UNIVERSITÀ DI PARMA

UNIVERSITA' DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN
"MATEMATICA"

CICLO XXXV

A comprehensive analysis of vision deep learning methods for object detection and 6D pose estimation: Real-time applications.

Coordinatore:
Chiar.ma Prof.ssa Alessandra Lunardi

Tutore:
Chiar.mo Prof. Marko Bertogna

Correlatore:
Chiar.ma Prof. Giorgia Franchini

Dottorando: Davide Sapienza

Anni Accademici 2019/2020 – 2021/2022

UNIVERSITY OF PARMA

Abstract

Mathematics
Department of Mathematical, Physical and Computer Sciences

Doctor of Philosophy

A comprehensive analysis of vision deep learning methods for object detection and 6D pose estimation: Real-time applications.

by Davide SAPIENZA

The popularity of Artificial Intelligence (AI) systems is growing rapidly, both in academia and society. In recent years, advances in computer vision and machine learning have enabled AI systems to be applied to a variety of scenarios, such as autonomous driving, robotics, and augmented reality applications. An obstacle detection system allows a car to detect and avoid potential hazards, or to brake in time to prevent an accident. Augmented reality can assist a surgeon in finding the most efficient way to make an incision, leading to better outcomes for patients. Automation of industrial processes can help reduce the risk of on-the-job injuries, by reducing the amount of wear and tear work.These applications require the detection, identification and pose estimation of objects, to improve people's quality of life. In order to obtain a working system, many factors must be taken into account, including the choice of data in the learning process, the choice of the learning method, and the choice of hardware platforms. The current research focuses on examining various techniques to enhance accuracy, speed, and stability in two key applications: Object Detection and 6D Pose Estimation. This thesis will mainly delve into deep learning methods, which have led to breakthroughs in these fields.

i) We will analyze the difficulties and characteristics of embedded Object Detection methods in detail, focusing on latencies, throughput, accuracy, memory and power consumption. We will evaluate the impact of each of these factors on the performance of the object detection system.

ii) We will discuss the challenges and biases related to datasets and methods, as well as the possible solutions to address them. The importance of awareness of the inherent limitations of a given problem will be addressed.

iii) Finally, a real-world case study of Object Detection and 6D Pose Estimation in underwater environments is presented, highlighting the challenges, pitfalls, and best choices for this particular scenario. The results of the experiments, on both simulated and real-world scenarios, will demonstrate that the proposed solutions are reliable and effective in detecting objects and estimating their 6D pose.

The findings of this research could be used to improve accuracy and efficiency for 2D Object Detection and 6D Pose Estimation methods.

Contents

List of Abbreviations

SOTA	**S**tate **O**f **T**he **A**rt
RGB	**R**ed **G**reen **B**lue color
RGBD	**R**ed **G**reen **B**lue **D**epth color
2D	**2D**imensions
3D	**3D**imensions
6D	**6D**imensions
NN	**N**eural **N**etwork
ANN	**A**rtificial **N**eural **N**etwork
DNN	**N**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
OD	**O**bject **D**etection
ODCNN	**O**bject **D**etection **C**onvolutional **N**eural **N**etwork
BB	**B**ounding **B**oxes
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
IoU	**I**ntersection **o**ver **U**nion
AP	**A**verage **P**recision
mAP	**m**ean **A**verage **P**recision
FPS	**F**rame **P**er **S**econd
MR	**M**ean **R**ecall
SSD	**S**ingle-**S**hot **D**etectors
Yolo	**Y**ou **O**nly **L**ook **O**nce
ADD	**A**verage **D**istance to the Corresponding Model Point
ADI	**A**verage **D**istance to the Closest Model Point
VSD	**V**isible **S**urface **D**iscrepancy
COU	**C**omplement **O**over **U**nion
PnP	**P**erspective-**n**-**P**oint
RANSAC	**RAN**dom **SA**mple **C**onsensun
ICP	**I**terative **C**losest **P**oint
6DoF	**6** **D**egree **o**f **F**reedom
LM	**L**ine**M**od
AF-LM	**A**rUco **F**ree-**L**ine**M**od
CAD	**C**omputer **A**ided **D**esign
AAE	**A**ugmented **A**uto**e**ncoder
EP	**E**fficient**P**ose
GPU	**G**raphic **P**rocessing **U**unit
GPGPU	**G**eneral **P**urpose **G**raphic **P**rocessing **U**unit
FPGA	**F**ield-**P**rogrammable **G**ate **A**rray
ASIC	**A**pplication-**S**pecific **I**ntegrated **C**ircuit
PS	**P**rocessing **S**ystem

PL	**P**rogrammable **L**ogic
TRD	**T**arget **R**eference **D**esign
ROV	**R**emote-**O**perated **V**ehicle
USV	**U**nmanned **S**urface **V**ehicle
LRAUV	**L**ong-**R**ange **A**utonomous **U**nderwater **V**ehicle
AUV	**A**utonomous **U**nderwater **V**ehicle
CAMERA	**C**ontext-**A**ware **M**ix**E**d **R**e**A**lity
NOCS	**N**ormalized **O**bject **C**oordinate **S**pace

Chapter 1

Introduction

The technological revolution of the past few years has presented us with a world that was once only imaginable in science fiction. We now use electronic devices in almost every facet of our lives. The speed of progress is unprecedented.

In 1865, James Clerk Maxwell published his famous Maxwell's equations related to electromagnetism at the conference "A Dynamical Theory of the Electromagnetic Field". Soon after, Thomas Edison invented the incandescent light bulb and the radio, disputed between Nikola Tesla and Giuseppe Marconi, was developed in the late 1800s. The early 1900s saw the invention of the internal combustion engine, television, and other household appliances, yet it wasn't until the second half of the twentieth century, roughly 60-70 years ago, that these items became widely available.

The development of information technology began in 1944 with Alan Turing's invention of the switch-programmable digital computer, Colossum, which had no operating system. A few years later, in 1948, Claude Elwood Shannon introduced the concept of the "bit". The 1950s saw the emergence of various calculators, although they were always of considerable size. It wasn't until the 1970s that home computing became a reality, with the introduction of the IBM Personal computer in 1981, which had an Intel 8088 microprocessor running at a clock frequency of 4.77 MHz and a maximum of 64 kB of primary memory on the motherboard. In 1991, Tim Berners Lee invented the World Wide Web, and today we are already seeing the emergence of distributed computing, edge-cloud computing, robotics and Artificial Intelligence.

This rapid growth of technology has given rise to the new discipline of Computer Science, which seeks to meet the needs of the new subject area by developing new hardware and software technologies. Through high-level programming, we are able to abstract complex operations and manage the low-level operations that interface with the memory, processor and peripherals.

Computer Science has also had a major impact on other disciplines, particularly mathematics. Areas such as Machine Learning and Deep Learning would not have been possible without the developments in Computer Science. The relationship between the two is strong and ranges from the management of databases to the coding of algorithms to train artificial neural networks. All of these techniques are enabled by the technologies available today and the ability to process huge amounts of data in a very short time.

Artificial Intelligence First coined in 1956, Artificial Intelligence (AI) is a subfield of Computer Science that seeks to replicate human intelligence and behavior. AI can also be seen as a branch of Cognitive Science, as it focuses on understanding and reproducing intelligent mechanisms. This leads us to the question: what is intelligence? Alan Turing believed it was too difficult to give a definitive answer, so he proposed the Turing test instead. Despite this, the Turing test does not adequately

measure human-level Artificial Intelligence and is not sufficient to form an appropriate or useful criterion.

In order to create machines able of intelligent mechanisms, AI attempts simulation and extension of human intelligence through the use of artificial methodology and technology.

Our brain takes in various inputs, such as images, sounds, and smells, through our senses, and then processes them with cognitive systems. All of this data, along with the cognitive systems' intuitions, provides ample information. Although these tasks are completed in real-time, they involve a range of complex procedures that are difficult to replicate artificially.

Human intelligence is characterized by the ability to learn through experience. This has led to the development of a new field, known as Machine Learning (ML). Unlike traditional machines, which simply follow instructions given by humans, ML algorithms are designed to learn from data alone. A subfield of ML, known as deep learning, has proved to be particularly successful over the past decade. It is based on artificial neural networks, non-linear mathematical models inspired by the human brain and neurons, and is considered a form of parallel and distributed processing. Deep learning pipelines are numerical models that optimize each stage of the processing by searching for parameters that minimize the error computed by the training loss function.

The application of deep learning is vast and can be used in a variety of fields. It is used in computer vision and natural language processing, allowing machines to understand and interpret images and text. It can also be used for robotics, as robots can use deep learning algorithms to learn how to move and interact with their environment. Deep learning can also be used for autonomous vehicles, as it enables the vehicles to recognize and classify objects in their environment, as well as detect and avoid obstacles.

This thesis will explore the ability of machines to understand images. This includes tasks such as classification, localization and pose prediction, which are relatively simple for humans, but extremely complex for machines. The Szeliski book "Computer Vision: Algorithms and Applications" [92] is a comprehensive guide to the latest techniques and algorithms in computer vision. The book covers topics such as Convolutional Neural Networks (CNNs), which are artificial neural networks inspired by the convolution operation in mathematics. The idea of convolutional neural networks was popularized by LeCun, Bottou et al. (1998) [47], where the first CNN-based architecture for digit recognition was introduces. These networks are able to extract feature maps from an image, while also preserving spatial information. In 2015, Convolutional Neural Networks (CNNs) outperformed humans in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), for the first time in the history of the competition (Figure 5.40 in [92]). This marked a major milestone in the development of CNNs not only for image classification. This unexpected outcome, achieved for one of the most researched tasks, namely classification, was enabled by scientific advancements in this field as well as the abundance of data collected over time for the classification task. Deep neural networks rely heavily on data quality and quantity to reach increasingly high accuracy. The better the quality of the data, the more the expressive power of the trained model will increase, although the network will be unable to identify something it has not been trained to recognize. The greater the amount of data, the higher the probability that all cases of interest will be subjected to network training. Over the years, it is anticipated that for other research areas such as 6D pose estimation, as well as for classification, the

boost in accessible trainable data will also enhance the effectiveness of these methods in outperforming humans. In the near future, we will no longer be confined to utilizing single networks for single tasks, but as some techniques are now exploring, there will be the potential of having multi-task models that amalgamate multiple tasks that are managed separately at present.

2D object detection and 6D pose estimation Two main topics will be addressed in this thesis: 2D object detection and 6D pose estimation problems. The 2D object detection is a computer vision task that involves locating and identifying objects in a two-dimensional image. This task is widely used in a variety of applications, such as facial recognition to quickly identify individuals in footage from traffic cameras; anomalous detection to spot manufacturing defects in a production chain; and obstacle avoidance for autonomous vehicles, to detect pedestrians, bicycles, or any other obstacle. The 6D pose estimation problem is another type of computer vision task that involves estimating the 3D pose of an object in a 6-dimensional space. This is usually done by using a camera to capture an image of the object and, then, using a computer vision algorithm to estimate the object's position and orientation in the 6-dimensional space. 6D pose estimation is used in a variety of applications, such as in augmented reality for medical applications, allowing surgeons to simulate procedures in 3D virtual environments to search for the best course of action; and in robotics to assemble or disassemble objects that might be dangerous or tedious for humans to handle.

When considering these two topics, it is important to remember that the real-time embedded domain plays an important role in the application examples described above.

Real-time applications Real-time computing is a form of computing that emphasizes the completion of tasks within a specific timeframe. It is typically used in applications that require data to be processed immediately or within relatively short periods of time. Real-time programs must guarantee response within specified deadlines. Examples of real-time computing include military radar systems, air traffic control systems, and stock market trading systems. Real-time computing requires specialized hardware and programming techniques in order to produce timely results.

In Artificial Intelligence, real-time can be referred to as those deep learning techniques that need little time to execute and do not require supercomputers to execute. Real-time Artificial Intelligence techniques should be light and fast, taking up little memory space and requiring low computational powers. For example, an AI application on video processing can be profanely defined as real-time if it is able to run on smartphones without flickering. In AI applications, the term "real-time" does not have the same rigorous and formal meaning as it does in the real-time computing community. It should be more accurate to use the term "embedded" in this context.For example, the sub-discipline corresponding to 2D object detection is "real-time object detection", which specifies Frame-Per-Second (FPS) in addition to accuracy, but does not directly address the issue of time constraints. A sub-field of 2D object detection that is of particular interest is "real-time object detection". This research area places emphasis not only on accuracy, but also on latency, measured by frames per second (FPS). However, this does not address the issue of time constraints directly. Artificial Intelligence for embedded domains is applicable to many

areas, such as robotics, autonomous vehicles, industrial automation and surveillance systems.

Generally, these kinds of applications require running on embedded platforms. Embedded platforms are computer systems typically designed to be compact, low-power, and resource-efficient. Embedded platforms are designed to perform a specific task, rather than being a general-purpose computer for multiple tasks. They are used when performance constraints such as safety and usability must be met or simply to reduce cost.

The subfields of Computer Science concerning Artificial Intelligence for real-time embedded domains involve the design of neural networks to meet the time, memory and power constraints, as well as specific hardware and software solutions to keep up with innovation. For the design of a new DNN architecture, computer scientists and analysts, mathematicians, and other specialists must consider the number of layers, the number of neurons per layer, and many more hyperparameters of the network. In addition, software and electronic engineers, computer scientists, and other experts must design new hardware platforms and software solutions to accelerate the DNN solution on those platforms.

1.1 Contribution and Organization

This thesis is an exploration of the Artificial Intelligence (AI) and Deep Learning approaches to 2D object detection and 6D pose estimation problems. By exploring the current state-of-the-art algorithms and models, we aim to better understand the challenges associated with these tasks and propose novel domains. Although mainly NNs-related topics will be covered, the focus is always on embedded techniques and applications as well. The topics covered in this thesis are only a part of the activities carried out during the doctoral program.

In order to make the thesis fluent to read, the most interesting topics and those that have been most deeply explored in the past three years. However, more information on the other topics which have been covered during this period can be found in:

- A systematic assessment of embedded neural networks for object detection [99];
- All You Can Embed: Natural Language based Vehicle Retrieval with Spatio-Temporal Transformers [81];
- Deep Image Prior for medical image denoising, a study about parameter initialization [79];
- Deep learning-assisted analysis of automobiles handling performances [80];
- **[submitted]** Uncovering the Background-Induced bias in RGB based 6-DoF Object Pose Estimation;
- **[into submission]** Model-based Underwater Object 6D Pose Estimation.

In this thesis, the main contributions are organized as follows:

- a thorough survey of the literature on real-time object detection and 6D pose estimation methods is provided in Chapter 2. For both topics, the discussion includes an exploration of the problem statement, an explanation of metrics, an

investigation of literature datasets, and a presentation of neural network methods. Finally, a brief overview of the embedded platform solutions is given;

- an exhaustive study of two main axes is presented in the chapter 3. On the one hand, convolutional neural networks for object detection (ODCNN) and on the other hand, heterogeneous embedded platforms. For each branch under analysis, several insights related to floating-point data representation, accuracy and latency results, power consumption, and memory utilization are discussed. All to finally identify the best platform and network with the best trade-off.

- In Chapter 4, an intriguing first study is introduced under the big hat of the explainability of the 6D pose estimation method. Surprising results are presented for the most recent 6D pose estimation problem, showing a bias that plagues the best of the RGB-based 6D pose estimation methods and the most common and widely used dataset for this task.

- a novel domain and its corresponding application in 6D pose estimation is presented in Chapter 5: objects recovery in underwater environments. Various well-known and common problems, such as symmetries, occlusions, and self-occlusion, are addressed, and new ones are added to directly address real-world domain applications. The methodological choices made in this new domain have been based on multi-constraint lines, in an attempt to meet real-time limitations, low-level camera resolutions, poor degree of object detail, and poor visibility quality of the environment under investigation.

Conclusions and open problems are discussed in Chapter 6.

