of the project is to implement a *Bin-picking* solution, using *Artificial Intelligence* techniques like *Computer Vision* and *Reinforcement Learning* .

Figure 5.1: Real view of the project environment

As it is possible to observe in figure 5.1 there is a complete view of the elements of the project. The robot and most of the components are located in a laboratory

where there is access to computers. Only the server with the *Artificial Intelligence* algorithms is in other laboratory.

However, in the picture it is difficult to identify all the components of the project. In figure 5.2, there is a diagram of all the complete architecture. All the components of the solution are focused on the environment and have been designed to answer to the needs of the environment.

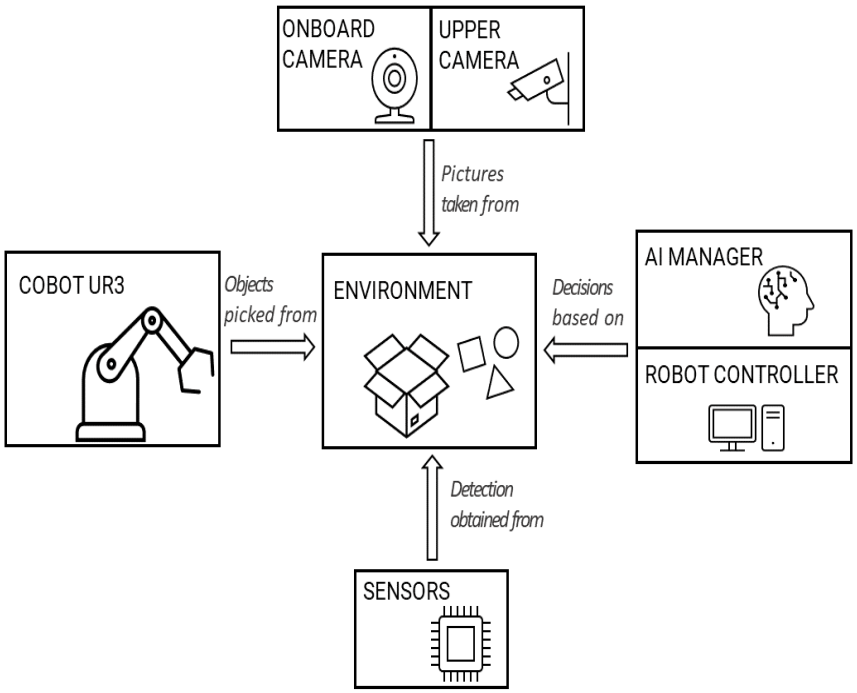



Figure 5.2: Complete architecture

- **Environment**: box containing the pieces the robot should pick.
- **Cobot UR3**: The robotic arm responsible for picking the pieces with a vacuum griper.
- **Onboard camera**: Camera in charge of taking the images in every pick movement. The images will be stored with the label *success* or *fail*
- **Upper camera**: Camera with a complete perspective of the environment.
- **Sensors**: Sensors will obtained useful and necessary information from the environment.

- **AI Manager**: *Artificial Intelligence* algorithms and the logic behind the decision process.
- **Robot Controller**: communications with the robot

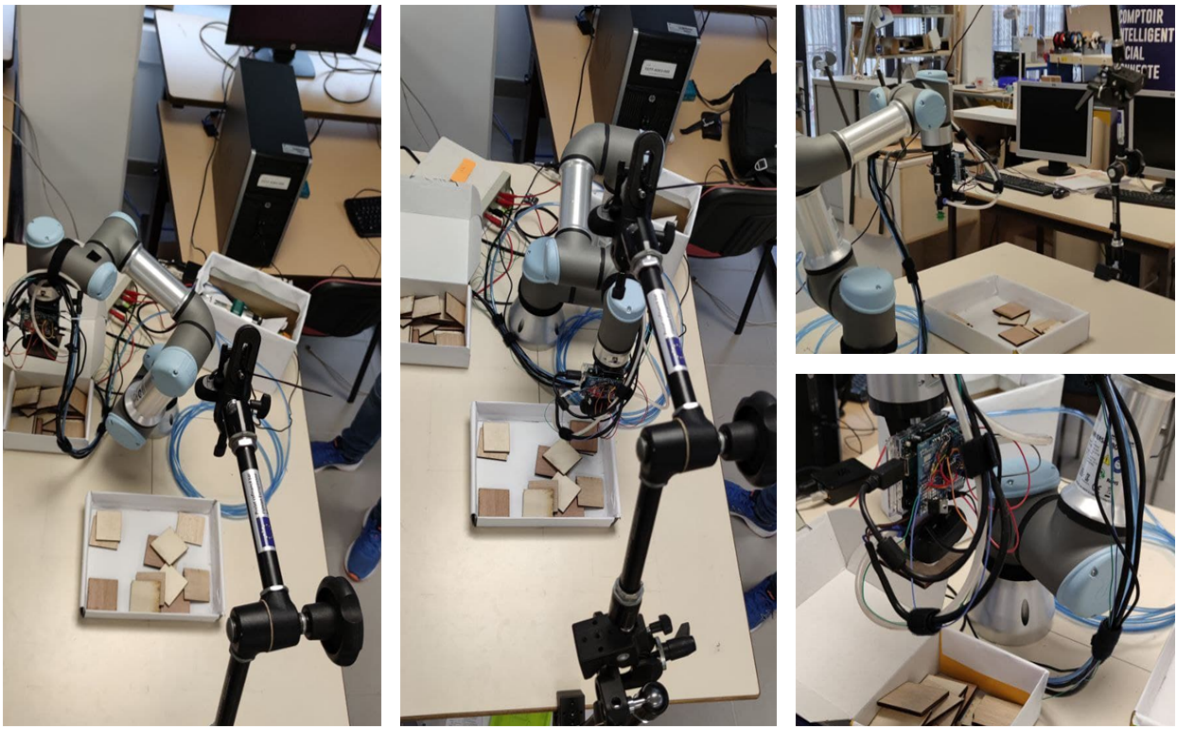


Figure 5.3: Environment from different perspectives

## 5.2 System Architecture

As explained in the description of *ROS* (see more in section 2.2) one of the most noticeable advantages is the flexibility that provides to projects involving robotics. In this project the use of *ROS* allows to have a complex architecture where the nodes can be distributed in different computers and can be composed of different electronic parts. The only restriction is that nodes need to be in the same network in order communicate with each other . These communications are possible using *ROS services* and *ROS topics*.

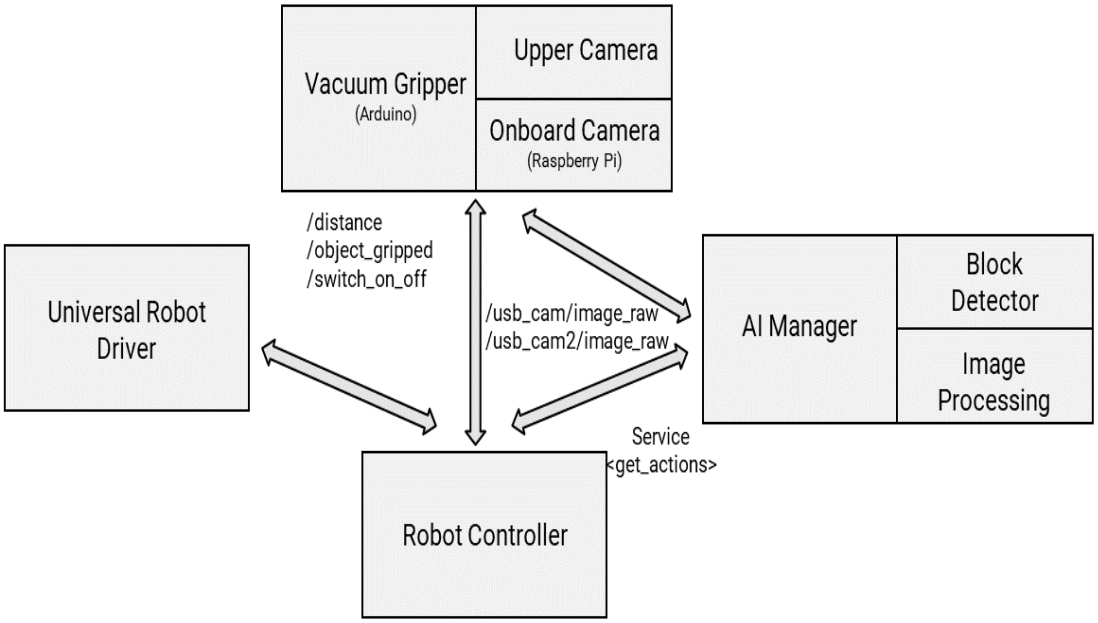



Figure 5.4: ROS architecture

As it is possible to observe in figure 5.4 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. Then *ROS services* are used. In order to understand clearly the structure and the communications, it is interesting to analyse the relation between the nodes.

1. Universal Driver and Robot Controller: The driver provides a stable and sustainable interface between UR robots and ROS. Robot Controller will communicate with the driver to send the actions that the robot must follow.

2. Robot Controller and Vacuum Gripper:

   - The gripper only needs to be switched on while it is in a down position, and while carrying the successful pick to the place position. Robot

Controller must send a message to the topic 'switch_on_off' to control the power on or off in the pump.

```
# Gripper topic
self.gripper_topic = 'switch_on_off'
# Publisher for the gripper topic
self.gripper_publisher = rospy.Publisher(self.
    gripper_topic, Bool)
```

Snippet 5.1: Gripper communication: switch on/off topic

- The down movement of the robot must be done carefully, the speed must be taken into account, if not, the robot will crash with the pieces or with the box. The 'distance' topic measures if the robot is touching a piece or the box, every few cm the robot will stop to measure. If the gripper is touching any surface, the gripper will send a True message to the topic, so the robot will subscribe from this topic, will obtain the information and will stop the movement.

```
# We retrieve sensor distance
distance_ok = rospy.wait_for_message('distance', Bool).
    data
communication_problem = False

```

Snippet 5.2: Gripper communication: distance topic

- The 'object_gripped' topic is very important. Robot Controller is subscribed to the topic, and the gripper will write a message True or False if the object was picked. If the object was picked, we will take the piece to the place position if not, the gripper will be turned off.

```
object_gripped = rospy.wait_for_message('object_gripped',
     Bool).data
if object_gripped:
# If gripped object is True, place into the desired point
  self.take_place()
else:
  # We turn off the gripper
  self.send_gripper_message(False)

```

Snippet 5.3: Gripper communication: object gripped topic

3. Robot Controller and Upper Camera: The Upper Camera will take the pictures every time the robot makes a successful pick, while the robot is in the "place position" dropping the piece. The image will be written in the topic = 'usb_cam2/image_raw'

```
self.image_controller = ImageController(image_topic='/
    usb_cam2/image_raw')
# get environment image
self.environment_image, w, l = self.image_controller.
    get_image()
self.image_controller.record_image(self.environment_image,
    True)
```

Snippet 5.4: Upper Camera topic

4. Robot Controller and AI Manager: the communication between these two nodes is very important. We need confirmation every time the robot controller demands an action from AI Manager, otherwise, actions may get lost and the training and movements from the robot could be chaotic. In the following snippets, there is the implementation of the *ROS service* so the actions are sent and confirm.

```
def get_action(robot, object_gripped):
  relative_coordinates = robot.calculate_current_coordinates
    ()
  rospy.wait_for_service('get_actions')
  try:
  get_actions = rospy.ServiceProxy('get_actions', GetActions)
  return get_actions(relative_coordinates[0],
  relative_coordinates[1],
  object_gripped).action
  except rospy.ServiceException as e:
  print("Service call failed: %s"%e)
```

Snippet 5.5: Robot Controller service

```
def get_actions_server():
  """
  Service initialization to receive requests of actions from
    the robot.
  Each time that a request is received, handle_get_actions
    function will be called
  :return:
  """
  s = rospy.Service('get_actions', GetActions,
    handle_get_actions)
  rospy.loginfo("Ready to send actions.")
  rospy.spin()
  rospy.on_shutdown(save_training)
```

Snippet 5.6: AI Manager service server

5. AI Manager and Onboard Camera. The onboard camera will take a picture before a pick action. The images will be saved into a topic called

'usb_cam/image_raw'. Then AI Manager will subscribe to the topic and wait for the image to arrive

```
def get_image(self):
  msg = rospy.wait_for_message(self.image_topic, Image)
  return self.to_pil(msg), msg.width, msg.height
```

Snippet 5.7: Onboard camera topic

### 5.2.1 The environment

The environment is a white box where the pieces will be displayed. The robot explores the space available and tries to pick the pieces. will be defined with parameters and restrictions.

Due to the limited mobility of the robot, the environment is restricted. There are places that the robot is not capable of reaching. The limits of the box are very restricted, if the robot cannot reach some of the pieces, the learning can be affected, and it will be impossible to empty the box if there are pieces in those points.

In the following table 5.1 there are some of the parameters used to set the restrictions and the limits of the environment.

| PARAMETERS | MEASURES |
|---|---|
| X length | 0.017m |
| Y length | 0.225m |
| Pick Distance | 0.01m |
| Action Distance | 0.02 |
| Camera Security Margin | 0.035 |
| Cartesian Center | [-0.31899288568, -0.00357907370787, 0.376611799631] |

Table 5.1: Environment parameters

'X length' and 'Y length' represents the size of the sides of the box and, for example, the 'Camera security margin' helps to set the boundaries because the camera is very close to the gripper, so it may crash when the robot is going down.

### 5.2.2 Vacuum Gripper

The function of the vacuum gripper is to collect the pieces every time the algorithm decides that the next action is "pick".

In order to do that, the sensors placed in the gripper have three tasks:

1. Due to the restrictions about the speed when performing the pick action. It is important to control how the robot goes down. In order to do that, the sensor detects if gripper is in contact with a piece. This information should publish in the *ROS topic* called */distance*. The robot stops when it is in contact with an object. This object can be a piece, the box or maybe another obstacle. This way of detecting the distance is also safer when interacting with human-performance.

2. The vacuum gripper cannot be activated during all the training. For this reason, the pump is only active when there is an order to switch on the motor of the pump. The gripper must be subscribed to the topic */switch on off*.

3. The gripper must detect if the object was successfully gripped or not. It publishes the information in the *object gripped* topic.

These sensors for the vacuum gripper are connected to an *Arduino* card. At the same time, the information of the sensors will be send to the Raspberry by the ROS serial connection.

Figure 5.5: Vacuum Gripper for the robot

### 5.2.3 Robot Controller

Robot Controller is a node containing the instructions for the robot. This node will receive the actions from *AI Manager* and will send the instructions to the robot.

Robot Controller is also in charge of calculating the current position of the robot, transform coordinates, change speed. This node is where the movements are defined. Also, Robot Controller will subscribe into the necessary topics while publishing others to the communicate and perform correctly.

The actions that Robot Controller can demand are the following:

1. Go to a specific coordinates in the environment.
2. Perform a pick action and take a picture from camera onboard.
3. Go north, south, east, west depending on the output of AI Manager

This actions are chosen based on the decisions from the AI Manager. That means that the Robot Controller node will be just a tool for intermediation with the Universal Robot Driver. The one in charge of moving the robot and translating the instructions.

### 5.2.4 Image Processing

This block is the core of this project.

Images are the most important input in the project. The pictures contain the information about the environment and based on them, the models must extract knowledge to make the best decisions in order to improve the performance of the robot. This node is where these images are being processed.

In figure 5.6 there is a diagram of all the steps previous to the AI Manager block.

The images are taken from the onboard camera of the robot. The 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 prepared for the model, some of them are used for training and the rest for test and validation. The images are loaded in their respective data-loaders and they are fed into the model. The model is a CNN model 6.7.2. 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.

In figure 5.6 it is possible to see the pipeline of this node.

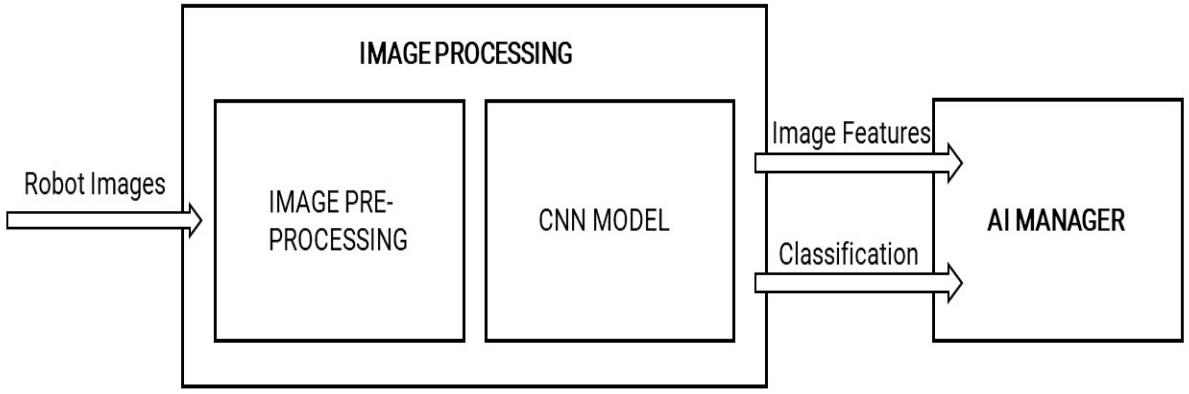



Figure 5.6: Image Processing steps

### 5.2.5 Block Detector

At first, whenever the robot entered a terminal state (successful pick or going out of boundaries) the robot performed a random movement. It will placed anywhere in the environment. Most of the times, if the box was almost or half emptied, the robot went into the white spaces. This usually happened when the algorithm was in an exploratory or random phase in the *Reinforcement Learning* algorithm training.

In order to avoid this useless movements where the robot wasted a lot of movements, the *Computer Vision* and *Image Recognition* techniques came very handy. The solution is to treat the image so the contours of the pieces can be detected and with that information, we can calculate the places with more pieces in it. The environment is divided in $n^2$ quadrants so, after calculating the quadrant with more pieces, the robot can be placed in this quadrant, just sending the coordinates.

Using some *Image Processing* 6.5.1 and techniques of contour-detection 6.6.1, the image is treated in order to standardize it. The goal is to have the most similar images, referring to the external conditions such as brightening or the colour of the image.Once the image is processed and the contours are detected, it is time to find out where is the place where the robot can have more success for picking new pieces.

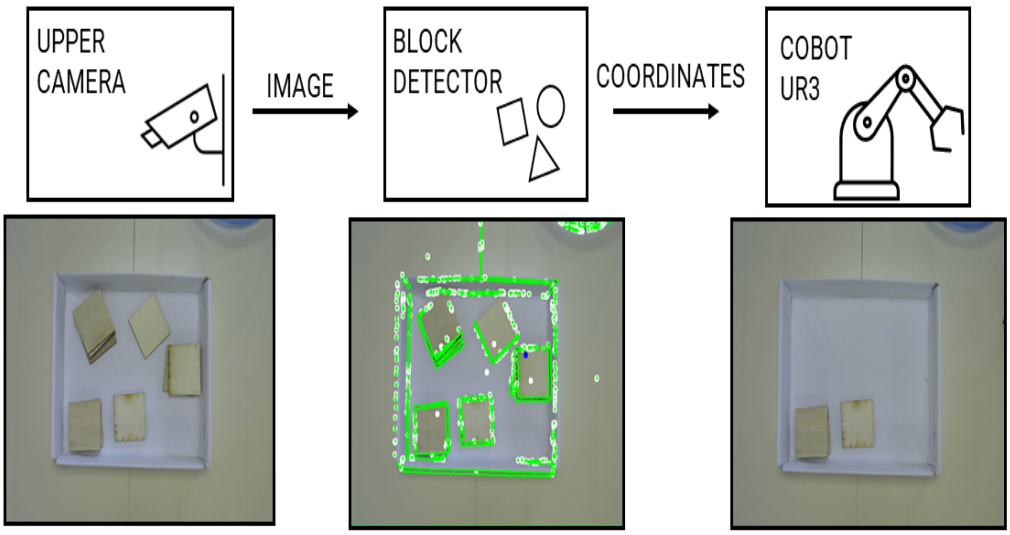



Figure 5.7: Diagram of How Block Detector works

### 5.2.6 AI Manager

This node can be described as the *"intelligence"* of the robot. It is the node in charge of the learning process with the *Reinforcement Learning* algorithm. The algorithm chosen is a *Deep Reinforcement Learning Algorithm*.Each time the robot perfoms an action, it reaches a *new state*. When the agent gets into a new state, this state can be an intermediate state or it can be an end state. When the state is a terminal state, the episode finishes and a new one begins.

Each action has a reward associated. Like in every *Reinforcement Learning* problem, actions have a value, and the main goal is to maximize the cumulative reward in the long run. This means that rewards are a very important parameter and they can vary depending on the performance and how the robot must behave.

In table 5.2 there is the summary of the rewards used for the *Reinforcement Learning* algorithm, that the *Computer Vision* block will use to test and improve the results.

| Action | Reward | Terminal State |
|---|---|---|
| Successful pick | +10 | True |
| Unsuccessful pick | -10 | True |
| Out of boudaries | -10 | False |
| Other movements | -1 | False |

Table 5.2: AI Manager rewards

The main goal of the project is to pick pieces successfully, so the "successful pick" is the only action with a positive reward. By only rewarding positively the success, the algorithm is learning to improve the precision. Terminal negative states like "failure" or "out of boundaries" are the worst situations, the reward is negative. And finally, other movements are penalized with a -1 so the robot is not tempted to keep moving around the environment without making an attempt to pick pieces.

The goal of this project is focused on the *Computer Vision* problem, so the *DQ-Learning* algorithm implemented is consider as a "black box" and it will be used to check if there is a better performance in the model, when the *Image Recognition* has a positive increase in performance.

# Chapter 6

# 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.

In this section, there is a description of every step in the *Computer Vision* pipeline.

## 6.1 General Overview

*Computer Vision* applications can vary, but most of the steps for processing images follow a similar pipeline. In figure 6.1 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.

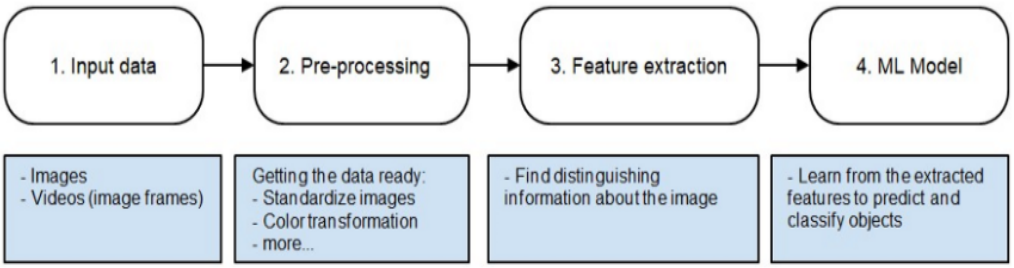



Figure 6.1: Computer Vision Pipeline [45]

## 6.2 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: [46]

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 [47]:

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.

The world coordinates are transformed into 3D camera coordinates using the Extrinsic Matrix. Once the Camera Coordinates are obtained,the Intrinsic Parameters of the camera are used to compute the projection of the coordinates in the image plane in the two system coordinates (u,v). This computation can be achieved with the Intrinsic Matrix.

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Xc \\ Yc \\ Zz \end{bmatrix} \tag{6.1}$$

With the following transformations:

$$u = \frac{u'}{w'} \quad v = \frac{v'}{w'} \tag{6.2}$$

The goal of the calibration is to find the K, R (Intrinsic and Extrinsic) matrices and the translation vector t by using some known 3D points and the corresponding image coordinates (u,v). Once we get the parameters the camera is finally calibrated.

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.

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 6.2 there are two examples of the chessboard from different angles. Approximately a good value of pictures for the calibration can be around 20 samples.

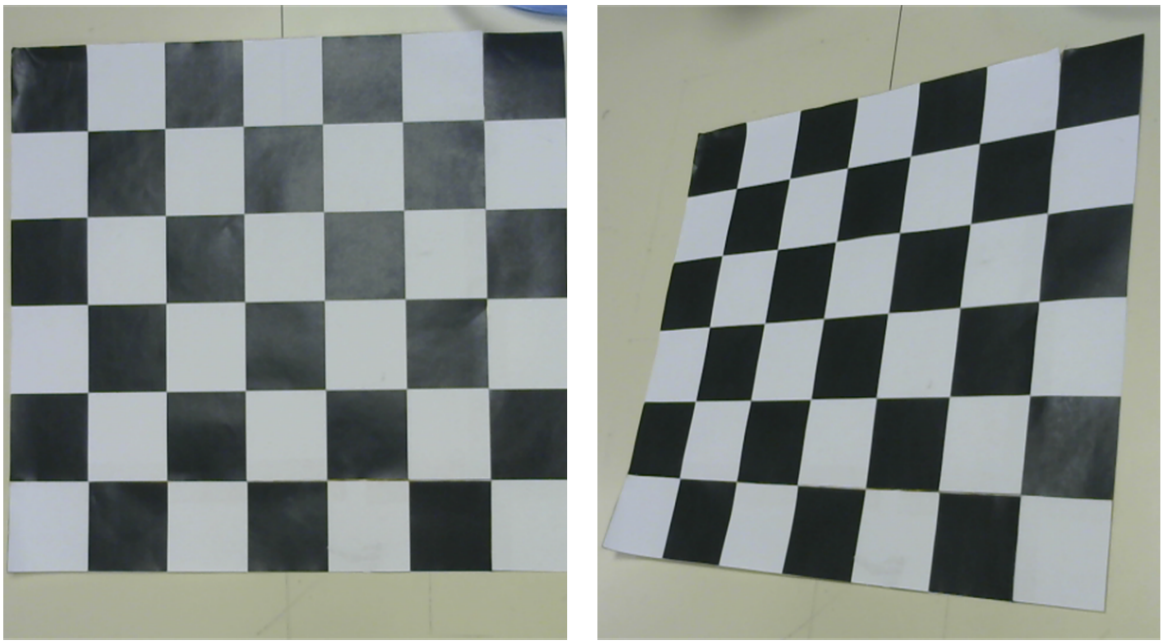


Figure 6.2: Chess board for Camera Calibration

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 6.3 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. 6.3

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

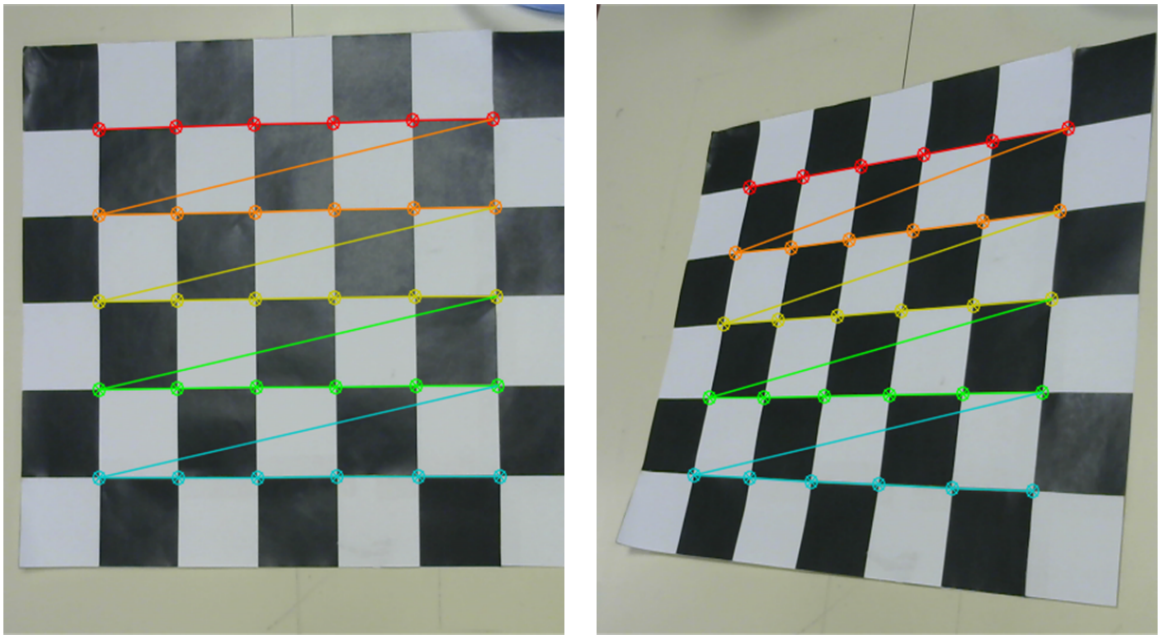


Figure 6.3: Camera already calibrated

## 6.3 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.

Figure 6.4: Example of image taken from the onboard camera

In figure 6.4 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.