Chapter 2

Background

2D object detection and 6D pose estimation problems, with a particular focus on real-time scenarios, are the main topic covered in the thesis. This chapter presents an overview of these subjects, focusing on problem definition, datasets, metrics, and some state-of-the-art methods. Specifically, the 2D object detection problem is discussed in section 2.1 and an argumentation limited to real-time methods is presented. An overview of the 6D pose estimation solution, focused on RGB methods only, is presented in section 2.2. Finally, a description of some real-time embedded platforms is provided in section 2.3.

2.1 2D Object Detection

Object Detection (OD) is an important computer vision task consisting of detecting instances of objects in an image, Fig. 2.1. Detection means both identifying a specific class for each instance in the scene (e.g., cat, dog, person, car, ...) and locating that instance in the image. The goal of this research area is to develop a computational model that is able to answer the following question: "What and where are the objects located within the RGB frame?".

The importance of the Object Detection research stems from its use in various other computer vision tasks, such as Instance Segmentation, Object Tracking, Image Captioning, and even in 6D pose estimation problems.

2.1.1 Problem

The 2D Object Detection problem can be formalized as: given an input image I and a predefined list of object types or classes $C = \{c_1, c_2, ..., c_m\}$, where m is the number of classes, an algorithm A, also called object detection model, not only classifies the objects' type depicted in the image but also identifies each object's location. The algorithm hence should return i) a predicted class $\hat{c}_i$, and its corresponding score (or confidence score) s_i, for each depicted object $i = 1, 2, ..., n$ where n is the number of objects present in the scene, and ii) the location of each object in the form of bounding boxes (BBs) $\{B_1, B_2, ..., B_n\}$ (one per object) where $B_i = \{(x_1, y1), (x_2, y2)\}$ is the set containing bounding-box top-left and bottom-right coordinates, which encodes 2D translation and scale [77, 19]. The algorithm A can be defined as:

$$A(I, C) = \{(o, \hat{B}, \hat{c}, s) : o \in O, \hat{c} \in C, s \in (0, 1]\} \tag{2.1}$$

where O is the set of depicted objects in the image I, and $\hat{B}$ is the predicted bounding box of each object. The elements in the scene are not necessarily different objects, but they could also be different instances of the same object. In the following discussion, only the deep learning algorithms will be treated. Deep learning Object Detectors are

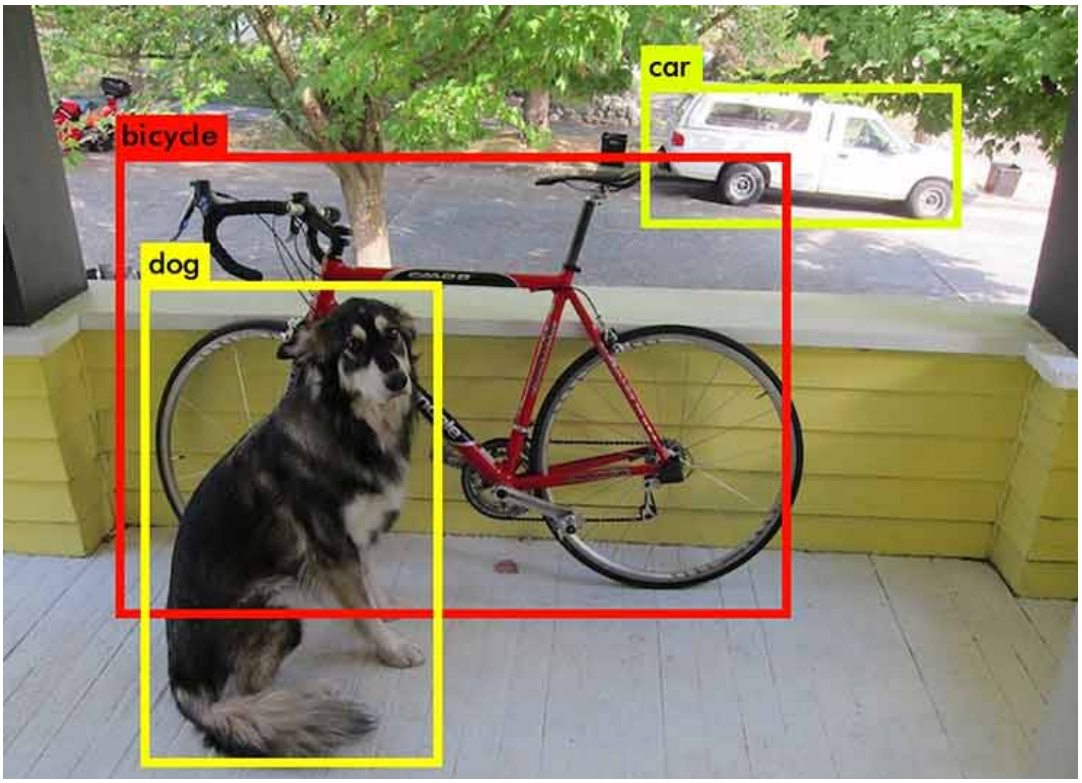


FIGURE 2.1: Object detection problem. [76]

divided into single-stage and two-stage algorithms, and typically both of them are composed by two principal elements: a feature extractor hereafter, backbone) and a detection head. The backbone is usually a CNN-based network that extracts the most prominent representations of a scene (from low to high-level features). Most backbones use pooling/convolution layers with strides to gradually reduce the size of feature maps and increase the receptive field of the network. Backbone networks are typically designed originally from image classification and pre-trained on existing datasets, as written in the following sections. They are fundamental for a better performance of the head procedure because their feature maps outputs are the detection head inputs. Then the head performs classification and regression to determine the label/class and location of the object instances.

2.1.2 Datasets

The success of deep learning object detectors can be attributed to the large Datasets available in literature used for training this kind of models. These Datasets are the same ones on which the methods evaluate various canonical benchmarks. Sample images of some datasets are shown in the Figure 2.2

The most common datasets used for the object detection task include MS COCO [55], PASCAL VOC [19], ImageNet [77], BDD100k [110], Waymo [87] [18].

The **Pascal VOC** dataset has 11k training images and more than 27k labeled objects for 20 different classes. This dataset today in addition to being used for classification and object detection tasks is also used for segmentation and action detection tasks.

FIGURE 2.2: Sample images of some of the most common datasets for object detection. On the top left is Pascal VOC, on the top right is COCO, on the bottom left is Imagenet, and on the bottom right is BDD100K.

The **ImageNet** dataset is particularly known as benchmark dataset for evaluating algorithm performances. This dataset was originally created for the classification task, and is now widely used to train the backbone networks of object detectors. The dataset size was scaled up to more than a million images consisting of 1000 object classification classes. 200 of these classes were hand-picked for object detection tasks, constituting more than 500k images. ImageNet also updated the evaluation metric by relaxing the IoU threshold to help include smaller object detection.

The **MS-COCO** dataset is one of the most challenging datasets available. It has 91 common objects found in their natural context which a 4-year-old human can easily recognize. It has more than two million instances and an average of 3.5 categories per image. Furthermore, it contains 7.7 instances per image, comfortably more than other popular datasets. MS COCO comprises images from varied viewpoints as well. It also introduced a more stringent method to measure the performance of detectors.

The **BDD100k** is the largest driving video dataset available. It has 100,000 videos for 10 different tasks to evaluate the latest advances in image recognition algorithms for autonomous driving. The dataset has geographic, environmental, and weather diversity, which makes it useful for training models that are more resilient to new conditions. This dataset is used for the traffic object detection task.

The **Waymo** dataset is composed of two different datasets: the Perception dataset [87] with high resolution sensor data and labels for 2,030 scenes, and the Motion dataset [18] with object trajectories and corresponding 3D maps for 103,354 scenes. The Perception dataset contains 2,030 segments of 20 seconds each, collected at 10Hz

		Actual class	
		Positive	Negative
Predicted class	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

TABLE 2.1: Confusion matrix.

(390,000 frames) in diverse conditions and geographical areas, while the Motion dataset contains 103,354 segments of 20 seconds each are mined to find interesting interactions, collected at 10Hz (over 20 million frames).

2.1.3 Metrics

A key aspect of deep learning methods is measuring their performances. The performances allow methods to be compared with each other, creating a ranking based on results and decreeing one method as better than another. Metrics are mathematical formulations used to measure the model's performance. They can be used to compare different models, as well as different versions of the same model by fine-tuning the artificial neural networks. This is the concept behind the strategy of exploring a field of research inherent to DNNs: the Neural Architecture Search (NAS). It is the process of automating architecture engineering in the neural networks: from the search space (connection weights, number of layers, number of neurons for layer and learning rules), to search strategy (generational evolutionary algorithms [33], to Randomly Wired Neural Networks [108]), and performance estimation strategy [27] [17].

To evaluate the performance of an object detector a lot of criteria are used.

Among the simplest metrics are precision and recall, defined as:

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$recall = \frac{TP}{TP + FN} \tag{2.3}$$

In these two formulations, precision can be described as the percentage of the True Positives over all observations, while the recall as the ratio of the True Positives over all ground truth. The confusion matrix with all possibilities is shown in Table 2.1.

Intersection over Union (IoU), on the other hand, is a specific metric for the object detection problem, evaluating location instead of classification, and is defined as the ratio of the overlap area to the union area between the ground truth and the estimated bounding box, defined as:

$$s_{IOU}(\hat{B}, \overline{B}) = area(\hat{B} \cap \overline{B}) / area(\hat{B} \cup \overline{B}) \tag{2.4}$$

where $\hat{B}$ and $\overline{B}$ are the predicted and ground truth bounding boxes respectively. A threshold $\gamma \in [0, 1]$ is used to determine if the detection is correct or not:

$$results = \begin{cases} TP & if \quad s_{IOU}(\hat{B}, \overline{B}) > \gamma \\ FP & otherwise \end{cases} \tag{2.5}$$

If the $s_{IOU}(\hat{B}, \overline{B})$ is more than the threshold γ, the predicted bounding box is considered correct and classified as True Positive (TP). Otherwise, it is incorrect and is

classified as False Positive (FP). In case of the object detector fails to detect an object present in the ground truth it is classified as False Negative (FN).

Another most common metric based on the above equations is the Average Precision (AP). AP wants to combine both classification and localization performance. To define the AP, one must first understand the precision-recall curve. This curve, denoted by $p(r)$, is constructed by plotting precision (on the y-axis) against recall (on the x-axis) at various thresholds γ. The threshold γ is used in Equation 2.1.3 to determine correct detections. Adjusting the threshold alters the number of TPs and FPs detected, and consequently affects precision and recall in different ways. A higher threshold will lead to higher precision but lower recall, while a lower threshold will lead to higher recall but lower precision. The precision-recall curve $p(r)$ often takes on a "zigzag" shape. The general definition of AP is the area under the precision-recall curve:

$$AP = \int_0^1 p(r)dr \tag{2.6}$$

Before computing AP for object detection, the recall-curve function is often smoothed to eliminate the zigzag pattern. The Pascal Visual Object Classes Challenge [20] introduced the interpolation of precision at each recall level:

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{2.7}$$

where $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$. At each recall level, the precision value takes the maximum precision value to the right of that recall level. As written in [20], this interpolation has the goal of reducing the impact of the "wiggles" in the precision-recall curve, due to small variations in the ranking of examples.

As precision and recall values, also AP values are between 0 and 1. Since only discrete values are available, AP is described as the mean precision at a set R of equally spaced recall levels, usually $\{0.0 : 0.1 : 1.0\}$ for a total of 11 values.

Finally, the AP could be approximated as follows:

$$AP(c, \Gamma, R) = \frac{1}{|\Gamma|}\frac{1}{|R|}\sum_{\gamma \in \Gamma}\sum_{r \in R} AP_r = \frac{1}{|\Gamma|}\frac{1}{|R|}\sum_{\gamma \in \Gamma}\sum_{r \in R} p_{interp}(r) = \frac{1}{|\Gamma|}\frac{1}{|R|}\sum_{\gamma \in \Gamma}\sum_{r \in R}\max_{\tilde{r} \geq r} p(\tilde{r}) \tag{2.8}$$

where Γ is the set of thresholds used in s_{IoU} and the common values on which is set are $\{0.50 : 0.05 : 0.95\}$, while R is the set of the discretised recall values. So, the AP is the mean over γ and the discretized recall r of the area under the precision-recall curve. The AP is calculated for a single class c, while the Mean Average Precision (mAP) is the mean over all classes of the AP:

$$mAP(C, \Gamma, R) = avg_{c \in C} AP(c, \Gamma, R) \tag{2.9}$$

The mean Average Precision (mAP) at 0.5 IoU to evaluate the performance of the models was introduced by the PASCAL VOC challenge [19]. Unlike the Pascal VOC and Imagenet, MS COCO challenge [55] calculates the IoU from 0.5 to 0.95 in steps of 0.05, then uses a combination of these 10 values as a final metric, also referred Average Precision (AP).

As discussed earlier, another important element of this treatment is the real-time component. For this reason, the generic network cannot be evaluated only in terms

of accuracy but must consider other aspects, such as the speed of response in the inference phase. These metrics depend not only on the network, but also on the combination of the network and the platform it is running on. For particular scenarios, it will not be enough to choose the most accurate or the fastest method, but the best trade-off between the two metrics must be found. In this sense, a useful metric is the Frame-per-second (FPS), which counts the number of test images processed in a single time unit (second):

$$FPS(A, \mathcal{I}, \Delta t) = \frac{\#I \text{ preprocessed by A} : I \in \mathcal{I}}{\Delta t} \tag{2.10}$$

where Δt is a time interval expressed in seconds, while $\mathcal{I}$ is the set of input images. FPS represents the average number of images that the algorithm A processes in a single second. Two other FPS variants are useful: the worst-case FPS and the best-case FPS which are the *min* and *max* of the single calculation of FPS formula respectively. In particular, in real-time applications, the worst-case is very important because it represents the method speed's minimum bound and the maximum response delay. Other important concepts, dealing with real-time applications, are those of latency and throughput. Latency, measured in milliseconds, is the amount of time it takes for a method to complete one iteration, consisting of three phases: preparing the data in pre-processing, performing the calculation in inference, and cleaning up the output in post-processing. Throughput is a measure of the rate at which a neural network can process data. It is usually expressed as the number of images processed per second or the number of images per second on which inference has been made. Higher throughput usually implies faster performance, but there are other factors such as model complexity and data set size that can also affect throughput. The trade-off between latency and throughput is an important factor in determining the overall performance of a deep neural network.

2.1.4 Neural Networks

In the Object Detection Convolutional Neural Network (ODCNNs) the backbone architecture is one of the most important components, as mentioned above. This kind of network aims to extract the most representative features from the input image. An object detector usually relies on one of the backbones discussed below and uses it to extract the most important representative features of an image; then the detector postposes a detection head to classify and regress the bounding boxes search. Among the most famous backbone, there are:

- **AlexNet** [45] is a CNN for classification task. It has 8 layers: five convolutional layers and three fully connected ones. At the end of the architecture, a softmax classifier is used to obtain a probability distribution over the number of classes. AlexNet won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) challenge in 2012 and breathed new life into Convolutional Neural Networks.

- **VGG** [85] is a CNN with a variable number of convolutional layers (from 8 to 16), followed by three fully connected layers and a softmax layer. Small convolutional filters are used. The authors showed that a set of small convolutional filters could capture a larger receptive field despite the reduction of network parameters, while maintaining high accuracy.

- **GoogleNet** [89] is a DNN with 22 layers based on multiple **Inception** modules [89] [90] [91]. Inception is a network that has multiple size filters at the same level. Input feature maps pass through these filters and are concatenated at the end, before being forwarded to the next layer. To regularize the gradient, in the intermediate layers, the network uses auxiliary classifiers. GoogleNet allows to have a sparsely connected architecture instead a fully connected one, reducing the number of parameters and still achieving state-of-the-art performances.

- **ResNet** [24] introduced the skip connections to mitigate performance decay due to continued networks growth. Resnet has typically 34 layers with large (7x7) convolutional filters followed by 16 bottleneck modules and a fully connected layer. The bottleneck module has typically two convolutional layers with a 3x3 filter. Other versions of ResNet exist and they have a higher number of layers (ResNet101, ResNet152). In ResNetV2 [25], batch normalization and ReLU layers are added in the blocks.