## 6.4 Understanding the image

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 6.5 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.

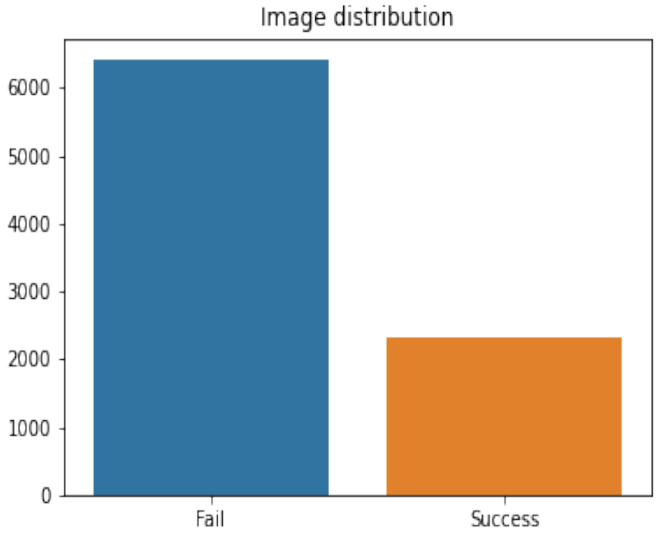



Figure 6.5: Distribution of dataset

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 6.8

**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 6.1: 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 6.1: 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.

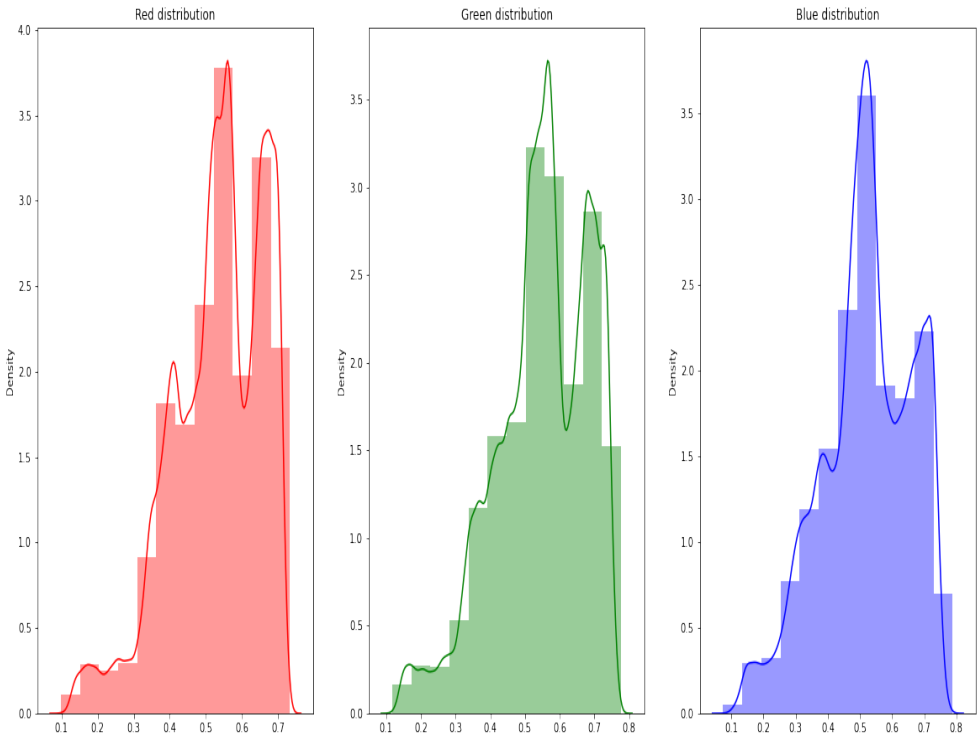



Figure 6.6: Colour distribution of the image

## 6.5 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.

### 6.5.1 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 6.7 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.

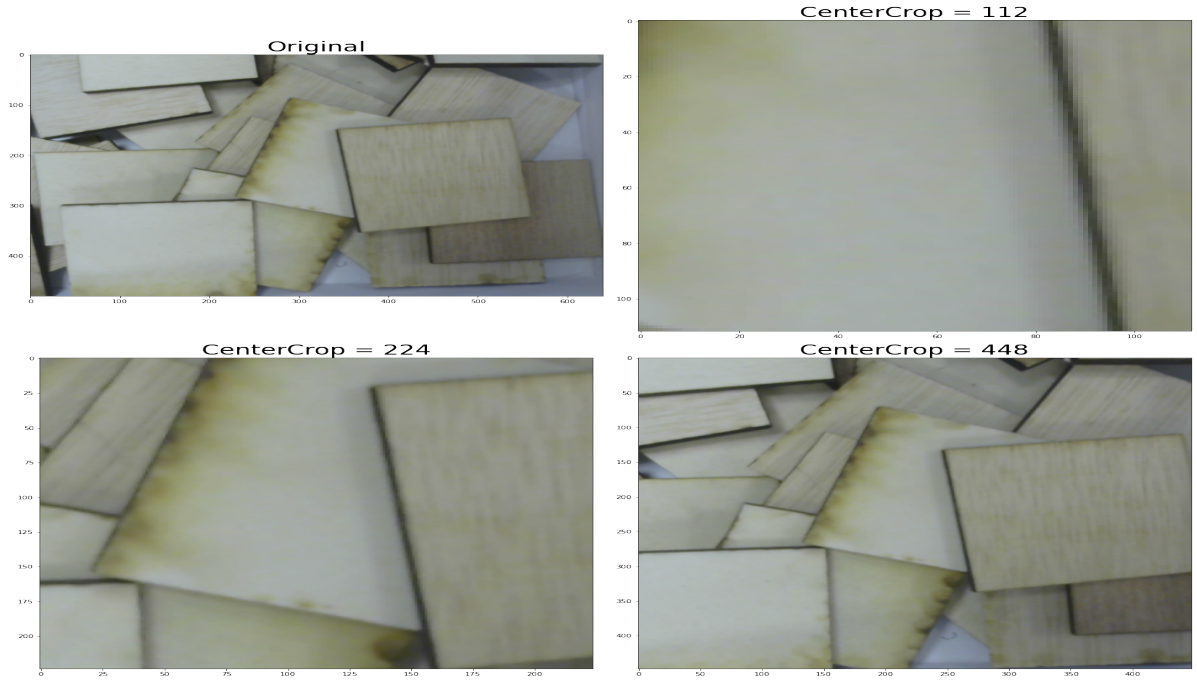



Figure 6.7: Center Crop

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 6.8 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.

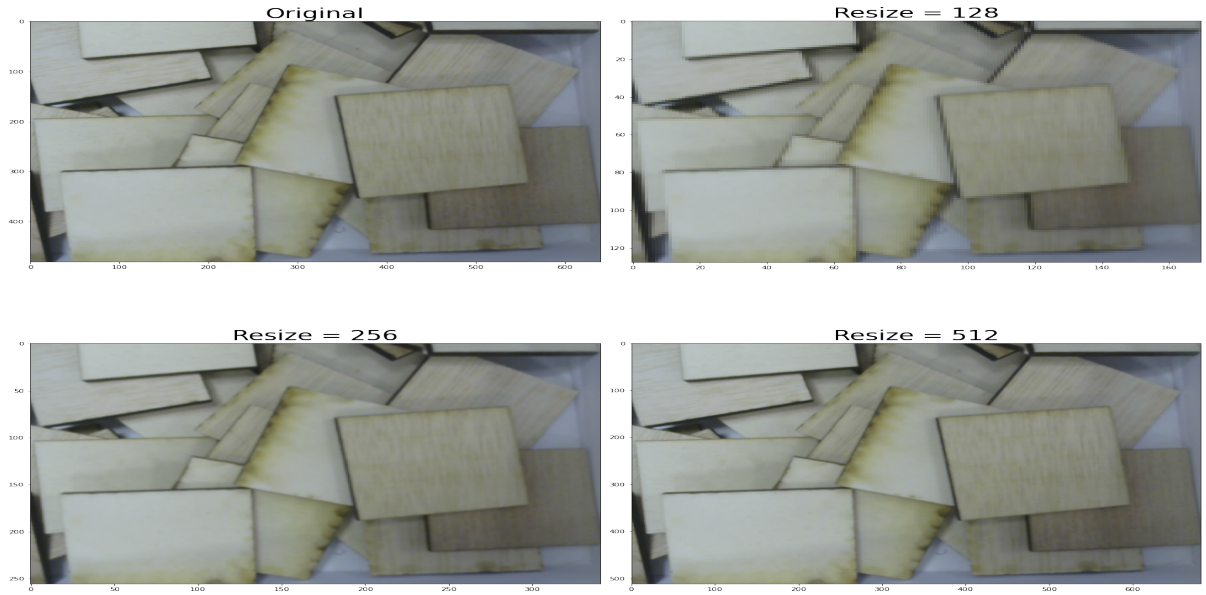



Figure 6.8: Resized images

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.

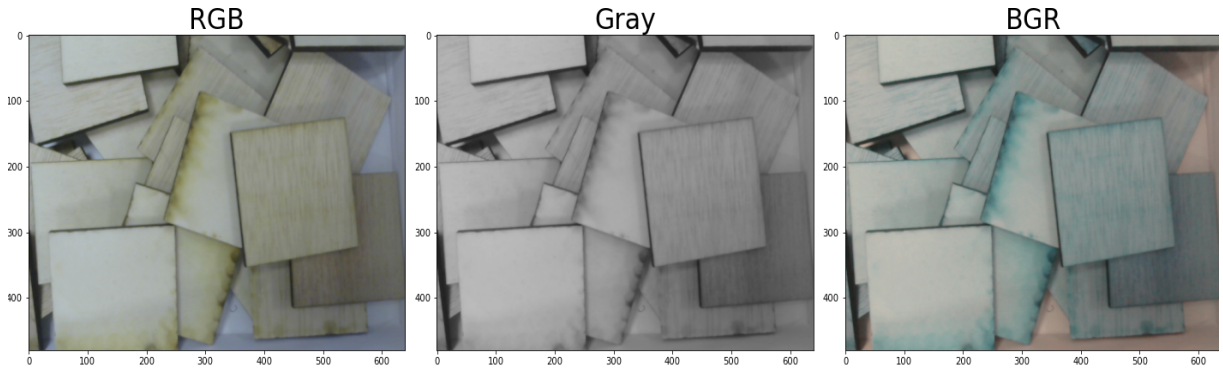



Figure 6.9: Blur techniques

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.

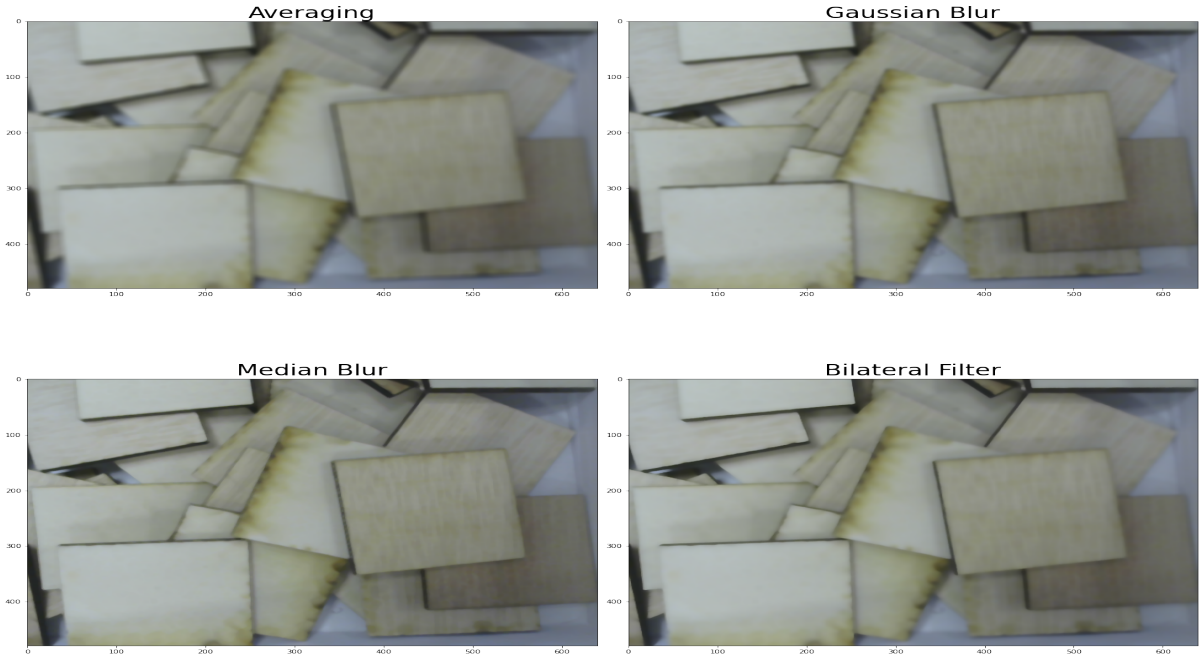



Figure 6.10: Blur techniques

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.