- **ResNeXts** [107] is an elegant and concise network based on ResNet architecture. The ResNeXT blocks are inspired by both VGG/ResNet-like block stacks and the inception modules.

- **EfficientNet** [93] is a simple and efficient neural network founded by a Neural Architecture Search (NAS) technique. NAS is a research field with the goal of iteratively finding the optimal architecture of a network without extensive trainings. In EfficientNet the authors proposed a compound coefficient that can uniformly scale the network depth, the network resolution increase, and the model scales.

- **Hourglass** [65] is a CNN born for the human pose estimation task. The architecture is based on the hourglass module, a set of convolutional and max-pooling layers combined in a symmetrical way. At each max pooling step, the network branches off and applies more convolutions at the original pre-pooled resolution. After reaching the lowest resolution, the network begins the top-down sequence of upsampling and combination of features across scales. Good network performance comes from the repetition of bottom-up and top-down processing.

- **DLA** backbone [111] is a CNN for classification that augments ResNet and ResNeXT with the Deep Layer Aggregation. This aggregation follows two different strategies: Iterative Deep Aggregation (IDA) and Hierarchical Deep Aggregation (HDA). DLA is a staged network, which groups blocks by spatial resolution, with residual connections within each block. There is a total of six stages, and at the end of each stage, the resolution is halved. The first stage maintains the input resolution and the last stage is 32x downsampled. By a global average pooling the feature maps are collapsed and then linearly scored. At the end of the network, a softmax is applied.

The detectors can be divided in two-stage and single-stage detectors. In the first case, the networks have a distinct module to generate region proposals. First, they try to find a number of object proposals and, secondly, classify and localize them. In the second case, for single-stage ones, the networks try to classify and localize the objects in a single shot using dense sampling. To localize objects they use predefined keypoints/boxes of several scales and aspect ratios.

The two-stage detectors typically have a more complex architecture and require more computational resources to compute than the single-stage detectors. For this reason, single-shot ODCNNs are usually used in real-time applications.

- two-stage detectors: R-CNN, Fast R-CNN, SPP-net, Faster R-CNN, FPN, R-FCN, Mask R-CNN, DetectoRS;
- single stage detectors: YOLO, SSD, YoloV2 and YOLO9000, RetinaNet, YoloV3, EfficientDet, YoloV4, CenterNet, PPYolo, YoloV5, YoloV6, YoloV7.

In the following, only some 2D detectors are summarised, some of which will be discussed in Chapter 3.

YoloV3 YoloV3 [75] is a one-stage detector that divides images into grid cells and predicts bounding boxes using dimension clusters as anchor boxes. It adopts independent logistic classifiers to output an object score for each BB. The BBs are predicted at three different scales through extracting features from these scales. YoloV3 uses a backbone network, named Darknet-53, for performing feature extraction, which is a residual network with 53 convolutional layers. Due to the introduction of Darknet-53 and multi-scale feature maps, YoloV3 achieves great speed improvement and improves the detection accuracy of small-sized objects when compared with YoloV2 [74].

YoloV3-tiny YoloV3-tiny is a lighter version of YoloV3. It uses the same concepts (independent logistic and anchors), but the backbone is composed of only 10 convolutional layers and there are only two scales (therefore 6 anchors rather than 9). Due to its compactness, it achieves high inference speed, although it performs worse than its full version.

Mobilenetv2-SSDLite MobileNetv2 [78] is an efficient CNN model with depthwise convolution layers that have fewer weights compared with ordinary convolution layers. It is one of the most used models for embedded systems because it is lightweight, and it can achieve high FPS also on mobile devices. SSD [58] is a one-stage detector which divides images into grid cells, and for each grid cell, uses a pre-generated set of anchors with multiple scales and aspect-ratios to discretize the output space of BBs. SSD predicts objects on multiple feature maps, and each of them is responsible for detecting a certain scale of objects, according to its receptive fields.

CenterNet CenterNet [112] proposes modelling an object as a single point. It uses key point estimation to find center points and regresses all other object properties including 3D location, pose orientation, and size. In this model, an image is fed to a CNN which generates a heatmap, whose maximum values represent the centers of the objects in the image. The objects' size and pose are regressed from features of the image at the center location. CenterNet was tested with four different backbones, i.e. ResNet18, ResNet101, DLA34 and Hourglass, substituting the convolutional layers with deformable convolutional layers v2 [113]. Deformable convolutional networks (DCN) [13] are detectors able to adapt to the geometric variations of objects. Regular convolutional networks can only focus on features of fixed square size (according to the kernel), thus the receptive field does not properly cover each pixel of a target object to represent it. The DCN produces a deformable kernel and the offset from the initial convolution kernel (of fixed size) is learned during training.

EfficientDet EfficientDet [94] is a scalable and efficient single-shot object detection network. Tan et al. introduce Bi-directional FPNs (BiFPNs) based on the Feature Pyramid Networks (FPNs)[54], which use inherent multi-scale hierarchical pyramids of feature maps in order to detect objects at different scales in different scenes. In BiFPN the authors overcome the one-way information flow limitation allowing top-down and bottom-up paths between feature network layers. EfficientDet has EfficientNet [93] as the backbone network and BiFPN as the feature network.

YoloV4 YoloV4 [4] has been introduced by Bochkovskiy et al. as an evolution of YoloV3. The authors changed both the architecture, the training and the data augmentation methods to achieve better accuracy performance, but still having real-time performances. YoloV4 has CSPDarknet53 as backbone, with 29 convolutional layers, a spatial pyramid pooling (SPP) additional module [26], Path Aggregation Network (PANet) neck [57], and YoloV3 (anchor based) head at three scales.

CSPDarknet53 is their developed feature extractor. The SPP is a layer that removes the fixed-size constraint of the network. The SPP layer pools the features (using a maxpool in this case) and generates fixed-length outputs, which are then fed into the fully-connected layers (or other classifiers). On the other hand, the PANet is used to perform parameter aggregation from different backbone levels for different detector levels, instead of the FPN used in YoloV3. Moreover, besides Leaky ReLu and ReLu, in YoloV4 the mish activation function[62] is also widely used.

YoloV4-tiny The lighter version of YoloV4 has a backbone composed of 15 convolutional layers and there are only two scales. A particular feature of this network is the grouped route layer: it does not refer to a full layer output, but only to part of it, in this case the second half.

In Chapter 3 a systematic assessment of these detectors is discussed. The compared methods are evaluating on latency, accuracy, and power consumption on several embedded platforms.

PPYolo PPYolo [59], proposed in 2020, uses YoloV3 as head. The backbone is ResNet50-vd [24] and some convolutional layers are replaced with deformable ones [13]. Its performance, albeit slightly, exceeds that of yoloV4.

YoloV5 YoloV5, released in 2021, is based on YoloV3. It is a custom implementation developed on pytorch instead of darknet and does not belong to the original authors of Yolo. Despite this, it achieves better performance than YoloV4 and is available in different versions to achieve the desired trade-off between accuracy and latency. There is no published paper for YoloV5.

Yolov7 YoloV7 [101], released in July 2022, is proposed by authors of the Yolo series. YoloV7 supports instance segmentation, classification, object detection, and pose estimation. The method is based on Efficient Layer Aggregation Network (ELAN) [102]. ELAN uses expand, shuffle, and merge cardinality to improve the model learning ability without destroying gradient flow paths. YoloV7 proposes a modified ELAN, the Extended ELAN. There is also a lighter version of YoloV7, namely YoloV7-tiny, which achieves lower accuracy but has very high throughput.

YoloV6 YoloV6, as well as YoloV5, is not part of the official Yolo series. Li et al. [49], developed a one-stage object detector in different versions. The authors differentiate the adopted backbone for light or large models, using RepBlock [15] and CSPStackRep Block [103] respectively. YoloV6 adopts PAN topology [57] following YoloV4 and YoloV5. YoloV6 adopts the anchor point-based paradigm [96] [22] in order to be an anchor-free detector. YoloV6, counterintuitively to the name, is the most recent network among those analyzed that has been released (September 2022).

2.2 6D Pose Estimation

The 6D-pose estimation problem differs from the 2D object detection one in terms of the desired model output.

Whereas in the case of a 2D object detection problem it is sufficient to identify the object only in its two (x, y) coordinates, in 6D pose estimation it is necessary to provide the pose of the object itself, described by its 6 degrees of freedom. The axes x, y, z describe the position within the image, also called translation, while $\delta x, \delta y, \delta z$ describe the angles of the three axes, which together explain the rotation. For the 6d-pose estimation problem, however, it is first necessary to distinguish the methods according to the type of data used. RGB images are one of the most widely used data types in computer vision applications and thus also in the 6D-pose estimation problem. In literature, many 6D-pose methods have been developed for RGB input data only, and this direction is also the most challenging. In some other cases, methods are proposed that also use depth information (RGBD). This type of data is produced by an RGBD camera, a type of depth camera that provides depth (D) and color (RGB) data in the same output. Depth information can be obtained from the depth map/image, created by a 3D depth sensor. These sensors are usually time-of-flight or other sensors, such as STEREO cameras, LIDAR, etc. The depth is a single-channel image, the same size as the RGB image, whose values correspond to the distance of each point/pixel represented in the image from the sensor, the reference camera.

RGB methods have a number of advantages, such as the use of commonly available data types, no need for special hardware sensors, and greater compatibility with existing applications. However, their accuracy is lower than that of methods using depth data. By contrast, RGBD image-based methods have been demonstrated to be more effective. The depth information is essential to accurately and easily predict the third component of translation.

In order to compare the different models in the literature, a distinction must first be made between these two scenarios. Nevertheless, many existing RGB image-based methods also have the option of integrating depth data. In these cases, this is referred to as refinement, an operation that follows 6d pose estimation phase and is performed at a non-zero cost to achieve better performances. Then there are also stand-alone refinement methods in the literature, and native methods based on RGBD images. In this discussion, only RGB-based methods will be covered. As mentioned before the focus is on those methods for real-time applications.

In the 6D pose estimation problem, several metrics are used to calculate the performance of the models, which will be explained below. Common datasets in the literature are then described. Finally, an overview of the best performing methods for only the two scenarios of end-to-end and two-stage methods based on RGB images are provided.

2.2.1 Problem

To address the problem of 6D pose estimation, it is necessary to give a mathematical formulation of it. Let be m a generic object, it can be represented by a model $\mathcal{M}$, which is typically a mesh given by a set of points in $\mathbb{R}^3$ and a set of triangles. Every element $\mathbf{x}_m$ of the model $\mathcal{M}$ is a specific 3D point expressed in the model coordinate system. In the 6d-pose estimation problem, a pose of a 3D object is described by:

$$\mathbf{P} = [\mathbf{R}, \mathbf{t}; \mathbf{0}, 1], \mathbf{P} \in \mathbb{R}^{4x4} \tag{2.11}$$

where $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix, $\mathbf{t} \in \mathbb{R}^{3\times1}$ is the translation vector. The matrix $\mathbf{P}$ transforms the point $\mathbf{x}_m$, expressed in its coordinate system $\mathbf{S_m}$, in a new 3D point $\mathbf{x}_c$ with a new coordinate system $\mathbf{S_c}$, the camera coordinate system. In other words, applying the rotation and translation on the 3D point $\mathbf{x}_m$, a new 3D point $\mathbf{x}_c$ is obtained:

$$\mathbf{P}[\mathbf{x}_m; \mathbf{1}] = \mathbf{R}\mathbf{x}_m + \mathbf{t} = [\mathbf{x}_c; \mathbf{1}] \tag{2.12}$$

In a generic 6D-pose estimation task more than one object can be considered. Let be $M = \{m_i | i \in \mathbb{N}, \quad i \leq n\}$ where n is the number of objects, every object is represented by a model $\mathcal{M}_i$ and an algorithm A has to estimate the pose of some of those objects grouped in the subset $\hat{M}$, with $\hat{M} \subseteq M$ and $\hat{M} \neq \emptyset$.

Let be $\mathbf{D}$ a Dataset defined as:

$$\mathbf{D} = \{f_j | j \in \mathbb{N}, j \leq |\mathbf{D}|\} \tag{2.13}$$

where f_j is a single image RGB or RGBD of the dataset, every input image f_j can depict more than one object m_i and more than one instance of the same object m_i: m_i^k with $k = 1, 2, ..K$, where K is the number of occurrences of the object m_i in the image f_j. Each object instance depicted in the images has the same model $\mathcal{M}_i$ but differs from 6D pose to 6D pose $\mathbf{P}_i^k$. Based on the object pose given by $\mathbf{P}$, only part of the object m_i is visible, some or some parts of the object surface, and it defines the object view $v_{i,j}$ of the model m_i in the image f_j. Multiple instances of the same object and single instances of other objects can be managed independently. In that case, given an image f_j, a set of generic object instances can be defined: $O = \{o_1, o_2, ..., o_l\}$. The algorithm A for each image f_j will have to provide a set of pose predictions, one for each object depicted in the image through the following structure:

$$E_{f_j} = \{(o_h, \hat{\mathbf{P}}_h, s_h) | h \in \mathbb{N}, \quad h \leq |O|\} \tag{2.14}$$

where $\hat{\mathbf{P}}_h$ is the estimated 6D pose of an object instance $o_h \in O$ depicted in the image with a confidence $s_h \in (0, 1]$. In the same way, for each image the 6D-pose ground truth can be defined as:

$$GT_{f_j} = \{(o_g, \overline{\mathbf{P}}_g) | g \in \mathbb{N}, \quad g \leq |O|\} \tag{2.15}$$

Given these formulations, two different types of problems can be identified. In the first one, called 6D Localization Problem, the size of the output is fixed with respect to the dataset and the algorithm has the $|O|$ in input. Here $|E_{f_j}| = |GT_{f_j}|$. In the second one instead, the output size depends on the estimator A and the number of the depicted objects in the image is not known a priori. In that case it is called 6D Detection Problem.

There are several ways to represent rotation matrices in 6D pose estimation problems. These include using Euler angles, quaternions, axis-angle representations, and rotation matrices. Each of these representations has its own advantages and disadvantages depending on the application. Additionally, the rotation matrix can be decomposed into a set of orthogonal basis vectors, which can also be used to represent the 6D pose. Quaternions are composed of four elements, each representing an axis of rotation. They are usually represented as four-dimensional vectors, with the elements being the x, y, z, and w components of the quaternion. The w component is used to represent the angle of rotation. Quaternions can be converted to and from rotation matrices and can be combined to form a single rotation matrix.