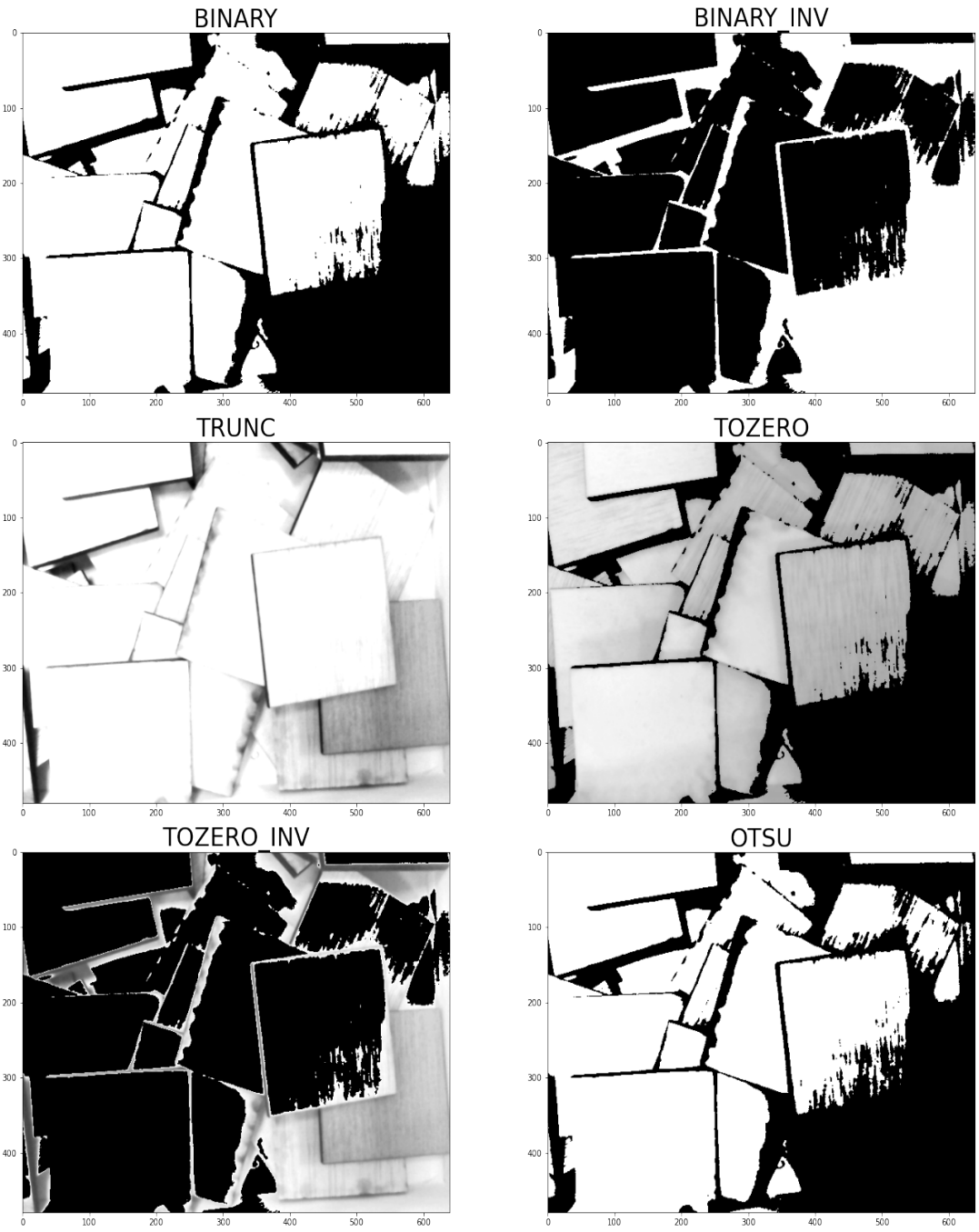



Figure 6.11: Threshold techniques

- Adaptative Threshold: The next version of Thresholds. This techniques can adapt better to the characteristics of the image, with better perfor-

mance but adding more time of processing. In figure 6.12 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.

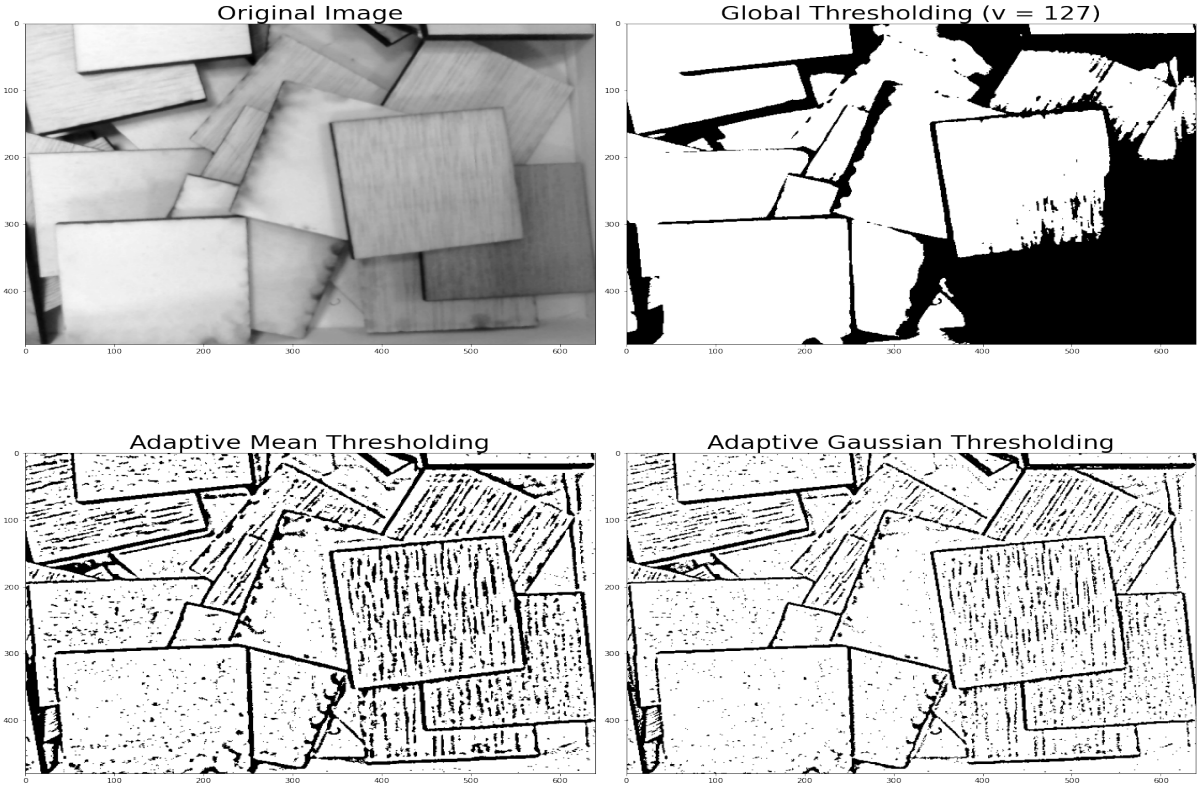



Figure 6.12: Threshold techniques

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 6.13 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.

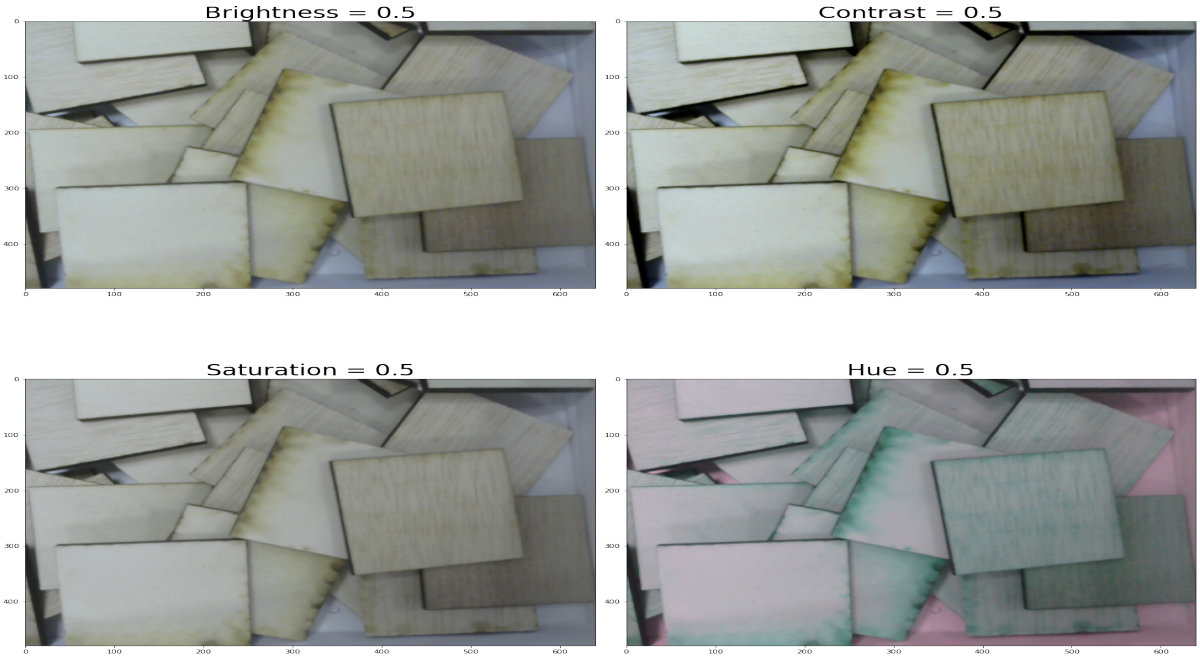



Figure 6.13: Brightness and contrast

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])
```

**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.

### 6.5.2 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* [48] 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 6.14

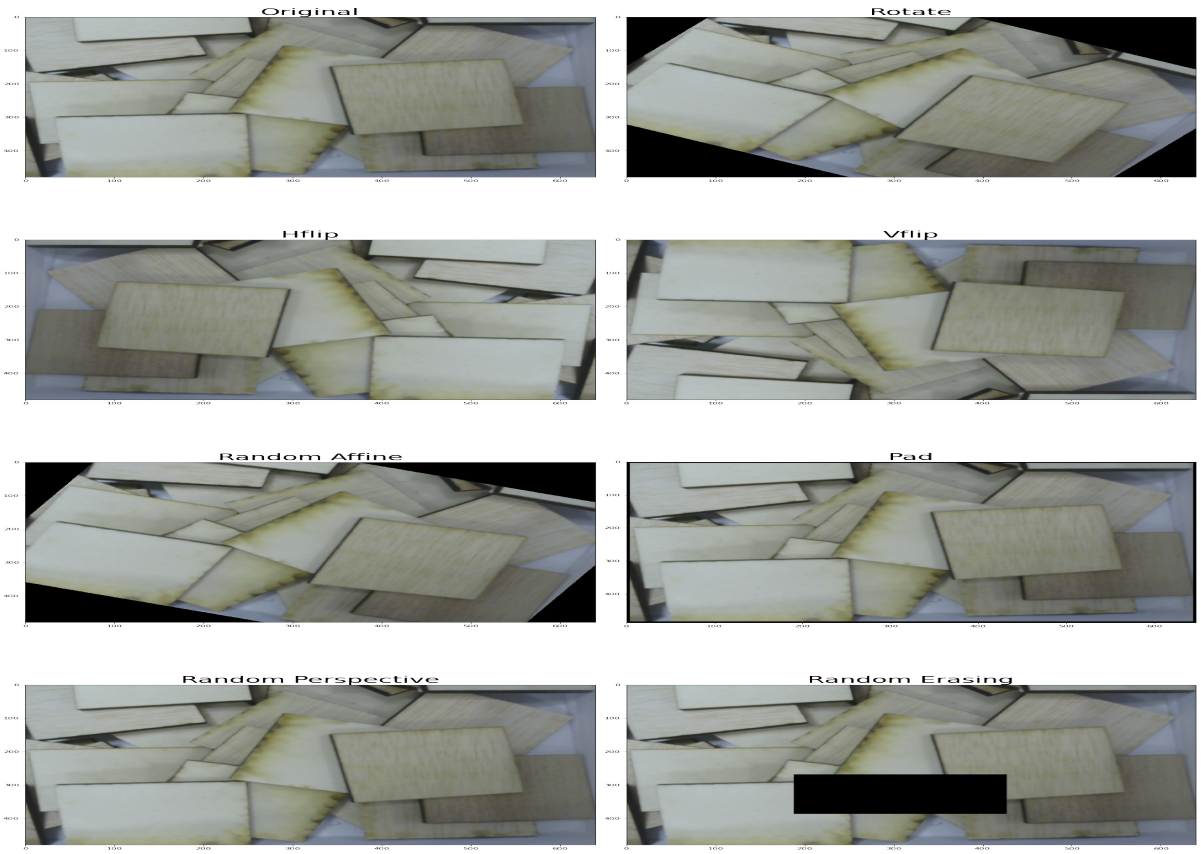



Figure 6.14: Different data augmentation techniques

**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

## 6.6 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.

### 6.6.1 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 6.15 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 6.15 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.

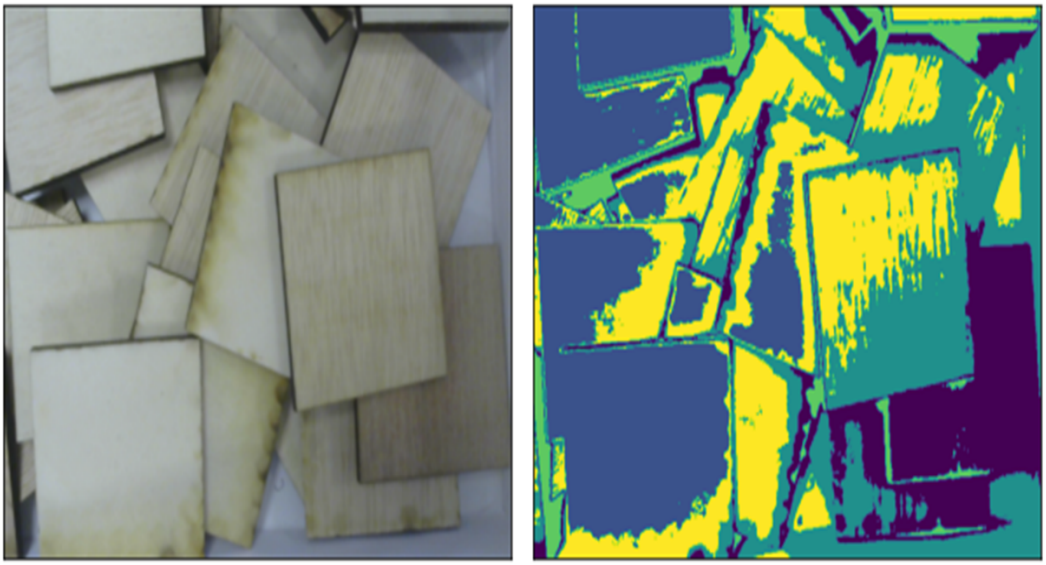


Figure 6.15: Original Image vs. Cluster features

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 6.16 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.

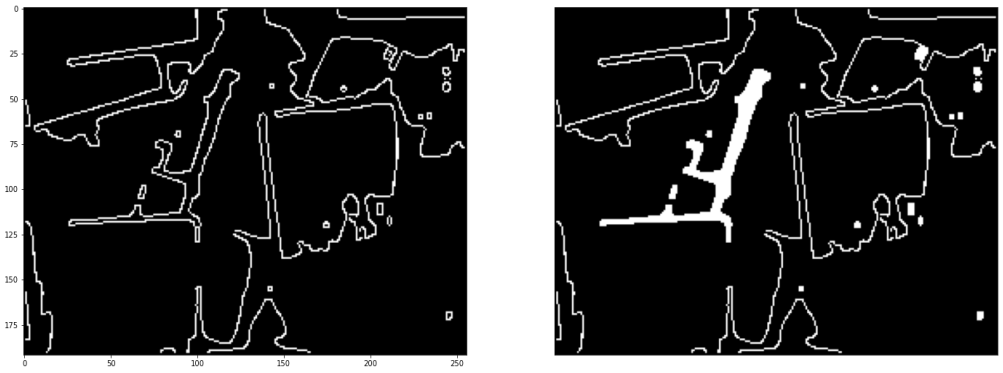



Figure 6.16: Canny Edge Detector

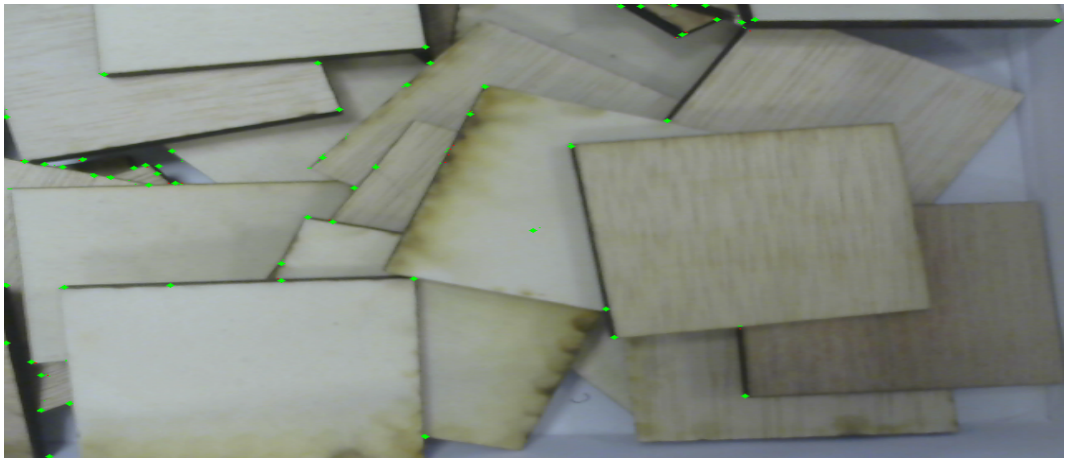


Figure 6.17: Harris Corner Detector

In figure 6.18 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* .

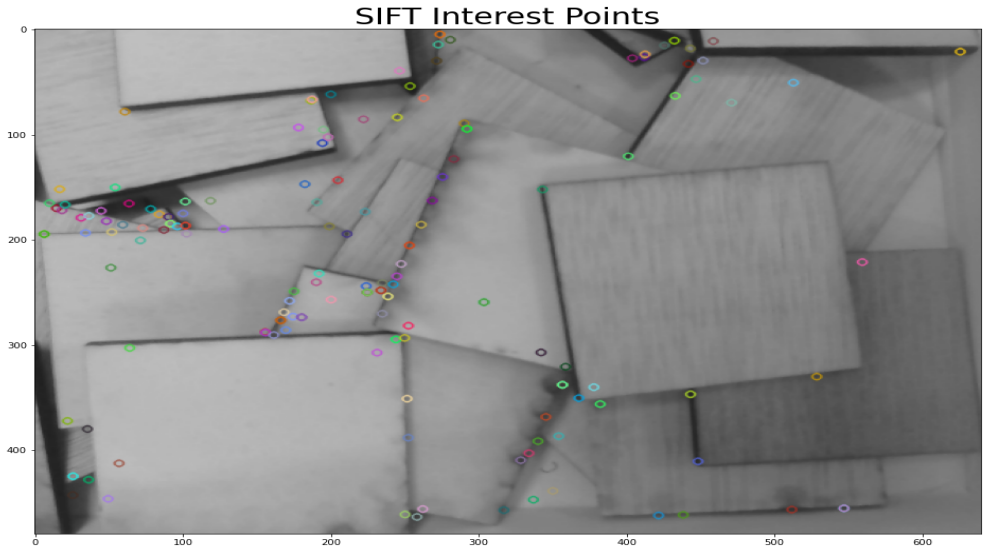



Figure 6.18: SIFT method for Feature Extraction

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.

### 6.6.2 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.

In figure 6.19 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.

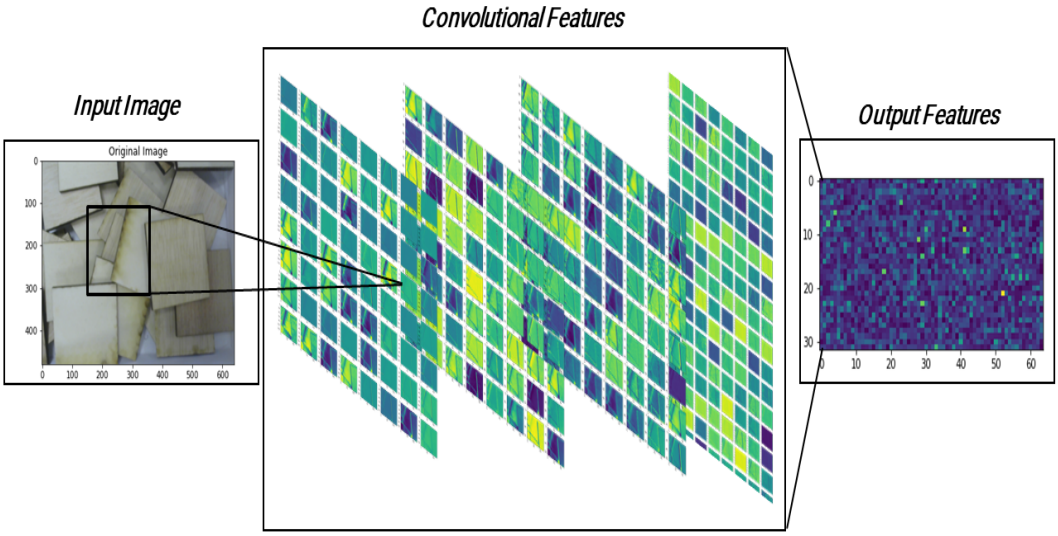



Figure 6.19: Features extracted from the input image

The first convolutional layers are high-level filters, so in 6.20 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.

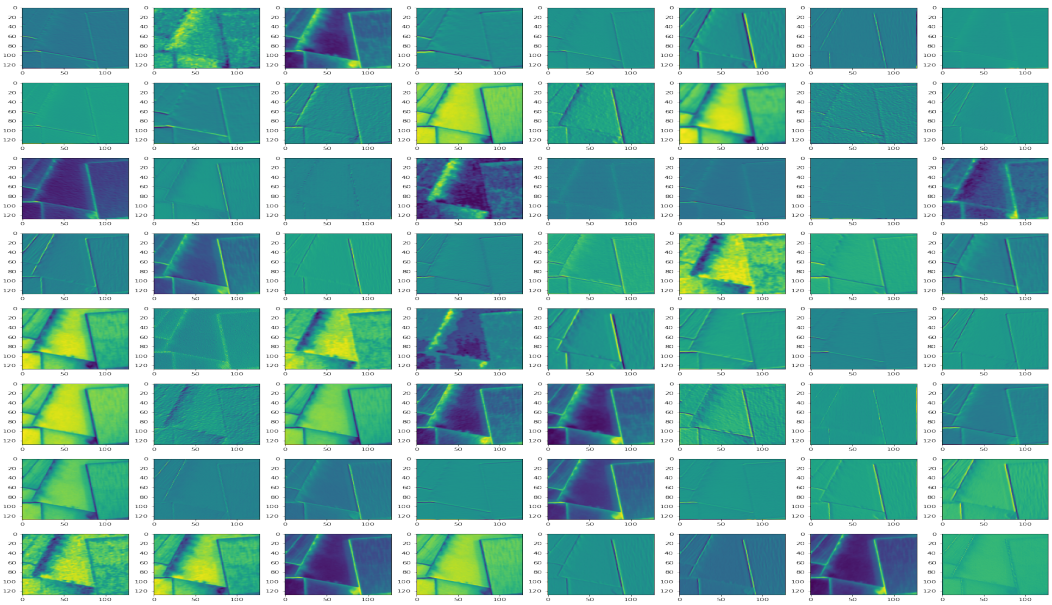


Figure 6.20: First Convolutional Layer

The deeper the layer analysed, the patterns and the filters become harder to understand. Figure 6.21 represents a convolutional layer inside the whole architecture of a ResNet50 model. The patterns are starting to vanish.

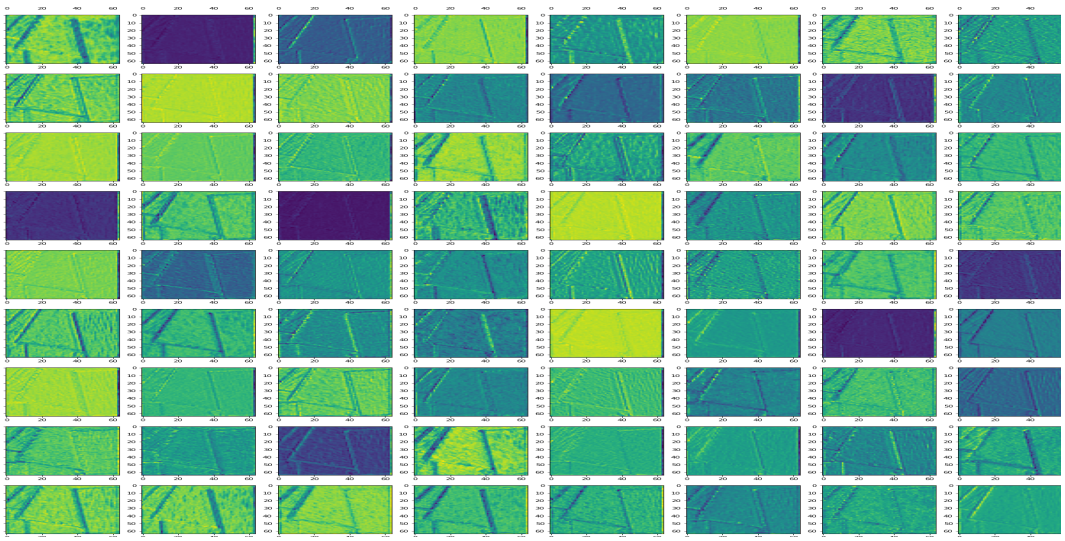


Figure 6.21: Deeper Convolutional Layer

This last figure 6.22 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.

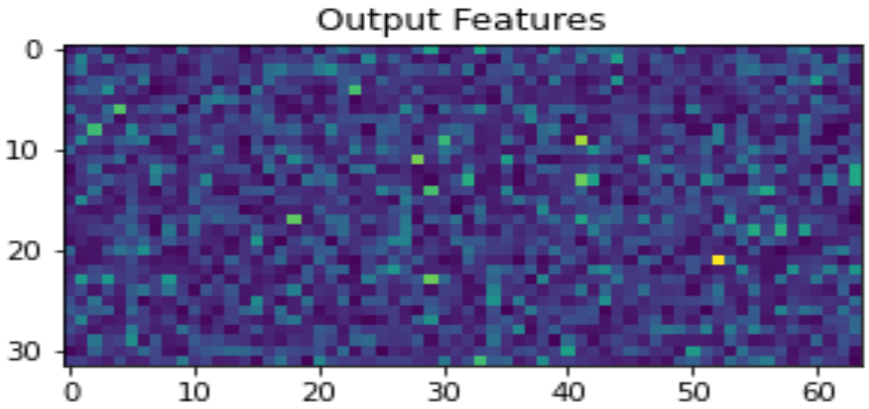



Figure 6.22: Feature Vector Output

## 6.7 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 6.5.1) 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.

### 6.7.1 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.

**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 6.2 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 6.2: Budget

From the results in table 6.2 the ResNet family of models are a good choice for starting the testing.

### 6.7.2 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." [49]

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 6.2: 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 6.3: 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'.

## 6.8 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:

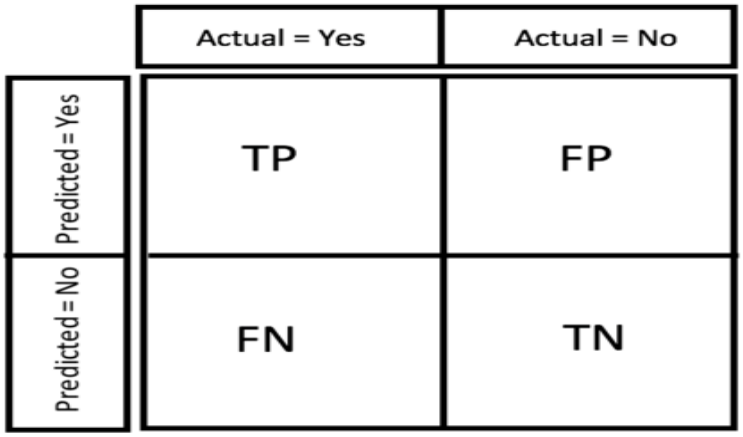



Figure 6.23: Confusion Matrix [50]

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 6.4, 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 5.2). 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 6.3: F$\beta$ score [51]

- **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.

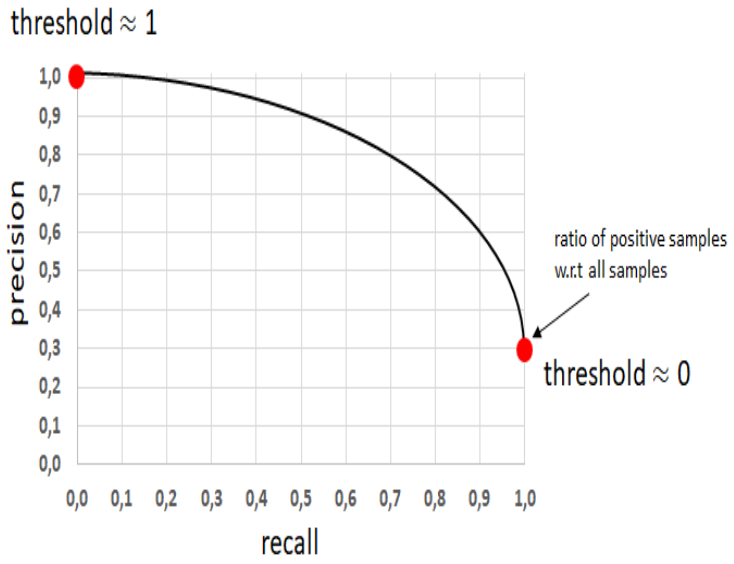



Figure 6.24: Confusion Matrix [52]

- **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. [53]

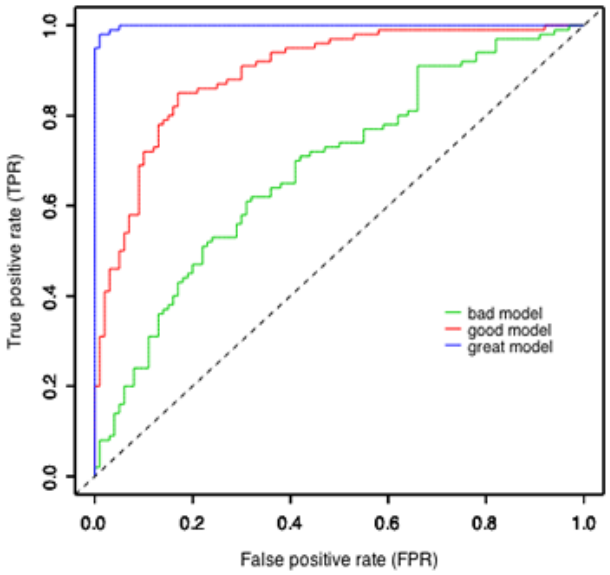



Figure 6.25: Confusion Matrix [50]

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.

# Chapter 7

# Results

This chapter shows the results of the *Computer Vision* vision problem, the main focus of this project, and how the results of the global solution are working, when adding the *Computer Vision* model to the *Reinforcement Learning* algorithm.

## 7.1 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 7.1 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.

Using the parameters obtained with the camera calibration (see 6.2), 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.

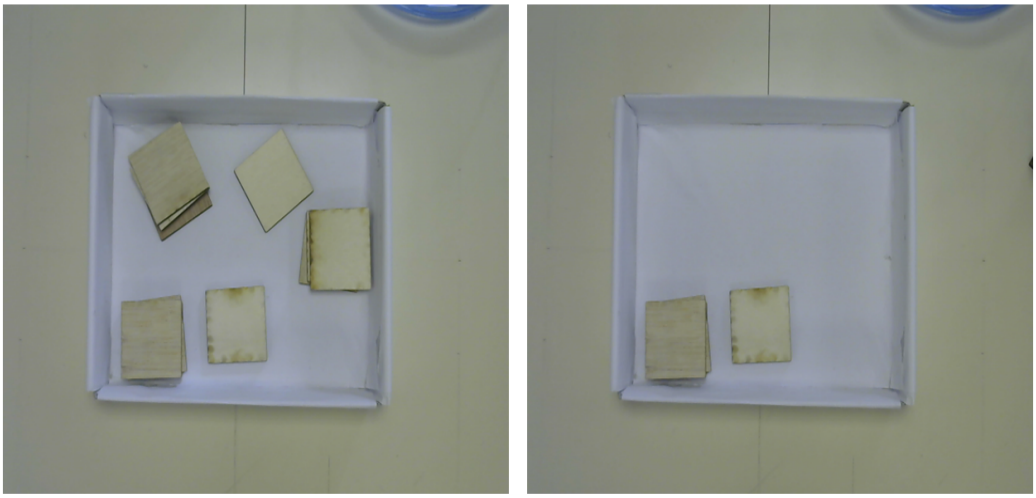


Figure 7.1: Original raw pictures

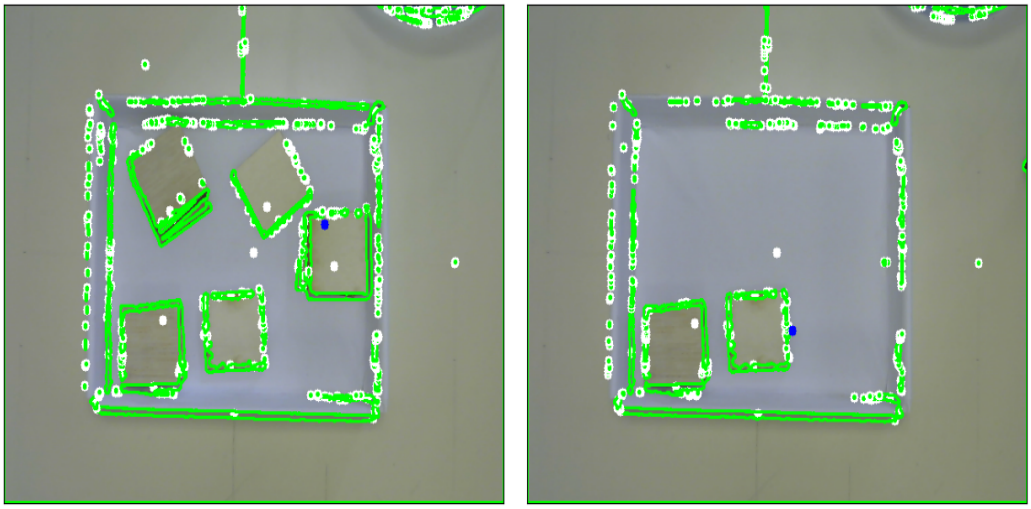


Figure 7.2: Results after the contour detection

After the processing of the images, in figure 7.2 the images show one of the most probable places where the robot can be most likely to succeed. This part of the code, avoids the random movements so the robot obtains better results and avoid blank spaces.

## 7.2 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'.

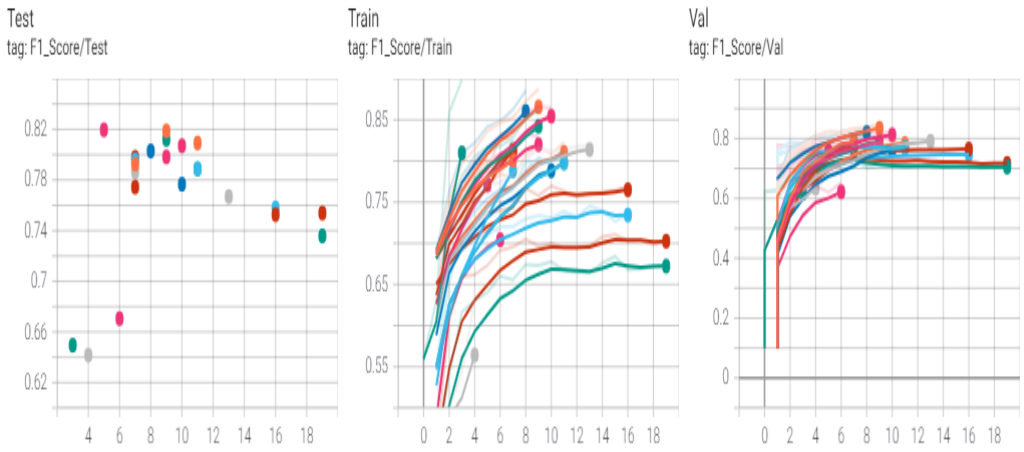



Figure 7.3: F1 Comparison between VGG and ResNet models

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.

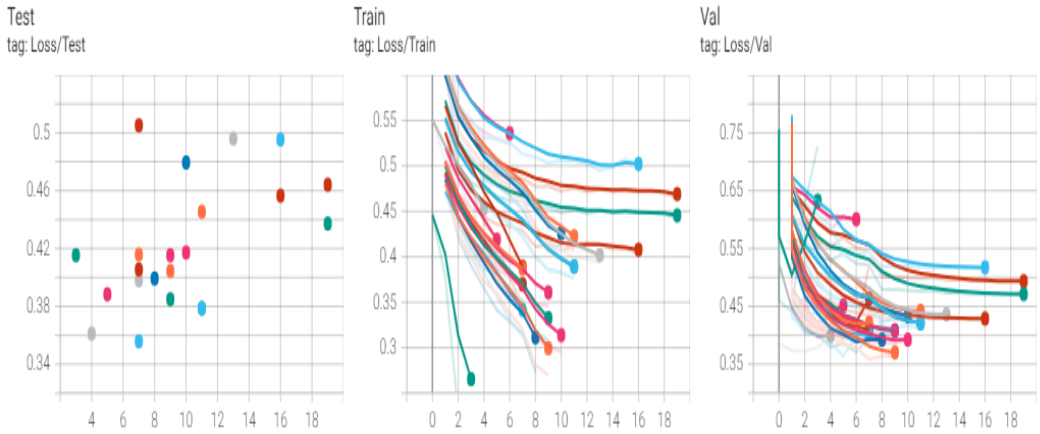



Figure 7.4: Loss Comparison between VGG and ResNet models

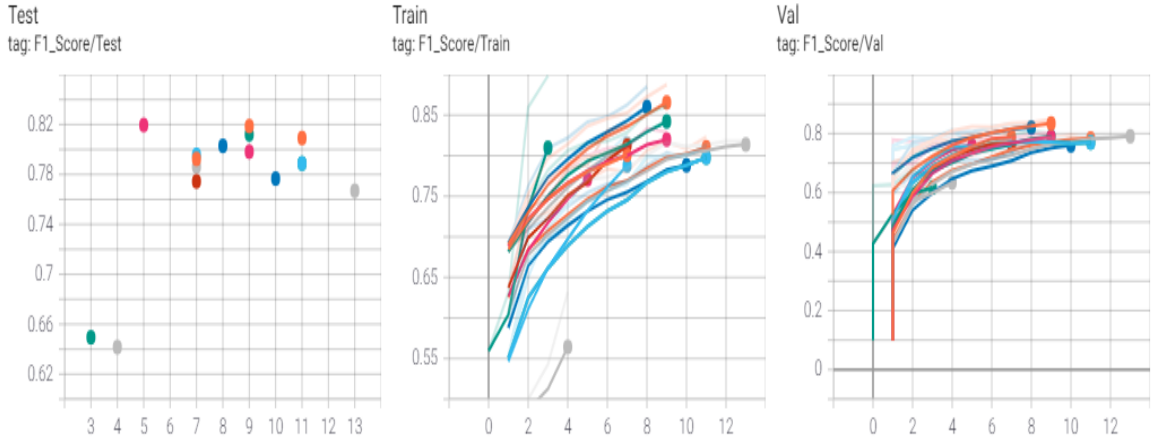



Figure 7.5: F1 Comparison between ResNet models

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.

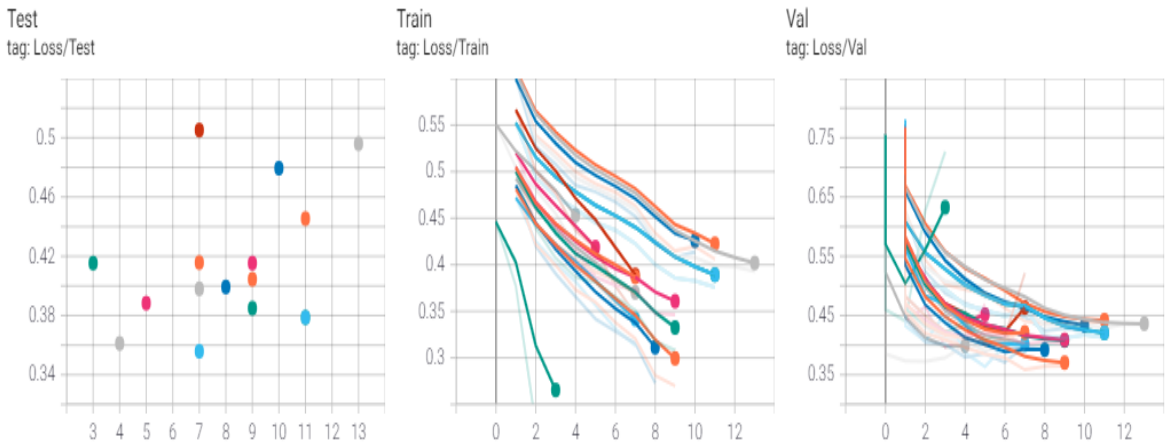



Figure 7.6: Loss Comparison between ResNet models

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.

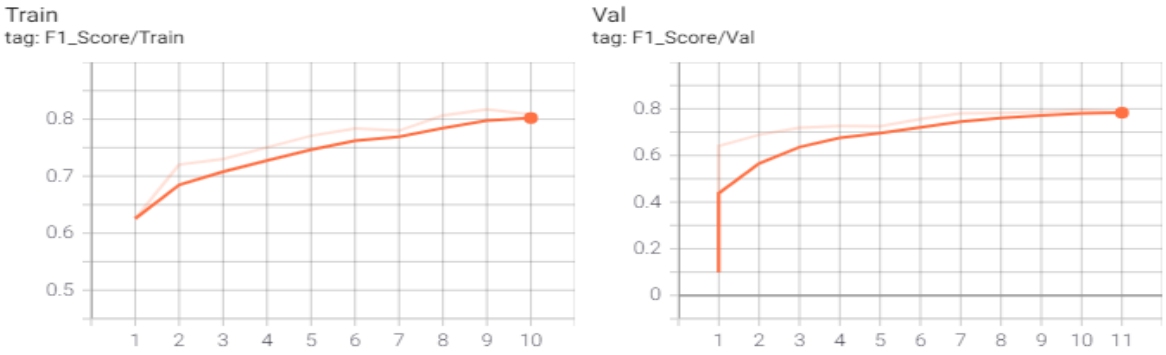



Figure 7.7: F1 Metric for chosen model: ResNet50

In figure 7.8 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 6.7.2

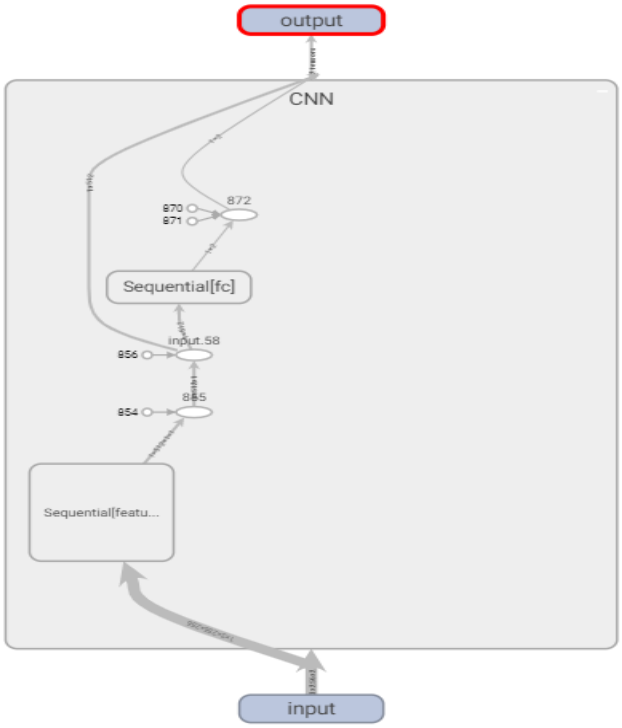



Figure 7.8: Model architecture

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.

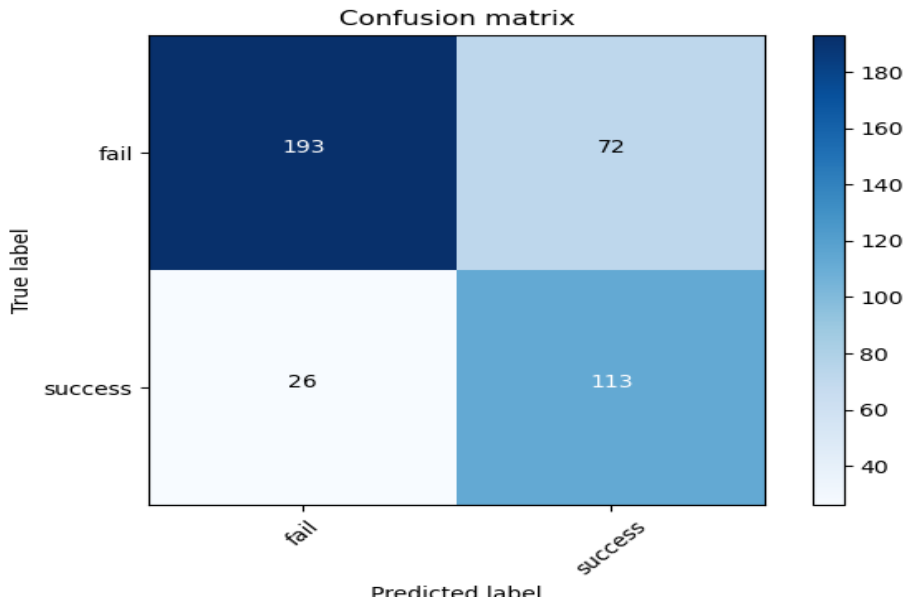



Figure 7.9: Confusion Matrix

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 7.9 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 7.10, 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.

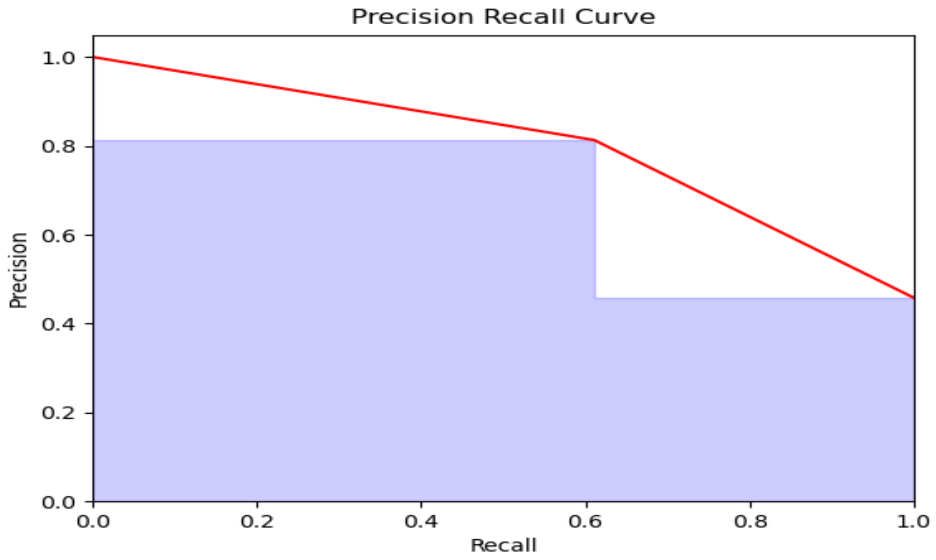



Figure 7.10: Precision-Recall Curve

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.

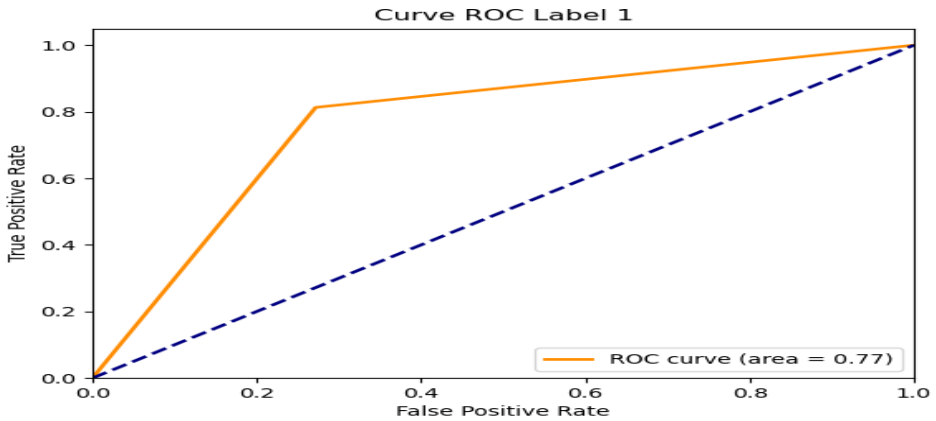



Figure 7.11: AUC-ROC Curve

Once the model has been validated with the metrics, it is time to see the

performance with real inputs. In 7.12 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.

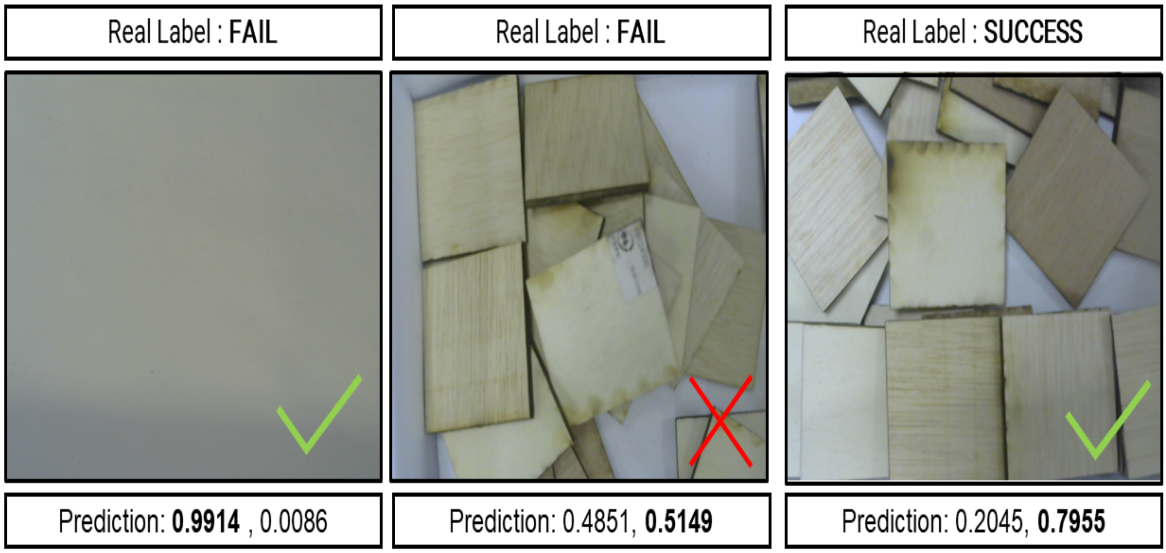



Figure 7.12: Predictions from the model based on different inputs

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.

Finally, nn the next chapter, the results of the whole project will be evaluated. The *Computer Vision* techniques will be applied to the *Reinforcement Learning* algorithm.

## 7.3 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.

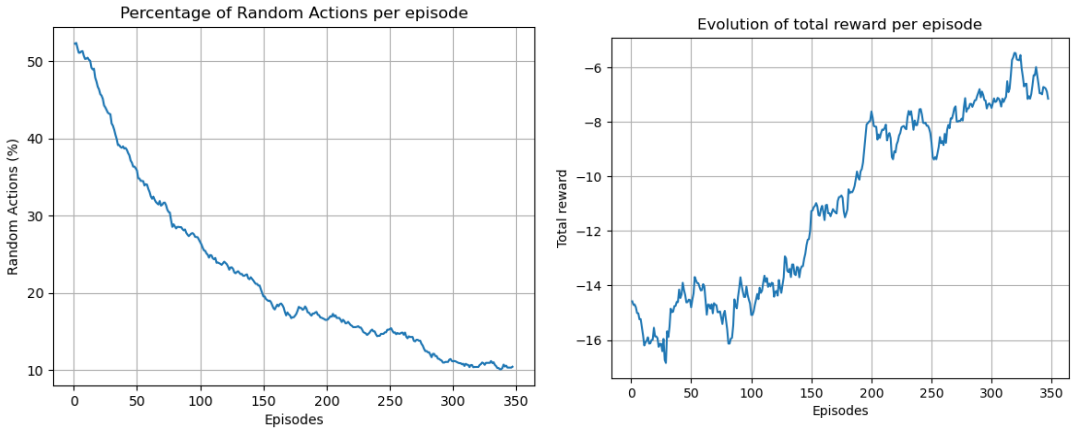
2. **Evolution of rewards per episode** This metric can be analyses together with the Random actions per episode metric. When the robot starts using the knowledge of the model, the rewards should be increasing also. The goal of the *Reinforcement Learning* algorithm is to allow the robot to make the 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.