2.2.2 Datasets

As mentioned earlier, DNN methods for object pose estimation can be divided into two subsets, RGB-based and RGBD-based methods. For both, typically classified as supervised methods, labeled data, which are both bounding boxes (2D or 3D) and the counterpart of pose information, must be provided in the training phase. This means that for each object represented in the scene, translation and rotation must be provided. This information is not present in the known datasets for the object detection task, but needs specific datasets. Sample images of some datasets are shown in the Figure 2.3

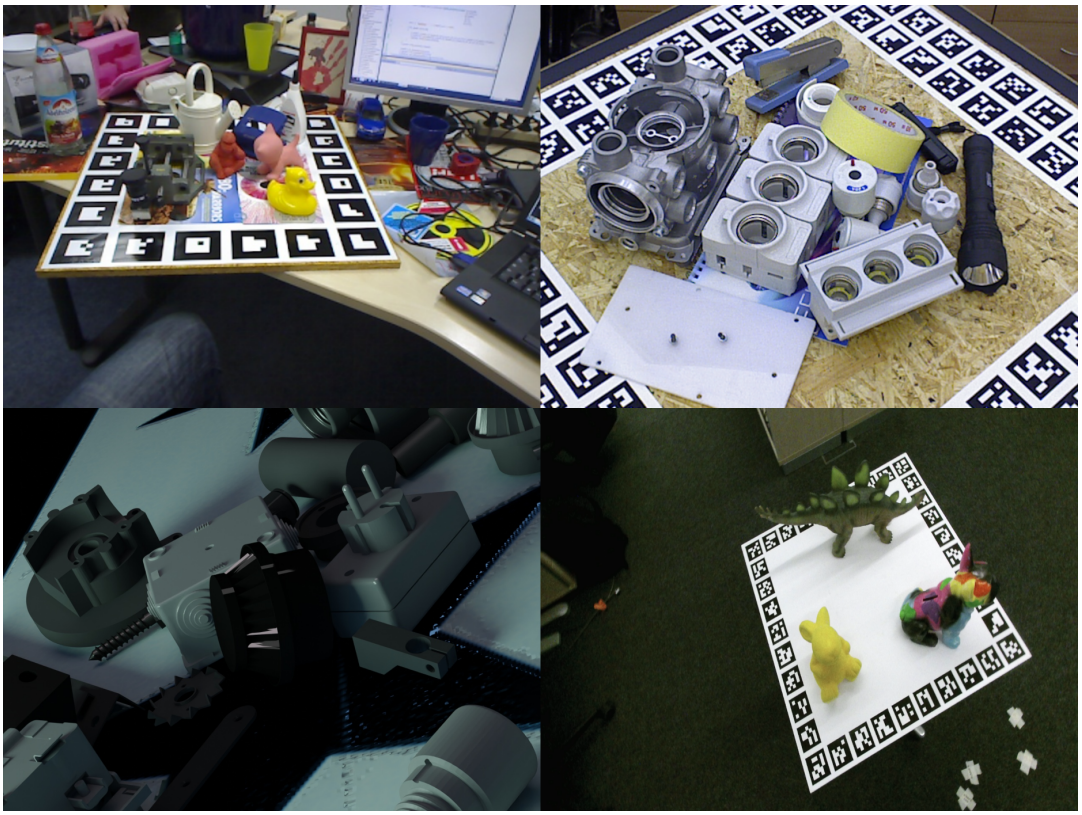


FIGURE 2.3: Sample images of some of the most common datasets for 6D pose estimation. On the top left is LineMod, on the top right is T-LESS, on the bottom left is MVTec ITOD, and on the bottom right is HomebrewedDB.

LineMod The most commonly used dataset for training and evaluation 6D pose estimation methods is LineMod [29]. LineMod is a dataset consisting of real images with 15 classes or object models, acquired from different views. Each class has ∼1200/1300 RGB, DEPTH, and MASK images for a total of 18273 test images and 19695 training images. For each object class, ground truth 6D pose labels are provided only for the target object, which is placed around the center of a custom-made work plane and surrounded by other cluttering objects that cause only mild occlusion. The target object is easily distinguishable, taking on typical sizes and colors, different from those of surrounding objects. The working plane consists of a chessboard-like structure delineated by custom-made ArUco markers, used to retrieve the ground truth labels. The ground truth object pose is retrieved by deploying geometric algorithms which use markers to recover first the board's pose in camera coordinates, and, successively, the object pose in the same coordinate system.

LineMod-Occluded Another version of LineMod, called LineMod-Occluded [5] tries to address the occlusion limitation of the previous dataset. In this case the authors added an extra ground truth, extra information about occluded objects present in the scene. The ground truth annotations are introduced for all modeled objects in one of the test sets, incorporating various levels of occlusion, resulting in a more challenging pose estimation task.

T-LESS Another dataset is T-LESS [32] created for industry-relevant objects, which lack texture or discernible color. That dataset has 30 classes, the training set is composed of 37584 real images and 76860 rendered images, while the test set is composed by 10080 real images for each of the 3 sensors. The real images come from 20 RGB-D scenes that were recorded through three synchronized cameras. The sensors are i) Primesense CARMINE 1.09 (a structured-light RGB-D sensor) ii) Microsoft Kinect v2 (a time-of-flight RGB-D sensor) iii) Canon IXUS 950 IS (a high-resolution RGB camera), and they are synchronized. The objects featured in the dataset present some symmetries and mutual similarities, with some being a combination of multiple objects.

HomebrewedDB Structurally similar to T-LESS, HomebrewedDB [42] covers a wider range of objects and provides more challenging occlusions. It consists of 33 highly accurate 3D models of toys, household objects, and low-textured industrial objects of varying sizes, along with 13 sequences containing 1340 frames filmed with two RGB-D sensors. The scenes range from simple (three objects on a plain background) to complex (highly occluded with eight objects and extensive clutter). Interestingly, a chessboard-like pattern similar to the one used in LineMod is clearly visible also in HomebrewedDB.

MVTec ITODD MVTec ITODD [16] contains industry-relevant objects, containing also objects with reflective surfaces.

HOPE NVIDIA HOPE (Household Objects for Pose Estimation) [97] introduced a new dataset of toy grocery objects. The annotations for this dataset were obtained manually, through the identification of point correspondences between images and 3D textured object models. During the acquisition phase, ten different environments, with five object arrangements/camera poses per environment, were used. These 50 different scenes exhibit a wide variety of backgrounds, clutter, poses, and lighting.

To provide additional clutter and partial occlusion, objects are also placed in other containers, such as bags or boxes. Of significance in the scope of this work, the dataset is advantageous as it does not utilize markers or ArUco markers during acquisition. Moreover, the different environments permit to better generalize. In total, the dataset contains 50 unique scenes, 238 images and 914 object poses. Once set the camera and the object position, some light effects are applied, in order to have more images with little differences in shadows and change colors, thereby resulting in more static images that did not need to be annotated.

2.2.3 Metrics

In order to determine the correctness of an algorithm A some specific metrics have been introduced. To simplify the notation, let be considered a generic input image I with a single depicted object instance o, the estimation can be defined as $(o, \hat{\mathbf{P}}, s)$ while the ground truth is $(o, \overline{\mathbf{P}})$. An estimation is considered correct with respect to the ground truth if the error $e(\overline{\mathbf{P}}, \hat{\mathbf{P}}, \mathcal{M}, I) \leq \theta$ where θ is a threshold of a pose error function e.

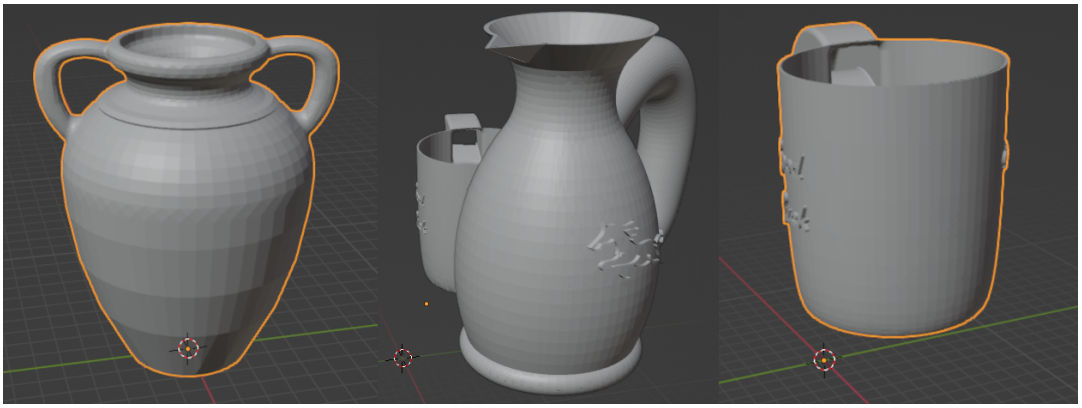


FIGURE 2.4: Examples of well-known problems in 6d pose estimation: symmetry, occlusion and self-occlusion.

The threshold permits a definition of a set of matchable estimated poses as correct with respect to the ground truth, but the threshold itself is not sufficient for some specific scenarios. These scenarios can be grouped into the indistinguishable pose problem. It is referred to some objects that are ambiguous, for example symmetric, occluded or self-occluded objects, such as a cup, bowl, box, glue, and so on (Figure 2.4). These objects, from a specific point of view and a subset of rotation and translation pairs, are invariant with respect to a specific object axis or more than one axes. In particular, let be $\mathcal{M}$ the model of an ambiguous object, the set of poses that are ϵ-indistinguishable from pose $\mathbf{P}$ can be defined as:

$$\mathbf{P}_{\mathcal{M},I,\epsilon} = \{\mathbf{P}^{'} : d(v_I, v_I^{'}) \leq \epsilon, \quad v_I = \psi(\mathbf{P}, \mathcal{M}), v_I^{'} = \psi(\mathbf{P}^{'}, \mathcal{M})\} \tag{2.16}$$

The visible portions of model surface $\mathcal{M}$ in image I are represented by v_I and $v_I^{'}$, which are obtained by applying the function ψ to two distinct poses, $\mathbf{P}$ and $\mathbf{P}^{'}$, respectively. The distance function d and tolerance ϵ are used to determine which poses in $\mathbf{P}_{\mathcal{M}_i,I,\epsilon}$ produce the same visible object view, up to the specified tolerance. For instance, when considering a bottle and restricting rotations to its longitudinal axis, different poses $\mathbf{P}$ may produce distinct orientations, but the resulting visible object view remains unchanged.

In that scenario, the pose error $e(\overline{\mathbf{P}}, \hat{\mathbf{P}}, \mathcal{M}, I)$ should consider it into account and the threshold itself is not sufficient. For this reason, in the following, some metrics will be investigated to address this problem.

Hinterstoisser et al. [29] proposed two different pose error functions: the Average Distance to the Corresponding Model Point (ADD) and the Average Distance to the Closest Model Point (ADI). These are the most widely used. For one 3D point of the object model $\mathcal{M}$, ADD error compares the resulting 3D point given from the estimation with its exactly corresponding one given from the ground truth:

$$e_{ADD}(\hat{\mathbf{P}}, \overline{\mathbf{P}}, \mathcal{M}) = avg_{x \in \mathcal{M}} \|\hat{\mathbf{P}}\mathbf{x} - \overline{\mathbf{P}}\mathbf{x}\| \tag{2.17}$$

On the other hand, to overcome the problem of indistinguishable views the authors proposed the ADI error:

$$e_{ADI}(\hat{\mathbf{P}}, \overline{\mathbf{P}}, \mathcal{M}) = avg_{x_i \in \mathcal{M}} min_{x_j \in \mathcal{M}} \|\hat{\mathbf{P}}\mathbf{x_i} - \overline{\mathbf{P}}\mathbf{x}_j\| \tag{2.18}$$

This error is more permissive than the previous one. In this case, it is possible to select two different 3D points used for the estimation pose and the ground truth one in order to evaluate the correctness of the algorithm when the model is symmetric, occluded, or self-occluded.

Hodan et al. [31] introduced a new error only over the visible part of an object model: The Visible Surface Discrepancy (VSD). In order to define VSD, one must first introduce the concept of a visibility mask. This mask comprises pixels where the surface of $\mathcal{M}$ is either in front of the scene surface or at most δ behind it. The visibility mask $\overline{V}$ is obtained by intersecting valid image pixels X_I with valid object pixels $\overline{X}$. The difference between the distance images obtained by rendering the model with the ground truth pose $\overline{D}$ and the distance image of the test scene D_I must not exceed a threshold of δ. The formal definition is as follows:

$$\overline{V} = p : p \in X_I \cap \overline{X} \wedge \overline{D}(p) - D_I(p) \leq \delta \tag{2.19}$$

When the predicted pose is used, the corresponding visibility mask $\hat{V}$ could be quite different. In this case, instead of considering the valid object pixels $\overline{X}$ given by the ground truth, the valid object pixel $\hat{X}$ given by the prediction will be considered. Similarly, instead of considering the distance images obtained by rendering the model with the ground truth pose $\overline{D}$, the distance images given by the predicted pose $\hat{D}$ will be used. An additional condition is required for $\hat{V}$ to ensure that the visible surface of the sought object does not occlude the object model surface: all object pixels included in $\overline{V}$ are added to $\hat{V}$, regardless of the surface distance at these pixels:

$$\hat{V} = p : (p \in X_I \cap \hat{X} \wedge \hat{D}(p) - D_I(p) \leq \delta) \vee p \in \overline{V} \cap \hat{X} \tag{2.20}$$

The Visible Surface Discrepancy (VSD) can be defined as the average of the matching costs for each pixel p present in both visibility masks:

$$e_{VSD}(\hat{\mathbf{P}}, \overline{\mathbf{P}}, \mathcal{M}, I, \delta, \tau) = avg_{p \in \hat{V} \cup \overline{V}} \begin{cases} d/\tau & if p \in \hat{V} \cap \overline{V} \wedge d < \tau \\ 1 & otherwise \end{cases} \tag{2.21}$$

The matching cost is determined by the following criteria: if p exists in the intersection of the two visibility masks and the distance d between the corresponding

points is less than the misalignment tolerance τ, then the error for that pixel is d/τ. If this condition is not met, the matching cost is set to the maximum value of 1. The distance d is computed as $|\hat{D}(p) - \overline{D}(p)|$, where $\hat{D}$ and $\overline{D}$ are the previously defined distance images.

Another popular pose error function is the Complement over Union (e_{COU}) [19]. This cost function is used for detection and segmentation methods in 2D domain:

$$e_{COU}(\hat{B}, \overline{B}) = 1 - s_{IOU}(\hat{B}, \overline{B}) = 1 - area(\hat{B} \cap \overline{B}) / area(\hat{B} \cup \overline{B}) \tag{2.22}$$

where s_{IOU} is the Intersection over Union introduced in [19] and represents the detection accuracy in 2D domain. Depending on the task, $\hat{B}$ and $\overline{B}$ can be rectangular regions (given by bounding boxes) or segmentation masks, and $\hat{B}$ (correspondent to $\hat{\mathbf{P}}$) and $\overline{B}$ (correspondent to $\overline{\mathbf{P}}$) are the estimated and ground truth 2D region respectively. In the 6D scenarios, the 2D regions are obtained by projection of the object model $\mathcal{M}$ in the two poses, the estimated and ground truth ones. This error function addresses the ambiguity problem and it is ambiguity invariant, but since it works with the projection of the model, it provides only weak information about the fitness of the object surface alignment. With respect to the measuring pose errors, also performance scores are used. In that case, Hinterstoisser et al. [29] suggest the Mean Recall (MR) and Mean Average Precision (MAP) to measure performance in the 6D pose estimation problem. In particular, the MR is:

$$MR = avg_{o \in O} \frac{\sum_I |\{(o', \hat{\mathbf{P}}, s) \in E_I^c : o' = o\}|}{\sum_I |\{(o', \overline{\mathbf{P}}) \in G_I : o' = o\}|} \tag{2.23}$$

where I is an input image, O is the set of objects depicted in I, $(o', \hat{\mathbf{P}}, s)$ is the output of the algorithm A and $(o', \overline{\mathbf{P}})$ is the ground truth.

On the other hand, the MAP, calculated as the mean of the Average Precision (AP) for each object is:

$$MAP = avg_{o \in O} avg_{r \in S_o} \frac{\sum_I |\{(o', \hat{\mathbf{P}}, s) \in E_I^c : o' = o, r \leq s\}|}{\sum_I |\{(o', \hat{\mathbf{P}}, s) \in E_I : o' = o, r \leq s\}|} \tag{2.24}$$

where S_o is the estimation confidence values set that are considered correct. The Precision-Recall curve is effectively described by the AP rate.

Several metrics have been presented to summarize the heterogeneity of the scores and errors used in the literature. The e_{ADD} and e_{ADI} are the most widely used cost function, one for the pose-distinguishable objects and one for the pose-ambiguity ones. However, these two pose error functions are not comparable to each other because e_{ADI} produces relatively small errors even for distinguishable views and is therefore more permissive than e_{ADD}. This is because in e_{ADI} the many-to-one correspondence of vertices, established by finding the nearest vertex, is easier to obtain. On the other hand, the e_{VSD} error is calculated over the only visible part of the object's surface and it works for both symmetric and asymmetric objects. It is able to treat the ϵ-indistinguishable poses of an object because is Ambiguity-invariant: those poses are considered approximately equivalent.