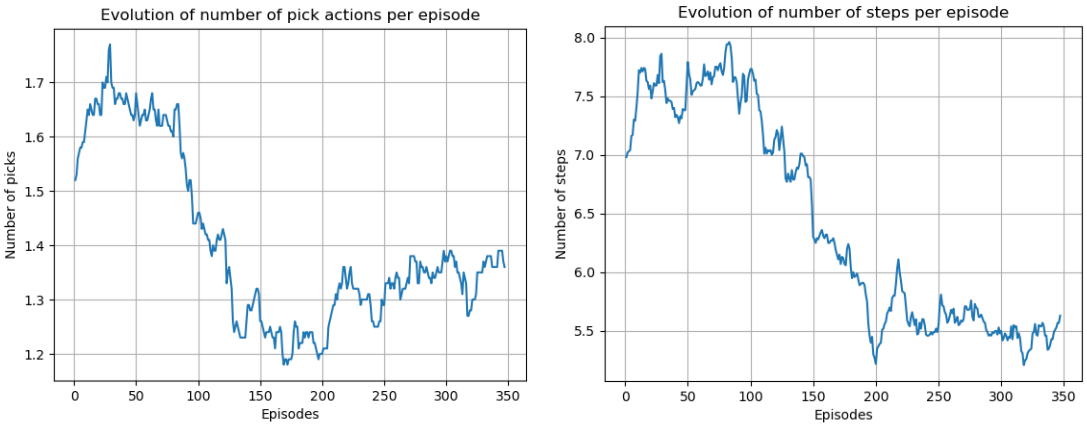



(a) Random actions per episode

(b) Evolution of rewards per episode

Figure 7.13: Is the robot learning?

During a training of 300 episodes, in figure 7.13 there are the curves for the random movements and the total rewards accumulated per episode. There is a satisfactory increase in the rewards at the end of the training, which is a very positive sign of learning. The value around the 300 episodes is closer to -6, which makes sense, due to the fact that every 'north', 'south', 'east' and 'west' movement has a reward of -1, so if the robot needs to move a little bit to get to the optimal position, there should be, also negative rewards. However, it is possible to see that there is still a 10% of random actions in the environment, so some of the actions can be influenced by the random exploration.




(a) Number of picks per episode (b) Number of steps per episode

Figure 7.14: Is the model improving?

In figure 7.14 there are the curves from the steps per episode and the pick movements, also per episode. In this case, the tendency that shows the project is working and improving is a decrease in picks and movements. This is also a positive sign of the success in addition to the increase in reward, previously mentioned. Also, is worth mentioning the fact that the box is constantly being refilled during training. So, there is always different number of pieces in the box, sometimes the robot can be trying to pick something, but there are no pieces. This is dangerous because it can affect the performance. The robot should not stayed a lot of time without pieces in the box.

Even though there is a lot of space for improvement in the project, the best comparison is when the project was an almost random bin-picking task. The *Computer Vision* and *Reinforcement Learning* models were not tuned correctly and we could not called the robot "intelligent".

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

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

Table 7.1: First training

As it can be seen in the table below 7.1 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.

**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.

# Chapter 8

# Conclusions

This project has set the bases for the improvement and continuance of solving the bin-picking problem with *Artificial Intelligence* . Even that most of the goals were achieved, it is true that as in every project, there have been problems and changes from the original plan. Besides, there are a few conclusions obtained from all the development of the project that can be useful when continuing with the research.

## 8.1 The problem of unbalanced data

In any *Deep Learning* problem data is the most important asset. Without information *Machine Learning* can not be performed. One of the most important tasks in this project was to build a dataset with an enough size so a correct performance could be achieved with *Deep Learning* .

Due to the nature of the project and as mentioned in the previous chapters, the dataset was unbalanced. Unbalanced data is one of the most common problems in *Machine Learning* . This causes errors in predictions and in performance, so it must be solved before training the data. There are different techniques of balancing the datasets so the training data is prepared to avoid bad performance.

Other very important issue about the balancing problem is that data should only be balanced in the training dataset. The idea of test and validation tests are that they check how the model will respond to the data unseen. In an real environment, the data will come unbalanced and we will not be able to observe the performance if the dataset is balanced. Of course, the train, test and validation datasets must be completely separated and well differenced in order to avoid other problems such as data leakage. One different problem also would be if suddenly the distribution of data changes, the model obviously will not perform correctly (covariate shift), so from time to time it is important to check that the data samples are similar to the training data.

## 8.2 Tuning a model

The most difficult task in this *Computer Vision* project has been the election of the model. Research and similar problems can help finding the first approach and the state-of-art techniques and methods. Nowadays, almost nobody develops a model from zero. Images are a type of unstructured data where *Deep Learning* is the best solution in the majority of cases. CNN and Transfer Learning, both used in this project, are the most standardized techniques when working with images. However, the possibilities are enormous. Only in PyTorch's framework, the Transfer Learning pre-trained models are twelve. [54] Not only there are lot of possibilities, but each of this models can have their own configuration, the network changes, the parameters are different... Furthermore, you can fine-tune or not the network, depending on the dataset. The combination can be infinite. There are guidelines and scientific articles to help speed the process, but *Deep Learning* can be a very meticulous work that requires very specific knowledge to improve results and get the best model possible.

## 8.3 Is Deep Learning always the best option?

Nowadays, *Computer Vision* and *Deep Learning* are inseparable. CNN is considered the state-of-art for *Computer Vision* problems. However, there are situations where maybe a simple techniques of *Image Recognition* or *Image Processing* are necessary to solve a problem. An example of this is the Block detector. When the upper camera was added, there were two options: training a new deep learning model to process the images of the complete environment or maybe an approach more traditional could be implemented. A few things were considered. The size of the dataset was very small, there were just a couple hundred pictures, which is a very small number of samples for *Deep Learning* . The second key point was the time needed to implement the solution. The main goal was to set quick fix that could improve the performance by removing the random movements. *Deep Learning* requires important amount of times to train. So, instead the resource was to use some of the pre-processing techniques learned and use the manual feature extraction to detect the contours and then, send the robot to one of the places with more probabilities of pieces.

We tend to think that *Computer Vision* problems needs to be solved with complex techniques, but sometimes easier approaches can have quicker results with an adequate performance depending on the problem facing. There are very powerful techniques for processing images that are validated and proved effective in multitude of projects over the last years. *Deep Learning* solutions are perfect for complex solutions, where the environment or the conditions change, so maybe the