2.2.4 Neural Networks

For 6D-pose estimation methods, a first distinction that can be made is between RGB-based and RGBD-based methods. For the color-based approaches, only the three color channels are used to compute the pose estimation, while in the depth-based ones also the depth information is used to improve the performances and in particular the scale estimation of the object's pose. Section 2.2 briefly discussed the pros and cons of RGBD image-based methods versus RGB image-based ones. While the former yield better performance, they require dedicated hardware and datasets, whereas the latter provide greater flexibility and reduce the need for specialized equipment and datasets. Additionally, Chapter 5 will show that in certain scenarios, acquiring depth information may not be possible due to environmental limitations and noise inherent in the use case. Consequently, this discussion will focus solely on RGB image-based methods, which pose a more challenging and interesting problem due to the absence of depth information.

The best-performing RGB image-based methods at the beginning were methods relying on local or global gradient-based image features. Template matching techniques were used, but they're not robust to cluttered images, noise, or light effects. Nowadays, with the rapid improvements in the Deep Learning field, they have been outperformed by trainable Convolutional Neural Networks (CNN).

In that case, several methods have been developed in recent years, and generally, they can be grouped into two categories: holistic methods and feature-based methods.

Holistic methods Given an image, these methods generally aim to estimate the 6D pose of an object in a single shot. Traditional methods rely on template-matching approaches. These approaches construct rigid templates and scan through the image, computing the similarity score at each image location. Then, comparing the similarity scores, the best match is obtained [28] [29] [7]. In the recent CNN architectures, the template is usually obtained directly from the object, by rendering the corresponding 3D model. Belonging to the Holistic category, both end-to-end and a sort of two-stage methods are included. With the advent of the CNNs architecture, due to their significant robustness to environment variations, the most trivial way to estimate 6D pose in an end-to-end manner is to regress it [43]. However, in RGB scenarios the lack of depth information for pose estimation with this approach still makes the task too difficult. In that case, a lot of methods are proposed splitting the problem into two different tasks: localize the objects in the 2D dimension in the image, inferring from this the depth, and then predict the object poses. PoseCNN [106] does exactly this, by decoupling the translation and rotation predictor, and regressing translation and rotation. However, directly estimating the 3D rotation is also difficult, since the non-linearity of the rotation space. To overcome this problem, other methods have proposed discretizing the rotation space to turn the regression task into a classification task [88].

The holistic methods typically are sensitive to cluttered environments and appearance change due to the low similarity score if objects are occluded. On the other hand, template-based methods are useful in detecting texture-less objects.

Feature-based methods Given an image and an object model, the features-based approaches try to establish the 2D-3D correspondences between the visual representation of the object in the image and the 3D object model, by extracting the local

features from both. Specifically, the correspondences are established by matching local features extracted from either points of interest or every pixel with the 3D model features. From these 2D-3D correspondences the 6D pose estimation can be recovered [71]. Keypoint-based methods are also part of this category. First, the 2D model keypoints are predicted from the image, and second, their 2D-3D correspondences with respect to the model features are computed. Keypoints could be chosen directly on the surface of the object or also they could represent eight 2D projections of the 3D model's cuboid corners. Keypoints detection traditional methods appear to have difficulties, especially in handling texture-less objects and processing low-resolution images. To solve it, recently CNNs were introduced also in this phase. For example, BB8 [73] uses segmentation to identify objects' image regions and, then, regresses keypoints. In addition, Tekin et al. [95] choose the YOLO architecture to estimate the objects' keypoints. Though they are based on low-resolution feature maps, they are not robust with respect to occlusions. Another quite different method is PVNET [72], which consists of predicting the corresponding 3D model point from each 2D pixel of the object. In this case, the network tries to overcome the problems in occluded and truncated scenarios. The authors directly regress pixel-wise vectors pointing to the keypoints and use these vectors to vote for keypoint locations. This kind of method is also called dense method, which consists of predicting the corresponding 3D model point from each 2D pixel of the object. Therefore, every pixel belonging to the depicted object produces a prediction and then casts a vote for the final result [23], [52], [86]. In other cases, to extract features and predict the corresponding 3D object coordinates for each pixel can be used random forest [5] [61] or CNNs [51].

These methods hence try to extract the relevant features directly from the image, using keypoints, coordinates, or informative pixels. To predict the final 6D pose of the objects instead they generally use a Perspective-n-point (PnP) strategy [69].

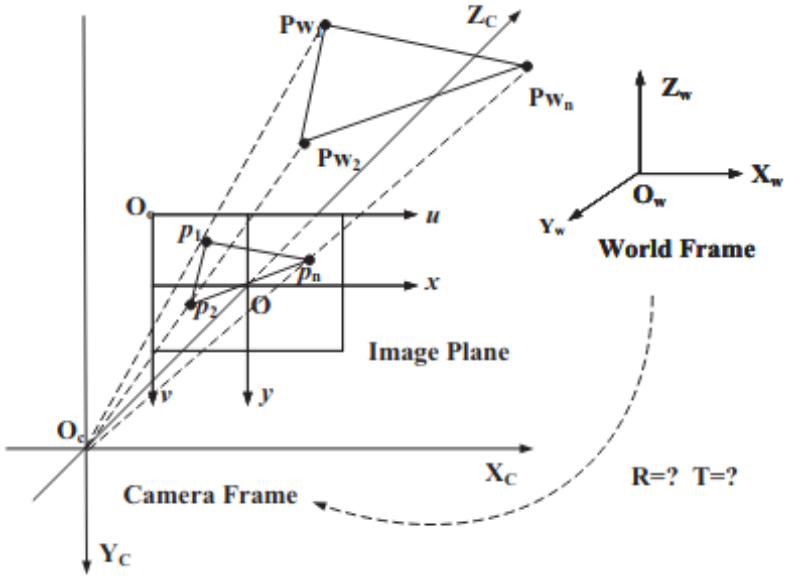


FIGURE 2.5: PnP problem. [69]

The PnP problem. illustrated in Fig. 2.5, is generally referred to estimation problems of a calibrated camera pose. In 6D pose estimation problem, PnP refers to the

estimation of the relative pose between objects and the camera, given a set of correspondences between 3D points and their projections onto the image. Let be:

$$\mathbf{P} = \{P_1, P_2, ..., P_n\}, \tag{2.25}$$

$$\mathbf{P}^I = \{P_1^I, P_2^I, ..., P_n^I\} \tag{2.26}$$

where $\mathbf{P}$ is a set of n known 3D points and $\mathbf{P}^I$ is the respective set of the projections of the points in $\mathbf{P}$. The points $\mathbf{P}$ are in a world reference frame, while the points $\mathbf{P}^I$ are in a camera reference frame. Given O to be the camera's optical center, the PnP algorithm aims to find the coordinates of each 3D point in $\mathbf{P}$ in the camera frame. Specifically, the PnP problem can be formalized as:

$$P_i^I = \mathbf{R}P_i + \mathbf{t} \quad i = 1, ..., n \tag{2.27}$$

where $\mathbf{R}$ and $\mathbf{t}$ are the rotation and translation respectively. Both are the relative transformation that PnP tries to solve to recover the positions of the 3D points in the frame. In the formulation described above the camera is considered already calibrated. Several methods have been proposed in the literature, differing first of all in the number of points considered. The P3P algorithm manages the smallest subset of points in PnP problems and is, presumably, the most studied. Along with P4P and P5P, P3P is part of the Special PnP problems. In general, in the PnP problem, several method strategies are adopted: iterative [12], non-iterative [48], [50]. Typically, the PnP methods can obtain multiple solutions. To choose a particular solution, post-processing steps could be required. Another common practice is to use Random Sample Consensus (RANSAC) algorithm [37] with a PnP method to make the solution robust to outliers in the set of point correspondences. The time performance of PnP depends on a few factors. Firstly, the complexity of the 3D geometry of the object being tracked affects the time performance. More complex objects take longer to track as the algorithm needs to process more data. Secondly, the camera resolution also affects time performance, as higher resolution cameras require more processing time. Finally, the distance from the camera to the object also affects time performance, as objects farther away from the camera take longer to track. On the other hand, the time performance of RANSAC depends on the size of the data set, the number of iterations of the algorithm, and the accuracy of the model being fit all affect the time performance. Additionally, the quality of the initial random sampling of the data points also affects the time performance, as the quality of the initial sample will determine the quality of the model to fit. In general, PnP offers good time performance, making it a popular choice for augmented reality applications. It is able to track objects in real time. However, some PnP algorithms, which need a RANSAC-based scheme to eliminate outliers, take a lot of time.

The feature-based methods typically are able to handle occlusions between objects and work fine also in truncated scenarios. However, they require textures on the objects in order to compute the local features.

These approaches could be divided into two categories: direct estimation of the 6D pose or two-stage estimation, one for the 2D keypoint detection and the other for solving a Perspective-n-point (PnP) problem for 6D pose.

Another common pose refinement, also in the 6D pose estimation problem, is the Iterative Closest Point (ICP) [2] [11]. ICP is a popular algorithm for registering

two point clouds, which involves finding the best alignment between a source point cloud and a target point cloud. The algorithm iteratively refines an initial estimate of the relative pose of the two point clouds by minimizing the distance between their corresponding points:

- i) An initial estimate of the two point clouds' relative pose is provided.
- ii) Corresponding points between the two point clouds are identified (point clouds centroids are used in some algorithm variants),
- iii) Weights are assigned to the corresponding points based on their proximity and other factors.
- iv) The relative pose of the two point clouds is estimated using a weighted method such as Singular Value Decomposition (SVD).
- v)The source point cloud is transformed based on the estimated pose.
- vi)The algorithm checks if the relative pose estimate has converged. If not, the algorithm returns to step 2.

The algorithm iterates until a given convergence criterion is met. For the 6D pose estimation problem, ICP is generally applied between the point clouds obtained from the CAD model and the RGBD image. The following discussion will focus solely on 6d pose estimation RGB-based methods; however, it's worth noting that some of these methods may use depth information outside the model for applying an ICP refinement.

BB8 Red et al. [73] introduced BB8, a novel method to predict directly the 6D pose of an object after its previous 2D detection. Given the object region, they first use the segmentation to better fit the occlusions and cluttered background. Hence, BB8 is a two-stage 6D-pose detector belonging to holistic approaches. Their CNN does not directly predict the translation and rotation, but the 2D projections of the corners of the object's bounding box, which consist of eight 2D corners. Then, BB8 computes the 3D pose via PnP algorithm [48] from the 2D-3D correspondences. Their method suffers with symmetric object views because it tries to learn a mapping from the image space to the pose space. For those objects that are perfectly symmetrical over an axis, for two or more poses their views look identical. To solve this problem, Red et al. train BB8 using images of the object under rotation in a restricted range, so the training set does not contain ambiguous images. For symmetrical poses that have rotation near the edges of the range, the model does not perform well. They also need Iterative Closest Point algorithm (ICP) for more accurate estimation. However, this approach can also not deal with pose ambiguities without additional measures. And this strategy depicts a tedious, manual way to filter out object symmetries in advance, but the treatment of ambiguities due to self-occlusion and occlusions is harder to address. For this reason, these types of methods have limitations in their applicability.

Coordinates-based Disentangles Pose Network (CDPN) Li et al. [51] proposed Coordinates-based Disentangles Pose Network (CDPN) to achieve highly accurate and robust pose estimation by disentangling the pose to predict rotation and translation separately. First, they employ a lightweight detector based on tiny YoloV3 [75] used for the next fixed-size segmentation to extract the object pixels. The translation

is predicted from the detected object region instead of the 2D-3D correspondences, while the rotation is solved by PnP from predicted 3D coordinates. They merged translation and rotation tasks in a unified network. They were able to reach good results also for occlusion and cluttered domains thanks to finding dense correspondences with their coordinates-based approach. The usage of these cropped image patches subsequently serves as the input of the actual 6D pose estimation approach which means that the whole method needs to be applied for each detected object separately. For these reasons, those approaches are often not well suited for use cases with multiple objects and runtime limitations, which inhibit their deployment in many real-world scenarios.

Real-Time Seamless Single Shot (YOLO6D) Tekin et al. [95] proposed a single-shot approach to detect the 2D projections of the 3D bounding vox vertices to the objects. Their method, based on YoloV2 [74], is end-to-end trainable and works fine also without any post-refinement. This single-shot deep CNN directly predicts the 2D projections of the 3D bounding box vertices from the image. With the 2D-3D correspondence, the 6D pose can be algebraically computed with a PnP algorithm [48], to which they pass only 9 control points. The idea is similar to the one introduced in BB8 [73], but Red et al. first find a 2D segmentation mask for a single object in a cropped image. Takin et al. instead work with the entire image. In this case, the number of objects in the scene does not affect the computational time to process the image itself, while it increases the computation time for the PnP part. Their networks make predictions based on a low-resolution feature map. However, these approaches can also not deal with pose ambiguities without additional measures. A disadvantage of this method is that it uses real labeled images, which can only be obtained within pose-annotated datasets.

Pixel-wise Voting Network (PVNet) Peng et al. [72] introduced a novel approach for overcoming the sensitivity to occlusion and truncation that the feature-based approaches suffer. Instead of directly deriving the keypoints to a given image, they consider every pixel of a cropped image referred to an object, and every pixel takes part in the keypoints identification. In particular, PVNet regresses pixel-wise vectors and uses these vectors to vote for keypoint locations. The vectors represent the directions from each pixel of the object toward the keypoints. Using the RANSAC algorithm [37], the directions then vote for the keypoint locations. RANSAC permits also pruning outliers predictions and gives a spatial probability distribution for each keypoint. In that way, this dense output permits to obtain better results in the PnP algorithm [48]. Using a pixel-wise RANSAC-based voting scheme, the method can be very time-consuming: the detection of the needed keypoints must be done separately for each object.

PoseCNN Xiang et al. [106] proposed PoseCNN, a new end-to-end semantic segmentation network that estimates the translation of an object by localizing its center in the image and predicting the depth from the camera, and the object rotation is regressed in the quaternion representation. PoseCNN decoupled the translation and rotation predictors to avoid the need to balance the training hyper-parameters needed in case of merged loss terms. Due to the behavior of the semantic segmentation, PoseCNN for estimating the translation suffers scenarios with multiple identical objects in the same image. On the other hand, directly estimating the 3D rotation is also difficult, since the non-linearity of the rotation space makes PoseCNN less

generalizable. Xiang et al. proposed also a novel loss function called ShapeMatch-Loss to handle symmetric objects. Due to their ambiguities, the network can be penalized unnecessarily during training when not taking their symmetry into account. PoseCNN reached great results over public datasets present in literature, but in real scenarios, where data are not pose-annotated this method is not easily usable.