best practice is a combination between traditional techniques and neural networks.

## 8.4 From theory to practice

Sometimes it is very difficult to plan the complete workflow of a project. The reason is very simple, complex problems have a lot of delays, change of plans and things that not depend only of the team. Besides, when dealing with *Deep Learning* learning problems, a lot of time to select the best model is required and this hinder the correct of estimation of time that the tasks will take.

Besides, when working with real environments that requires hardware, there are problems with connections or communications between the nodes. Until the project is tested for the first time in the real environment you can not be conscious of some important requirements. One example might be that the robot does not work well with coordinates that are not relative, like in our projects. Other problem about hardware is that the structures or essential parts (like the gripper) must be designed and built. This type of tasks involved a lot of time, and the real tests can not be performed unless the complete architecture is all set.The reality between planing and executing is very different. It is necessary to deal with situations that are not expected or the implementation may not work as expected. Theory can be exact numbers, very good models and tasks completed in time. Practice is more chaotic and focused on problem solving.

## 8.5 When is enough?

*Machine Learning* algorithms can almost always be improved. The model can be fine-tuned, the data can be processed in a new way, new information can be added, dimension reduction can be performed and the possibilities can be almost infinite. This brings the question of when the performance of the model is consider enough. How can be decided that the results are good for the task. Three main responses are obtained from this project. First of all, time is an important factor. It influences how much time is given for the deployment. With infinite time, all the improvements can be done. However, most of the times, time is scarce and this forces developers to focus on things and results that already worked in other projects. Then, it is important to understand the impact of the project and the requirements. Industrialized robots, for example, need a performance and it is not acceptable a bin-picking robot that needs hours and hours to empty a bin. And the last, and most important factor is metrics. It is vital to have tangible metrics and objectives. Progress needs to be measured, if there are no clear objectives, it is impossible to know when the model is enough or how far the results are from

the goal.

## 8.6 Complex structures and points of failure

One of the most difficult problems to solve was the integration of the whole architecture. The communication of all the nodes is done via TCP/IP which means that our structure depends on the network. Also, if there is a failure on writing on ROS topics, the other nodes can keep waiting for a message that will not arrive. This type of problems have been solved using ROS services or sending a burst of messages so the changes of losing messages decrease significantly. Also, the robot was configured to stop moving when the connection is lost. This was really helpful so the robot did not crash or collapsed due to communication losses.

However, if one of the nodes stops working the whole structure is compromised. If the camera does not take pictures or the gripper does not detect the objects gripped, the whole system crashes. For academic research the current structure works and it is enough, but for industrial environments this is not admissible and some solutions should be prepared.

# Chapter 9

# 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.

## 9.1 Tune the models

Tuning the parameters in *Deep Learning* models can be one of the most complex tasks when training a model. There are hyperparameters 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.

## 9.2 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 imporve the information of the images in the model could be the use of a 3D camera or a implementation of a Multiflash solution.

This Multiflash 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 [56] and [57]

## 9.3 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.

# Chapter 10

# Aligment with the Sustainable Development Goals

Artificial Intelligence is a very popular trend right now. There a lot of technical studies and research being conducted, and new applications and improvements are achieved every day. Innovation plays an important part in the development of a more sustainable future. Research and new technologies must be used as tools to improve our current situation and bring better alternatives into the industry.

The Sustainable Development Goals aim to improve the dramatic and complicated situation that we are living globally by year 2030. This project is an extended research in new techniques and algorithms, with the clear objective to improve the industrial situation and the performance of machines. There is a closely link to some of the goals and objectives that should be achieved. Specifically, we found the following goals: [55]

- **Goal number 8: "Decent work and economic growth"**:
  - **8.2.** "Achieve higher levels of economic productivity through diversification, technological upgrading, and innovation, including through a focus on high value added and labour-intensive sectors"
  - **8.3.** "Promote development-oriented policies that support productive activities, decent job creation, entrepreneurship, creativity, and innovation, and encourage the formalization and growth of micro-, small- and medium-sized enterprises, including through access to financial services" The goal is to achieve a solution ready to be implemented in industrial scenarios. The project is thought as an easy solution, with easy components that can be adapted to several problems without a big budget. This idea can set bases for new ideas and applications.
- **Goal number 9: "Industry, Innovation and Infrastructure"**:

  - **9.5**. "Enhance scientific research, upgrade the technological capabilities of industrial sectors in all countries, in particular developing countries, including, by 2030,encouraging innovation and substantially increasing the number of research and development workers per 1 million people and public and private research and development spending"
  - **9.b.** "Support domestic technology development, research and innovation in developing countries, including by ensuring a conducive policy environment for, inter alia, industrial diversification and value addition to commodities" This project is an extensive research with the clear intention of augmenting the knowledge in Artificial Intelligence. This knowledge is planned to be shared with the community with the code and the conclusions obtained.