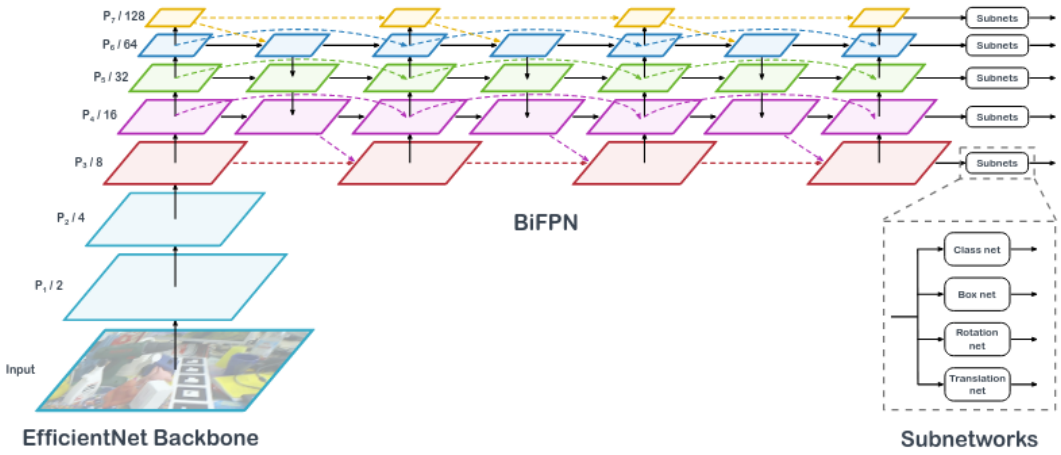


FIGURE 2.6: Schematic representation of our EfficientPose architecture. [6]

EfficientPose (EP) Bukschat et al. [6] developed a novel holistic end-to-end method called EfficientPose, depicted in Figure 2.6. This network extends the state-of-the-art object detection architecture family EfficientDets [94], based on the popular convolutional backbone EfficientNet [93], to deal with the 6D pose estimation problem. The authors add two extra subnetworks to EfficientDet, one for the rotation and one for the translation task, analogous to the classification and bounding box regression subnetworks. The rotation subnetwork predicts the rotation vector $\mathbf{r} \in \mathbb{R}^3$, in an axis-angle representation. The translation network shares a similar structure. Instead of regressing directly (t_x, t_y, t_z), the translation architecture predicts separately (c_x, c_y), which represents the object center in the image, and t_z. After this, t_x and t_y are obtained from (c_x, c_y) and fixed camera parameters, as done in [106]. Its architecture is similar to the class and bounding box regression, with the addition of an iterative refinement module. EfficientPose is capable to detect multiple object types and multiple instances of the same object in a single shot. The Loss function used is based on ShapeMatch-Loss introduced by PoseCNN [106]. Like the EfficientDet architectures, also EfficientPose is able to maintain scalability. The whole network, and also the two new subnetworks for translation and rotation, controls the size by scaling hyperparameter ϕ. The two new subnetworks are relatively small and share the computation of the input feature maps with the existing subnetworks for classification and bounding box regression. For the rotation task, the authors chose axis angle representation, while for the translation task, they adopted the same method used in PoseCNN [106], splitting the task into predicting the 2D center point and the distance separately. They also introduced a 6D-augmentation, a novel data augmentation procedure for the 6D pose, increasing performance and generalization. This proposed augmentation technique allows to augment also image rotation and scaling, moreover classical augmentation related to brightness, contrast, saturation, and hue. With respect to other methods, EfficientPose can work with the entire image,

for both multi-class and multi-instance scenarios, with negligible increase in computational time. It reaches Very good results on LineMod dataset and outperforms a lot of state-of-the-art methods.

Augmented Autoencoder (AAE) Sundermeyer et al. [88] proposed a real-time RGB-based pipeline for object detection and 6D pose estimation. Augmented Autoencoder (AAE) is their novel 3D orientation estimator based on a variant of the Denoising Autoencoder [100]. This Deep Neural Network differs from the other deep methods because it does not explicitly learn the 3D pose from data annotations but it implicitly learns the representations from the rendered 3D model views. For this reason, the AAE is classified as a self-supervised method, it does not require pose-annotated training data. The Augmented Autoencoder is trained with a novel Domain Randomization strategy. Domain Randomization (DR) consists of training a model on rendered views with several data augmentation techniques, such as random lighting conditions, different backgrounds, different levels of saturation, contrast, hue, square occlusion, and gaussian blurring. Such training permits to generalize on different backgrounds and also to real images.

Furthermore, the structure of a generic Autoencoder (AE) consists of two parts: an Encoder and a Decoder, both arbitrary learnable function approximators which are usually neural networks. Given an input image, the AE is supposed to learn how to reconstruct it while it passes through a low-dimensional bottleneck (latent space). If noise is added to the input image, AE is able to reconstruct the denoised image, due to noise invariance of the latent space. Sundermeyer et al. demonstrated that, as the Denoising Autoencoder is invariant against the noise, their method can be invariant against different input augmentations. This AAE invariance property solves the gap between reality and simulation.

Directly estimating the 3D rotation is difficult, since the non-linearity of the rotation space makes CNNs less generalizable. The authors to avoid this problem discretize the rotation space and cast the 3D rotation estimation into a classification task.

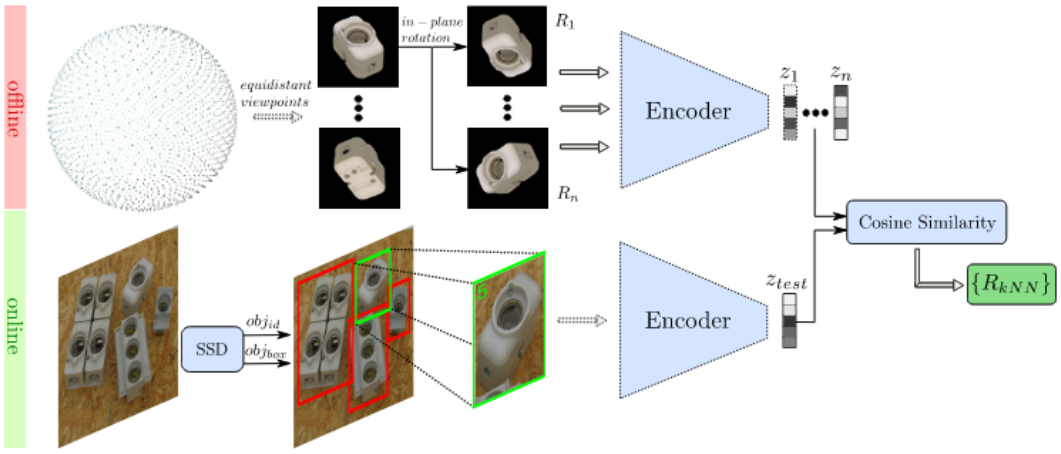


FIGURE 2.7: At the bottom, object detection and 3D orientation estimation utilizing the nearest neighbor(s) with the maximum cosine similarity from the codebook; whereas, at the top, constructing a codebook from the encodings of discrete synthetic object views. [88]

Due to their methodological choices, the AAE overcomes some well-known problems that affect other 6D-pose methods. AAE is independent from the object-pose

representations, it does not suffer from ambiguity poses given by symmetric views, and it is robust against occlusion and different environments. In detail, first, AAE applies a random augmentation f_{aug} to input x and reconstructs the original image. Then, an encoder-decoder training reconstructs the original input. Parameters are learned during the backpropagation phase, based on the per-sample loss: $l_2 = \sum_{i \in D} \|x_i - \hat{x}_i\|_2$.

After training, a codebook is created by generating a latent representation $z_i \in \mathbb{R}^l$ of each possible object view and its corresponding rotation matrix, as shown in Figure 2.7.

At test time, first the object is detected and cropped with an external detector. Secondly, the AAE gives its latent space features $z_{test} \in \mathbb{R}^l$. Then, cosine similarity is computed between the input latent representation code and all codes in the codebook:

$$cos_i = \frac{z_i z_{test}}{\|z_i\| \|z_{test}\|} \tag{2.28}$$

The highest similarity is chosen and the corresponding rotation matrix from the codebook is returned as 3D object orientation.

Only at inference time, a pose refinement can be done with the depth information and applying the ICP algorithm.

2.3 Platforms

The performance improvement allowed by modern deep neural networks and underlying computing platforms is paving the way toward new applications leveraging an improved understanding of the surrounding environment. This is particularly relevant for autonomous systems in the automotive, avionics or industrial robotics domains, where embedded domain controllers are adopted to perform inferencing workloads in real-time, while meeting strict timing, weight and power constraints.

The advent of DNNs considerably increased the complexity of designing and engineering embedded solutions, due to the many conflicting goals, the abundance of tunable parameters and configurable points, and the fragmented literature on the matter. When characterizing the performance of an object detection network for an embedded system, two main metrics are adopted: latency, i.e. the time needed for a single frame to be processed, and accuracy, i.e. the quality of the output given the input. Both metrics are of paramount importance to address the safety and timing guarantees of industrial applications. They represent a well-known bi-dimensional trade-off, as improving one metric often worsens the other. Heavier networks are typically avoided for embedded systems, as their improved precision is obtained by sacrificing latency guarantees, potentially affecting the detection reactivity of the system. Conversely, faster networks better fit the limited resources of an embedded system, but they pay a higher penalty in detection accuracy. A third important metric for embedded systems is represented by power consumption, which directly correlates with the attainable precision and latency, hence compelling the system design to a third trade-off dimension. Power consumption is a key factor to keep under control for industrial systems, especially in the automotive and avionics domains, to avoid the need for heavier batteries and heat sinks, allowing improved autonomy in a wider range of thermal conditions. Energy efficiency is often obtained by reducing operating frequencies, or by leveraging domain-specific parallel accelerators that are

able to improve performances at a smaller energy footprint than classic computing cores. In this section, only GPGPU (general-purpose computing on graphics processing units) and FPGA (Field Programmable Gate Array) families will be presented. Their embedded solutions are the best known state-of-the-art for real-time neural networks. The following is an excerpt of these embedded platforms. There is also an emerging class of AI-dedicated accelerators implemented with Application-Specific Integrated Circuits (ASICs), e.g. Google TPU, Huawei NPU, and Intel Nervana NNP. Their design is tuned to provide the best inference efficiency for the supported workloads. However, only specific networks can be supported by these accelerators, due to their limited programmability.

FIGURE 2.8: An example of a working table can be seen with a NVIDIA Xavier AGX enclosed in a green circle. Its shapes are much smaller than those of a laptop or desktop PC, demonstrating its small size and compactness.

2.3.1 GPGPU-Accelerated Platforms: NVIDIA

Orin AGX The Orin AGX is the currently best performing NVIDIA SoC. It is released in two versions: 32GB and 64GB; it is composed by 8 and 12 NVIDIA Cortex CPU cores for the two versions respectively. Orin AGX has an integrated GPU, which is based on the Ampere architecture, and sports 1792 CUDA cores and 56 Tensor cores for the 32GB version, while 2048 CUDA cores and 64 Tensor cores for the 64GB version. Both available versions have 2 second-generation Deep-Learning Accelerators.

Orin NX The Orin NX is on the second step of the podium for performance among NVIDIA SoC. It employs an SoC design that incorporates 6 and 8 NVIDIA Cortex CPU cores for the two versions of 8GB and 16GB respectively. For both versions, the integrated GPU is based on the Ampere architecture and sports 1024 CUDA cores and 32 Tensor cores. Both available versions have 1 second-generation Deep-Learning Accelerator.

Orin Nano As the name suggests, the Orin Nano is the lightest version of the NVIDIA Orin category. It is composed by 6 NVIDIA Cortex CPU cores for both two versions, 4GB and 8GB respectively. Orin Nano has an integrated GPU, which is based on the Ampere architecture, and sports 512 CUDA cores and 16 Tensor cores for the 4GB version, while 1024 CUDA cores and 32 Tensor cores for the 8GB version. The Orin Nano board does not include any deep-learning-specific accelerator. The 4GB version can work in two power modes at 5W or 10W, while The 8GB version at 7W and 15W.

Xavier AGX Xavier AGX was NVIDIA's best-performing SoC until 2021; now, it stands in third place in terms of performance among NVIDIA SoC. It is composed by 8 NVIDIA Carmel CPU cores, and an integrated GPU. This is based on the Volta architecture and sports 512 CUDA cores and 64 tensor cores. They are programmable fused matrix-multiply-and-accumulate units that execute concurrently alongside CUDA cores and implement HMMA (Half-Precision Matrix Multiply and Accumulate) and IMMA (Integer Matrix Multiply and Accumulate) instructions for accelerating various applications and, in our interest, deep learning inference. In spite of their availability, we had to ignore the presence of the 2 first-generation Deep-Learning Accelerators, whose support appears to be still immature—we obtained a considerable performance degradation when enabling them.

Xavier NX The Xavier NX is a lighter version of Xavier AGX. It employs an SoC design that incorporates 6 NVIDIA Carmel CPU cores, and an integrated GPU, which is based on the Volta architecture and has 384 CUDA cores and 48 tensor cores. As the Xavier AGX, also Xavier NX has 2 first-generation Deep-Learning Accelerators.

TX2 The Jetson TX2 has been the most powerful embedded board by NVIDIA until 2018, nd today it has been largely exceeded. It employs an SoC design that incorporates a quad-core ARMv8 A57 processor, a dual-core superscalar ARMv8 Denver processor, and an integrated Pascal GPU. There are two 2-MiB L2 caches: one shared by the four A57 cores, and one by the two Denver cores. The GPU has two streaming multiprocessors (SMs), each providing 128 1.3-GHz cores that share a 512-KiB L2 cache. The six CPU cores and integrated GPU share 8 GiB of 1.866-GHz DRAM memory.

Nano Nano is NVIDIA's cheapest and lowest performing product. It features a Maxwell GPU with a peak performance of 472 GFLOPs. The Nano board does not include any deep-learning-specific accelerator, and can work in two power modes at 5W or 10W. It does not take advantage of Tensor cores and NVDLA engines for inference acceleration.

2.3.2 FPGA-Accelerated Platforms: Xilinx

XCZU9EG As a point of reference for FPGA-based System-on-Chip, the SoC belonging to the ZCU102 development board[1] is the most powerful Xilinx evaluation board for the Zynq UltraScale+ family, yet the oldest launched hardware in our interest. The SoC is composed of a Processing System (PS) with four ARM A53 CPU cores and two ARM R5 cores for hard real-time usage, and a 600K-logic-cells FPGA Programmable Logic (PL). In neural network applications, typically the FPGA is

[1] https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

configured to host a Target Reference Design (TRD) of the Deep-learning Processing Unit[2] (DPU) with a number of processing soft cores for executing DNN-related tasks, e.g. such as convolutions, pooling, etc. The maximum available working frequencies for this architecture are 666 MHz for the DSPs, 333 MHz for the DPU cores.

XCZU7EV The smaller SoC is featured in the ZCU104 development board, equipped with 2 GiB of LPDDR4. The SoC shares the same PS as the XCZU9EG, but has only 504K PL cells.

[2] https://github.com/Xilinx/Vitis-AI/tree/master/DPU-TRD

Chapter 3

Object Detection NNs on embedded platforms

This chapter offers a systematic assessment of neural networks for 2D object detection in a real-time scenario, with a survey of multiple embedded platforms. The results reported below arise from a project in collaboration with Tetrapak s.p.a. in 2020. The goal was to identify the best trade-off between the most accurate object detector and the best performing platform in terms of latency, accuracy, and power consumption available when the study was done. Both ODCNN and the embedded platform should be integrated into the production line to solve the problem of defective items. To do so, the object detector is adopted only for the inference phase and its training phase is not relevant.

Regarding fast inference on-device computation, three major axes have been identified [8] to maximize performances or to contain power consumption.

Network Model Design There has recently been an increasing interest in deriving faster deep learning methods to be adopted for real-time embedded systems. New models are being investigated with a reduced number of parameters, a lower latency, and/or an improved accuracy. Some examples of these novel models include MobileNets [35], Single-Shot Detectors (SSD) [58], Yolo [76], and SqueezeNet [38]. In this work, at the time of writing (the year 2020), the best performing networks along the considered metrics were YoloV3 and YoloV3-tiny [75], YoloV4 and YoloV4-tiny [4], Mobilenetv2-SSDLite [78], CenterNet-Resnet101 and CenterNet-DLA34 [112].

Model Compression Quantization, parameter pruning, and knowledge distillation are popular methods used to reduce the DNN model size while sacrificing accuracy compared with the original model, in order to improve performance. In this work, quantization is adopted as a model compression method in order to achieve faster performance. Quantization is supported on many modern embedded platforms and involves converting parameters to lower precision formats, such as half-floating point precision (FP16) or INT8 inference.

Platforms Classic x86 CPU architectures have long been dominating the high-end segment of the industrial automation domain for central control systems, as they offer the best sequential performances (throughput and response time) and the easiest programmability. General-Purpose computing on Graphics Processing Units (GPGPUs) provides order-of-magnitude improvements in parallel throughput and in power usage, at a reasonable programming cost. Field-Programmable Gate Array (FPGA) platforms share a similar ambition, requiring a higher programming

cost, but providing a more flexible communication paradigm with simpler computational units. There is also an emerging class of AI-dedicated accelerators implemented with Application-Specific Integrated Circuits (ASICs), e.g. Google TPU, Huawei NPU, and Intel Nervana NNP. Their design is tuned to provide the best inference efficiency for the supported workloads. However, only specific networks can be supported by these accelerators, due to their limited programmability. For this reason, these platforms were not considered.

In this project, some original contributions are delivered, and an extension of a previous work [99] is presented below:

- in Section 3.1 the description of the platform-optimized reference deployment setup for hosting ODCNN workload for 3 hardware families and 6 platforms: an industrial PC (Intel i7-7700), NVIDIA (TX2, Xavier AGX, and Nano), and Xilinx (Zynq Ultrascale+ ZCU102 and ZCU104).
- in Section 3.2 the reference selection and implementation of 7 different SOTA ODCNNs, with free and open-source access to the code for NVIDIA[1] and Xilinx platforms[2].
- in Section 3.3, 3.4 the systematic and fair comparison of the 7 ODCNNs, in terms of mean Average Precision (mAP), latency, throughput, and power consumption on the 6 embedded boards. The setup is uniform on the input size, training dataset, and threshold for bounding boxes' (BBs) confidence, while the software stack setup and ODCNN implementation are designed to be the best available on each hardware platform.

Given the wideness of optimization means currently employed, and the number of conflicting and diversely-relevant goals, the existing reviews of the literature did not dare to address a comprehensive inspection of such a fragmented state-of-the-art (SOTA) in embedded object detection. This work goes beyond by providing a systematic assessment, evaluating all the combinations of the aforementioned networks, compressions and platforms, against all the aforementioned evaluation metrics.

3.1 Platforms

Aiming at considering representative exponents of the latest and best solutions available at the time of writing, six different hardware platforms ranging from embeddable x86 processors for industrial PC, to Arm-based SoC, including acceleration from GPGPU and FPGA are selected. To extend the scope of the comparison and maximize its fairness, for each (sub-group of) platform, what appears to be the best DNN framework and runtime available in 2020 is selected and configured. The salient features and details of the selected six DNN platforms are reported in Table 3.1. The remainder of the section highlights and motivates the rest of the platform features–hardware, system software and DNN framework.

3.1.1 GPGPU-NVIDIA

From the latest NVIDIA platforms, three boards have been chosen, widely spanning the price/performance trade-off offerings: TX2, Xavier AGX, Nano, already

[1] https://github.com/ceccocats/tkDNN
[2] https://git.hipert.unimore.it/gbrilli/dpunn

TABLE 3.1: Details of the considered boards. N.U. stands for Not Used for the implementation.

	Intel i7-7700	**Xilinx XCZU7EV**	**Xilinx XCZU9EG**
CPU	Intel i7-7700 4 cores @3.60GHz	Arm Cortex-A53 (v8) 4 cores @1.2GHz	Arm Cortex-A53 (v8) 4 cores @1.2GHz
GPU	–	Mali-400 [N.U]	Mali-400 [N.U]
Memory	16 GiB RAM	2 GiB DDR4 64-bit SODIMM w/ ECC	4 GiB DDR4 64-bit SODIMM w/ ECC
Power	65 W	≈25 W	≈30 W
Board	Industrial PC	Zynq UltraScale+ ZCU104	Zynq UltraScale+ ZCU102
DNN Accelerators	–	2× DPUv1.4@250MHz	3× DPUv1.4@330MHz
Data types	FP32	INT8	INT8
Operating system	Windows 10 Enterprise LTSC 1809	Debian Buster 10.0	Debian Buster 10.0
Framework used	ONNX Runtime	DNNDK v3.0	DNNDK v3.0
Release year	2017	2014	2014

	Nvidia Jetson Nano	**Nvidia Jetson TX2**	**Nvidia Jetson Xavier AGX**
CPU	4-core Arm A57 @ 1.43 GHz	4-core Arm Cortex-A57 @ 2 GHz, 2-core Denver2 @ 2 GHz	8-core Arm Carmel v.8.2 @ 2.26 GHz
GPU	128-core Maxwell @ 921 MHz	256-core Pascal @ 1.3 GHz	512-core Volta @ 1.37 GHz
Memory	4 GiB LPDDR4, 25.6 GiB/s	8 GiB 128-bit LPDDR4, 58.3 GiB/s	16 GiB 256-bit LPDDR4, 137 GiB/s
Tensor cores	–	–	64
Power	5W / 10W	7.5W / 15W	10W / 15W / 30W
Board	Jetson Nano	Jetson TX2	Jetson Xavier AGX
DNN Accelerators	–	–	2× Deep Learning Accelerators [N.U.]
Data Types	FP32, FP16	FP32, FP16	FP32, FP16, INT8
Operating system	Ubuntu 18.04.4 LTS, Jetpack 4.4	Ubuntu 18.04.2 LTS, Jetpack 4.4	Ubuntu 18.04.3 LTS, Jetpack 4.3
Framework used	tkDNN with TensorRT	tkDNN with TensorRT	tkDNN with TensorRT
Release year	2019	2017	2018

explained in Section 2.3.1. For the NVIDIA boards, the DNN frameworks TensorRT and tkDNN were used:

TensorRT The reference NVIDIA framework[3] is written in CUDA and optimizes the inference of deep learning models on their GPU. Using TensorRT allows one to reduce the precision data type, performing inference at 8-bit integer (INT8), or at half-precision floating-point (FP16) to replace the single-precision floating-point (FP32) in representing the weights and parameters of deep learning models. A common result [44, 34, 109] is that the overall latency of the model can be dramatically reduced, though the final accuracy could be degraded. Exploiting this framework is not always trivial, especially if the networks that need to be ported have unusual layers (e.g. deformable convolutional layers), in which case the effort of the programmer is not negligible and several plugins need to be implemented.

tkDNN The open-source Deep Neural Network library by the HiPeRT Lab[4] is built with cuDNN and tensorRT primitives, specifically thought to work on NVIDIA Jetson boards, whose main goal is to exploit those boards as much as possible to obtain the best inference performance. It is indeed confirmed as the currently fastest runtime by YOLO authors[5].

3.1.2 FPGA-Xilinx

From the Zynq UltraScale+ family by Xilinx, two exponents of development boards have been chosen: XCZU9EG and XCZU7EV explained in Section 2.3.2.

For the XCZU9EG platform, the FPGA is configured to host a Target Reference Design (TRD) of the Deep-learning Processing Unit(DPU) with a number of processing soft cores for executing DNN-related tasks. In the Programmable Logic (PL) of the XCZU9EG SoC, three DPU cores of the FA4096 family, and one Softmax core were instantiated. In the setup used, the A53 cores do not strictly behave under

[3] https://developer.nvidia.com/tensorrt
[4] https://github.com/ceccocats/tkDNN
[5] https://github.com/AlexeyAB/darknet/blob/master/README.md

the ideal offloading paradigm, as it happens in the GPU-based platforms, where it deals only with pre-/post-processing operations, image transmission to the DPU, and task code transmission, but it needs also to execute DNN layers, when they are not supported by the accelerator.

For the XCZU7EV, as described in Section 2.3.2, due to the intrinsic constraints of this architecture with respect to the previous one, the configured DPU for the experiments was obtained by reducing the ZCZU9EG TRD with: only 2 DPU cores, no Softmax core, only 35 UltraRAM blocks per core, and lower frequencies for the DSPs and DPUs respectively at 500 and 250 MHz.

The DNN frameworks DNNDK and Vitis-AI have been used for the two Xilinx boards.

DNNDK and Vitis-AI In order to help users create custom NN models, Xilinx provides the DNNDK[6] framework and the Vitis-AI[7] development suite. These collect some commonly-used NN models in the Model zoo repository, as well as tools for network quantization and deployment. These suites are integrated with the Caffe [40] and TensorFlow [1] frameworks, through which it is possible to define the model of a NN and export the trained weights. Other NN engines for FPGA exist, e.g.: Neuraghe, presented by Meloni et al. [60]; FINN, proposed by Umuroglu et al. [98] (natively supported for SoCs that integrate the Pynq framework e.g. Avnet Ultra96); or CHaiDNN, once developed by Xilinx[8]. However, DNNDK, which assumes DPU, was selected because: (i) it is directly supported by Xilinx; (ii) it is well documented both for hardware design and drivers API; (iii) it is easier to use, thanks to the NN repository, which includes a wide range of object-detection DNNs like YoloV3 and YoloV3-Tiny.

3.1.3 PC-Based Platform: Intel

x86 architecture dominates the market of central control platforms for industrial automation and is battling to penetrate the high-performance embedded segment. With respect to ARM competition, PC-based platforms often offer lower costs, maintainability, tried and true hardware with extensive processor power and nearly unlimited memory. The Intel i7-7700 is an exponent of the CPU that can be found in current Industrial PC (IPC) that combines high performance and a compact size[9], thus comparable to the other reference platforms. In that case, the DNN frameworks used are the ONNX Runtime and Pytorch library:

ONNX Runtime The chosen development framework[10]is a cross-platform, high-performance inference engine for ONNX (Open Neural Network Exchange)[11]for Machine Learning and Deep Learning models.

[6] https://www.xilinx.com/products/design-tools/ai-inference/edge-ai-platform.html#dnndk

[7] https://github.com/Xilinx/Vitis-AI

[8] https://github.com/Xilinx/CHaiDNN

[9] See products from, e.g., Advantech, Beckhoff, B&R.

[10] https://microsoft.github.io/onnxruntime/

[11] https://github.com/onnx/onnx

It allows integration of models trained from a variety of frameworks, and a convenient deployment on different platforms, including some optimization for the specific hardware accelerators and runtimes available, all using a single inference engine. To guarantee good performances during the pre-processing and post-processing of the images, the C++ API has been exploited.

PyTorch DNN models have been exported from PyTorch 1.4 [70], which natively supports ONNX layer export supporting the Constant Folding graph optimization technique. A smooth conversion is guaranteed if PyTorch operators are supported by ONNX. Custom PyTorch operators can be still registered as ONNX operators through the ONNX Registration API. ONNX stable opset supported by PyTorch 1.4 is version 9. We have encountered no problem exploiting opset version 11 for the conversion of the model studied.

3.2 Neural Networks

The selection of the ODCNNs under analysis has been driven by the sake of excellence and currentness. An exhaustive analysis has been conducted to look for the best performance in terms of execution time, mean average precision, or trade-off between the two. The Object detectors chosen in this analysis are CenterNet, YoloV4 and its tiny version, YoloV3 and the tiny counterpart and Mobilenetv2-SSDLite, already explained in the section 2.1.4.

Several survey articles [105][41][115] review OCDNN, but all of them focus only on the mAP metric. The four kinds of NNs chosen are : designed for real-time applications, well-established in the literature, lead to the best results[12]. Table 3.2 shows, for every selected ODCNN model: the number of parameters (# Params) – read-only data expressed in Millions of elements (M); the intra-layer Input/Output tensor size (I/O Tensors) – read-write data expressed in Millions of elements (M); the computational complexity (# MACCs) expressed in Billions of operations (G).

TABLE 3.2: Selected ODCNN Models Specification

Network	# Params	I/O Tensors	# MACCs
YoloV3	61.9 M	16.8 M	49.9 G
YoloV3-tiny	8.8 M	8.49 M	4.2 G
Mobilenetv2-SSDLite	4.3 M	12.6 M	2.1 G
CenterNet-ResNet101	49.0 M	12.6 M	47.7 G
CenterNet-DLA34	19.0 M	8.4 M	30.8 G
YoloV4	64.3 M	16.8 M	45.5 G
YoloV4-tiny	6.0 M	4.1 M	5.2 G

For YoloV3 the porting of the original network, as it is, was possible on the NVIDIA, Xilinx and i7-7700 boards. On the NVIDIA platforms, For YoloV3-tiny the porting of the original model was possible. On the other hand, a small change has been required for the XCZU9EG and XCZU7EV implementation. Precisely, the last max-pool layer has been removed as it was not supported during the deployment phase of the model on the Xilinx DPU platform. The issue is caused by a mismatch between how Darknet and Caffe frameworks handle max-pooling layers of stride

[12] https://paperswithcode.com/sota/real-time-object-detection-on-coco

1, with input size equal to output size. However, removing this single layer did not affect too much the performance of the network. The mAP of the network has been measured with and without the aforementioned layer and the tests showed a negligible accuracy drop (from a mAP of 11,9% to 11,7%, computed on the Xavier AGX). The conversion of YoloV3 and YoloV3-tiny to ONNX has been done automatically by the PyTorch to ONNX exporter, without adding new operators. Also for Mobilenetv2-SSDLite the porting of the original model was possible on the NVIDIA platforms. Instead, to obtain the original model on ONNX Runtime, it is necessary to highlight the switch from the ReLU6 PyTorch operator to the Clip ONNX operator with a minimum value of 0 and a max value of 6.

On XCZU9EG and XCZU7EV, whose AI library ecosystem does not currently support the PyTorch framework, a model conversion to Caffe [40] through an open-source tool[13] has been necessary. This also forced us to replace the final Reshape layers, which are not supported by the DPU architecture, with Flatten type layers. In addition, the final Softmax layer has been removed from the model and implemented in software and accelerated with OpenMP, because the memory movements between the Programmable Logic and Processing System would have been inefficient. The time achieved with OpenMP-accelerated Softmax is 9.56ms compared with the FPGA-based implementation that can achieve at most 13.64ms.

For CenterNet, due to the particular structure of this model, the porting on the NVIDIA platforms required a great implementation effort, while the porting on the Xilinx platforms was not possible. Indeed, at the moment of writing, the instruction set of the DPU microarchitecture does not support deformable convolutional layers. Although the DNNDK / Vitis-AI ecosystem supports the integration of software-implemented layers, the resulting network could not achieve satisfactory performance due to the inefficiency of the A53 cores in carrying out convolution operations with respect to the DPU cores, and due to the continuous shift of weights and activations from PS to PL. Neither was it possible to convert and properly port CenterNet on ONNX Runtime. According to our research in 2020, there is no supported DCNv2 operator for PyTorch, ONNX, and ONNX Runtime for CPU. Further work involves the implementation of DCNv2 as PyTorch custom operator for CPU, its registration as a custom ONNX operator and as an ONNX Runtime operator.

Also for the YoloV4 the porting of the original model was possible on the NVIDIA platforms. For the Xilinx platforms instead, the porting was not possible[14] due to the maxpool of the SPP module. It was the same problem as for YoloV3-tiny, but in this case, removing those maxpool layers would have changed the core of the network, leading to a different architecture. On the i7-7700 platform, an available ONNX model has been tested, but this test led to very poor results. Latencies are very high, more than 3 times the expected, and precision very low. Probably, not the model nor the ONNX-RT implementation of the mish layer are mature enough to fairly compare this porting with the others, which is why its results are not reported here.

Finally, for YoloV4-tiny the porting of the original model was possible on the NVIDIA platforms and on the i7-7700, but not on Xilinx boards, where the peculiar grouped route layer was not representable.

[13] https://github.com/xxradon/PytorchToCaffe

[14] https://forums.xilinx.com/t5/AI-and-Vitis-AI/Yolov4-support-to-DNNDK/td-p/1103325

3.3 Test Data

Dataset To properly compare ODCNNs performances, network configurations must be as uniform as possible. It would be unfair to compare, e.g., the latency if input sizes are different, or the accuracy when input image resolution used for training is different. Hence, one of the most widely used datasets in object detection has been adopted, i.e. COCO [55]. Input size and training set were chosen after the newer networks (CenterNet, YoloV4), using COCO2017 with input size 512x512. This input size allows discriminating more clearly the differences between the latencies, while achieving good accuracy. With respect to the 2014 version, COCO2017 is divided in a 118K-images training set, and a 5K-images validation set, but it preserves the same 80 semantic classes.

Training For CenterNet and YoloV4 networks, the weights from the original implementations were used, already fitting our requirements. Instead, YoloV3, YoloV3-tiny and MobileNetv2-SSDLite were trained, performing a single, full-precision training per network, and then exporting the obtained weights for the different frameworks. YoloV3 and YoloV3-tiny were trained using darknet, their original framework, on the selected dataset, with input size 512x512, using the default hyperparameters. MobileNetv2-SSDLite was also trained from the original implementation (PyTorch 1.3), on the selected dataset and input size, using default hyperparameters except for the number of epochs that was set to 400.