- **Goal number 17: "Partnership for the goals"**
  - **17.8.** "Fully operationalize the technology bank and science, technology and innovation capacity-building mechanism for least developed countries by 2017 and enhance the use of enabling technology, in particular information and communications technology" If the solution is suitable, alliances with business and enterprises can be made, so innovation can be used in different environments.

# Bibliography


[1] Universal Robots. *Universal Robots UR3*. URL: https://www.universal-robots.com/es/productos/robot-ur3/.

[2] Universal Robots. *Cuales son las diferencias entre un cobot y un robot industrial*. URL: https://blog.universal-robots.com/es/cobots-vs-robots-industriales.

[3] Open Robotics. *About ROS*. URL: https://www.ros.org/about-ros/.

[4] Luis Cruz. *ROS (Robot Operating System) — Fundamentos*. URL: https://medium.com/@robtech.impaciente/ros-robot-operating-system-fundamentos-e92478c26e02.

[5] Open Robotics. *Core Components*. URL: https://www.ros.org/core-components/.

[6] Open Robotics. *Installing and Configuring Your ROS Environment*. URL: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment.

[7] PickNik Robotics. *Moving robots into the future*. URL: https://moveit.ros.org/.

[8] Open Robotics. *Integration with Other Libraries*. URL: https://www.ros.org/integration/.

[9] OpenCV. *About*. URL: https://opencv.org/about/.

[10] *From research to production*. URL: https://pytorch.org/.

[11] URL: https://ai.facebook.com/tools/pytorch/.

[12] Ray Johns. *PyTorch vs TensorFlow for Your Python Deep Learning Project*.

[13] Kaggle. *State of Data Science and Machine Learning 2020*. 2021. URL: https://www.kaggle.com/kaggle-survey-2020.

[14] *Lightning in 2 steps*. URL: https://pytorch-lightning.readthedocs.io/en/stable/new-project.html.

[15] Tensorflow. *TensorBoard: TensorFlow's visualization toolkit*. URL: https://www.tensorflow.org/tensorboard.

[16] The SciPy community. *What is NumPy?* URL: https://numpy.org/doc/stable/user/whatisnumpy.html.

[17] *scikit-learn.* URL: https://scikit-learn.org/stable/.

[18] *Pillow, Overview.* URL: https://pillow.readthedocs.io/en/stable/handbook/overview.html.

[19] Robotics Online. *Pick and Place Robots: What Are They Used For and How Do They Benefit Manufacturers?* URL: https://www.robotics.org/blog-article.cfm/Pick-and-Place-Robots-What-Are-They-Used-For-and-How-Do-They-Benefit-Manufacturers/88.

[20] Andreas ten Pas Marcus Gualtieri and Robert Platt. *Category Level Pick and Place Using Deep Reinforcement Learning.* URL: http://juxi.net/workshop/deep-learning-rss-2017/papers/Gualtieri.pdf.

[21] Bnejamin Joffe. *Pose estimation and bin picking for deformable products.* DOI: https://doi.org/10.1016/j.ifacol.2019.12.566. URL: https://www.sciencedirect.com/science/article/pii/S2405896319324851.

[22] Dimiter Dobrev. *A Definition of Artificial Intelligence.* 2004. URL: https://arxiv.org/pdf/1210.1568.pdf.

[23] Gartner. *5 Trends Drive the Gartner Hype Cycle for Emerging Technologies, 2020.* URL: https://www.gartner.com/smarterwithgartner/5-trends-drive-the-gartner-hype-cycle-for-emerging-technologies-2020/.

[24] Ana Sandoiu. *Artificial intelligence better than humans at spotting lung cancer.* URL: https://www.medicalnewstoday.com/articles/325223.

[25] Jessica Hamzelou. *AI system is better than human doctors at predicting breat cancer.* URL: https://www.newscientist.com/article/2228752-ai-system-is-better-than-human-doctors-at-predicting-breast-cancer/.

[26] SAS. *Computer Vision, what it is and why it matters.* URL: https://www.sas.com/es_es/insights/analytics/computer-vision.html.

[27] Ryan Holbrook. *What is Deep Learning?* URL: https://www.kaggle.com/ryanholbrook/a-single-neuron.

[28] Analytics Vidhya. *Learn about Convolutional Neural Networks (CNN) from Scratch.* URL: https://courses.analyticsvidhya.com/courses/convolutional-neural-networks-cnn-from-scratch?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch.

[29] *About ImageNet.* URL: http://www.image-net.org/about-overview.

[30] CS231. *Transfer Learning.* URL: https://cs231n.github.io/transfer-learning/.

[31] Pytorch. *Transfer Learning for computer vision.* URL: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html.

[32] Deepanshu Mehta. *State-of-the-Art Reinforcement Learning Algorithms.* URL: https://www.researchgate.net/publication/338396174_State-of-the-Art_Reinforcement_Learning_Algorithms.

[33] OpenAI. *Part 2: Kinds of RL Algorithms.* URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[34] Reinforcement Learning Working Group. URL: https://github.com/rlworkgroup/garage.

[35] Ray Team. URL: https://docs.ray.io/en/master/rllib.html.

[36] OpenAI. URL: https://github.com/openai/baselines.

[37] Deeplizard. *Reinforcement Learning - Goal Oriented Intelligence.* URL: https://deeplizard.com/learn/video/nyjbcRQ-uQ8.

[38] Deeplizard. *Markov Decision Processes (MDPs) - Structuring A Reinforcement Learning Problem.* URL: https://deeplizard.com/learn/video/my207WNoeyA.

[39] DeepLizard. *Policies And Value Functions - Good Actions For A Reinforcement Learning Agent.* URL: https://deeplizard.com/learn/video/eMxOGwbdqKY.

[40] Deeplizard. *Q-Learning Explained - A Reinforcement Learning Technique.* URL: https://deeplizard.com/learn/video/qhRNvCVVJaA.

[41] Deeplizard. *Q-Learning Explained - A Reinforcement Learning Technique.* URL: https://deeplizard.com/learn/video/qhRNvCVVJaA.

[42] Anaconda. *The State of Data Science 2020.* 2021. URL: https://www.anaconda.com/state-of-data-science-2020.

[43] Pier Paolo Ippolito. *Feature Extraction Techniques.* URL: https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be.

[44] Inc. Preferred Networks. *Bin-picking Robot Deep Learning.* URL: https://www.youtube.com/watch?v=ydh_AdWZflA&ab_channel=PreferredNetworks%2CInc..

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

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

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

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

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

[50] 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.

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

[52] 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.

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

[54] *Torchvision.models*. URL: https://pytorch.org/docs/stable/torchvision/models.html#torchvision-models.

[55] Department of Economic and United Nations Social Affairs. *The Sustainable Development Objetives*. URL: https://www.un.org/sustainabledevelopment/es/infrastructure/.

[56] 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.

[57] 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.