3.4 Experiments

3.4.1 Experimental setup

All the tests were performed on the COCO2017 validation tests, counting 5K images. Best-, average- and worst-case were measured for end-to-end latency, average power consumption and mAP. For the NVIDIA platforms, all supported data types were considered: FP32, FP16 for TX2, Xavier AGX and Nano, INT8 for Xavier AGX only. XCZU9EG and XCZU7EV supported only INT8, and only FP32 is supported by the i7-7700. The INT8 quantizations have been obtained on 1000 images of the COCO2017 training set, both on Xilinx and NVIDIA boards. To maximize NVIDIA platforms' performance, the mode has been set to MAX N and *jetson_clocks* has been launched before the tests. To model an industrial setup with external workload, e.g. a central control platform with real-time motion tasks, only two cores of the i7-7700 have been exploited. Henceforth the acronyms used are: CNet(D34) for CenterNet-DLA34, CNet(R101) for CenterNet-ResNet101, Yolo3 for YoloV3, Yolo3tiny for YoloV3-tiny, Yolo4 for YoloV4, Yolo4tiny for YoloV4-tiny, Mv2(SSD) for MobileNetv2-SSDLite.

3.4.2 Metrics

Mean Average Precision The mean Average Precision (mAP) 0.5:0.95 is the "de facto" metric for object detection [77] [56] and depends on two thresholds: (i) confidence threshold t_c, taking into account only BBs with a confidence score greater than t_c; and (ii) the IoU thresholds t_{IoU} to discriminate two BBs representing the same object if their classes match and if their IoU is greater than t_{IoU}.

End-to-end Latency The execution time can be divided in: (i) pre-processing to convert the image in the NN input, (ii) NN inference, (iii) post-processing to convert

the output of a NN into BBs. The end-to-end latency is the time elapsed between feeding an image to the detector and obtaining the BBs. In each board, the pre-processing and the post-processing is performed in full-precision (FP32), while the inference can be quantized (FP16 or INT8). Pre-/post-processing are performed on the CPU, except for NVIDIA boards, where pre-processing of every network and post-processing of CNet(D34) and CNet(R101) were optimized on GPU, implementing a CUDA version of the corresponding (slower) OpenCV functions. The tests were performed on 5k images, and maximum, minimum and average end-to-end latency over those 5k latency records were computed. The first value of each measurement series has been filtered out, because it would pessimistically account for code and network weights memory movements and cold caches. This is a behavior of the first iteration. This situation won't repeat in these kinds of recurrent tasks, and this justifies our choice.

Efficiency Efficiency was measured as Frame Per Seconds (FPS) over the power consumption (W). The power usage has been sampled at 40 Hz on the NVIDIA and Xilinx boards using *powerapp* tool[15], at 1 Hz on the i7-7700 using Open Hardware Monitor 0.9.2.

Parallelization Lastly, the performance of the embedded platforms has been evaluated in a multi-stream scenario, comparing the behavior of the various method using 1, 2, 4 and 8 images at a time, in terms of end-to-end latency and FPS. The network-level parallelism considered in this work is quite different comparing NVIDIA and Xilinx implementations.

Batching Within the NVIDIA board, the classical neural network batching can be exploited. Working with more than one image means adding a dimension to the inference tensor. In tkDNN, as well as in many other frameworks, the tensor shape is (N, C, H, W), which stands for *(number of batches, number of channels, height, width)*. A bigger tensor, comprising N images, is fed in inference at one time: the latency for a single image will increase, but, thanks to the better exploitation of all the GPU resources, also the throughput will. However, pre-processing and post-processing need to be performed as well, and it is important to mention that those instructions are not batched but, in this implementation, run sequentially.

Multi-core For the Xilinx counterpart, the DPU ecosystem allows to efficiently exploit its multi-core architecture by replicating the full computing pipeline of the DNN. In the implementation proposed in our open-source repository, a multithreaded program has been developed that, based on the number of streams considered, spawns an equal number of tasks that offload the inference on the least loaded DPU core. For the sake of comparisons, a *Master Task* scatters the images of the validation set across the *Slave Tasks* which in turn offload the computation to the DPU cores, but in a general scenario, it is possible to have also different DNN models running in parallel on top of different DPU cores. This solution could in principle allows for a complete parallelization across the replicated components of the FPGA. Our setup cannot fully exploit it, though, due to the constrained PS that host pre/post-processing phases load—a linear jitter increase is expected.

[15] https://git.hipert.unimore.it/tetra-pak/dl-arch/powerapp

3.4.3 Results and Discussion

Only a selection of the obtained results is presented below. The entire results are available at https://git.hipert.unimore.it/edbench/edbench/.

Confidence Threshold and its Effects

The confidence threshold t_c is a ODNCC parameter used at post-processing time to heuristically exclude possible false positives. To showcase high mAP results, t_c is usually set to 0 (or 0.05, if the method does not allow 0, as for YOLO and SSD), while 0.3 is preferred in real world applications. For the sake of fairness and realism, we let $t_c \in \{0.05, 0.3\}$. Fig. 3.1 shows the results focusing on three aspects: (i) the mAP of the network (y-axis); (ii) the worst-case post-processing latency of the network (x-axis); (iii) the number of detections (radius of the points). When changing the confidence threshold, more or less BBs are returned, affecting only the post-processing time. Varying the threshold affects the mAP and number of detections in all three platforms: when $t_c = 0.05$, mAP and number of detections are higher, increasing the post-processing latency to a different extent on the considered platforms.

Platform Differences The NVIDIA platforms have the highest variance in the post-processing phase. For example, considering Mv2(SSD), lowering the threshold makes AGX, TX2 and Nano respectively score a slowdown of 85×, 135×, 400× on the maximum latency. The latency distributions in these cases exhibit an extremely wide tail, e.g. for Xavier AGX we recorded these values (ms): 595 max, 10.1 3rd quartile and only 4.96 median. The i7-7700 platform is considerably stabler, showing only a 3× slowdown in the same example. This is due to the fact that, post-processing operation being sequential and executed on CPU, where the i7-7700 scales considerably better than Nvidia's Arm CPUs. Xilinx platforms, instead, are the slowest ones for $t_c = 0.3$, but are also modestly sensible to threshold variation, since the multithread implementation allows core parallelization. Considering mAP at full precision, it can be noticed that NVIDIA obtains higher values than the i7-7700. We suppose that the small accuracy gap (under 1-2%) using identical models and the weights can be ascribed to the different runtimes and instruction sets architecture adopted—ONNX-RT over x86-64 Intel's Kaby Lake, and TensorRT over PTX NVIDIA's Volta.

Networks differences YOLO methods produce fewer BBs than the others, even with small thresholds, while the number of detections for Mv2(SSD) explodes when using $t_c = 0.05$. CNet(D34) and CNet(R101) have stabler post-processing times, thanks to the fact that the method always returns at a maximum of 100 BBs for image and it is optimized on GPU.

In the rest of the assessment, $t_c = 0.3$ is used.

Platforms Comparison

Latency Against Precision It is explored in Fig. 3.2 (top), where grey, dashed lines suggest Pareto-optimality curves. For any network and precision, we first observe that mAPs are expectedly very similar on the various platforms. The highest mAPs for all networks are achieved by the NVIDIA boards, except for Mv2(SSD) at INT8, where a higher mAP is obtained on the XCZU9EG and the XCZU7EV. Where different precisions are considered, we unsurprisingly noticed that mAP can be exchanged for latency reduction by varying from FP16 to INT8. The same holds also

TABLE 3.3: Best/Worst measured mean power consumption. In brackets, the respective data-type and ODCNN model.

Platform	Best [W]	Worst [W]
Jetson Nano	8.61 (FP16, Mv2(SSD))	13.57 (FP32, Yolo3)
Jetson TX2	7.86 (FP16, Mv2(SSD))	15.585 (FP32, Yolo3)
Jetson Xavier AGX	13.11 (INT8, Mv2(SSD))	38.43 (FP32, Yolo3)
Intel i7-7700	32.42 (FP32, Mv2(SSD))	39.86 (FP32, Yolo3)
Xilinx XCZU9EG	17.86 (INT8, Mv2(SSD))	22.77 (INT8, Yolo3)
Xilinx XCZU7EV	18.43 (INT8, Mv2(SSD))	22.29 (INT8, Yolo3tiny)

when changing from FP32 to FP16, but not on NVIDIA platforms, where remarkably a minor or negligible mAP variation can buy a significant latency reduction.
The Xavier AGX is the platform that clearly Pareto-dominates the others in terms of both metrics. Following, there is the TX2, which dominates the rest, but no dominance relation can be established between the remaining four platforms.

Latency Breakdown Pre-processing, inference, and post-processing latency are separated in Fig. 3.3 using the worst-case. On Intel and NVIDIA platforms, the longest phase is clearly inference. For the Xilinx boards, instead, pre- and post-processing are longer. This can be firstly ascribed to the A53 being the slowest performing CPU of the considered platforms. Furthermore, the load of the post-processing phase has been augmented so to include the final normalization steps of the NN, which could not be executed on the DPU because of missing API (i.e. sigmoid for Yolo3 and Yolo3tiny) or unacceptably slow implementation (i.e. softmax for Mv2(SSD)). Also relevant is the fact that inference on Xilinx platforms is quite fast: beside Xavier AGX, it is shorter than Nano, TX2 and Intel, which all are at FP32 precision. Considering only NVIDIA, the Xavier AGX always dominates the other two and the TX2 always dominates the Jetson Nano. The latter has similar results to the i7-7700, and for the Mv2(SSD) is even slower.
When considering average-case latencies in Fig. 3.4 the time spent in pre- and post-processing is smaller, while the inference latency is similar to the worst-case. Besides, considering Jetson Nano for the average-case, it performs better than i7-7700 for Yolo3, and worse for the Yolo3tiny model.

Efficiency Against Precision This tradeoff is depicted in Fig. 3.2 (bottom). The i7-7700 is the platform that consumes the most (up to 39.86 W). Being also slow, it is therefore the least efficient board according to this metric. Xilinx platforms are instead the ones that consume less. However, due to the low FPS achieved, those are not the most efficient. Xavier AGX configured at FP16 dominates the rest of the platforms except for the INT8 configuration. After these the Pareto-dominance orderly ranks: TX2, Nano, XCZU9EG, XCZU7EV, and i7-7700.

Table 3.3 shows, for every platform, the best and worst mean power consumption (in Watt) measured during the execution of all the different ODCNN inferences. For each power consumption case presented, the table displays also the linked model and data-type. Excluding the x86 architecture, the Jetson Xavier AGX is the most power-hungry embedded device (up to 38.43 W for FP32, Yolo3). Instead, the Jetson Nano and TX2 are the boards draining the least power among the platforms under test, both consuming approximately the same amount of power (from $\approx$ 8 W to 15 W). The Xilinx boards sit in the middle, with power consumption ranging between 18 W and 22 W.

Networks Comparison

Latency Against Precision Fig. 3.2 (top) shows that Yolo4 always achieves the highest mAP, even when quantized at INT8. Yolo3tiny has always the lowest latency in each board, for each precision, but also the lowest mAP. There are only few Pareto-dominance relations to highlight because the selected networks widely span the latency/precision trade-off. We can remark only that CNet(D34) is dominated by all other networks, and that same happens also for Yolo3 in the restricted case of Xavier AGX.

Efficiency From Fig. 3.2 (bottom), it can be noticed that Yolo3tiny and Yolo4tiny are always the most efficient networks, with the former dominating the others in terms of efficiency. Yolo3 is again dominated by CNet(R101), and is the least efficient one on all platforms.

Multiple stream

Fig. 3.5 reports average-case end-to-end latency (x-axis) versus average FPS (y-axis). For each board and network, a line connects performance with $N \in \{1, 2, 4, 8\}$ images. For the sake of readability, the chart reports only tests with a significant throughput improvement—at least of 5% when comparing $N = 1$ with $N = 8$.

It can be noticed that the throughput visibly increases only for the three boards at INT8 precision: XCZU9EG, XCZU7EV and Xavier AGX.

In general, however, the behavior of the boards in this scenario is not excellent. Two factors need to be considered to achieve better performances when using more images: (i) the dimension of the NN model, and (ii) the input size of the network. In the reported results, better improvement results can be seen for the three lighter networks (i.e. Yolo3tiny, Yolo4tiny and Mv2(SSD)) with precision INT8. On the TX2 and the Nano this effect cannot be appreciated because those boards do not support INT8, have the poorest GPUs with respect to Xavier AGX and have smaller memory. Using a smaller input size would help to achieve better results on those small boards.

However, focusing on the overall view, the Xavier AGX is again confirmed as the best board among all the considered.

Fig. 3.6 offers an insight into the latency phases' performance when using multi-stream, showing all the boards but only the networks supported by them all. The chart reports pre-processing (top row), inference (middle row) and post-processing (bottom row) behaviors for single stream, when $N \in \{2, 4, 8\}$, normalizing the values with the $N = 1$ case. This helps understanding where is the bottleneck when using more streams. First of all, the bottleneck does not depend on the network model, but only on the platform. For the NVIDIA platforms, the heaviest part is definitely the inference, which, in these cases, grows linearly with N. This is due to the batched inference: you have to pay the total computation of N images even to obtain single stream results. On the opposite, the pre-/post processing phases are constant with respect to $N = 1$. This behavior is similar for the Xavier AGX, the TX2 and the Nano, and it is not influenced by the data type (INT8 or FP16).

On the other side, for the Xilinx boards, the inference phase is almost constant for each considered N: in this case, the hardware is parallelized (there are 3 DPUs) and the inferences are independent. However, these boards do suffer the pre-/post-processing phases, due to CPU contention of the various threads.

General considerations

The best real-time object detection networks at the time of writing have been considered and deployed on six representative cutting-edge embedded boards, exploiting the corresponding ML frameworks. For the CenterNet and YoloV4 models, only the porting on the NVIDIA boards was possible, due to limitations of DPU and ONNX APIs. For this reason, it would be very helpful having also on Xilinx an easy way to write and add not supported layers, to smooth the porting of a model (as TensorRT plugins).

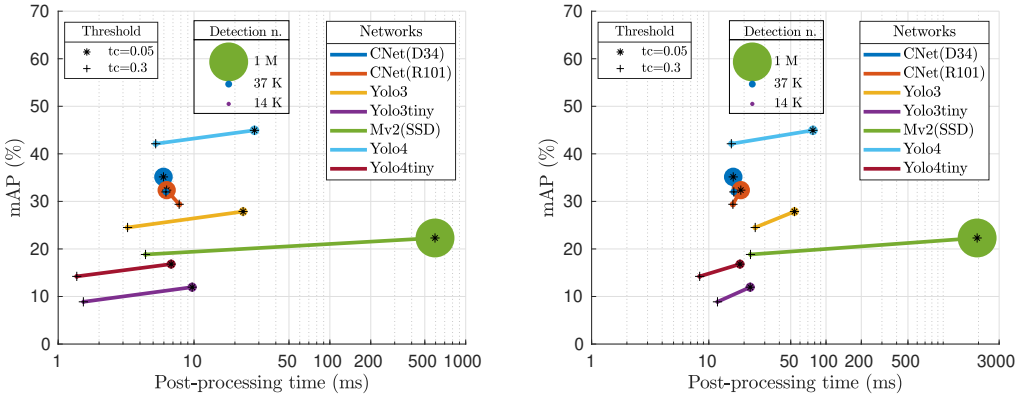
An exhaustive evaluation of networks performance shows that the Pareto-optimality curve intercepts four of the seven considered DNNs. YoloV4 is the one achieving the highest accuracy, while YoloV3-tiny is the best network in terms of latency and power consumption, but also the one achieving the smallest mAP. YoloV3 is dominated by other models and it is the most power greedy. MobileNetv2-SSDLite is the most affected by the confidence threshold.

About platforms, Xavier AGX is the clear winner in almost all considered aspects, achieving the best power efficiency, as well as the highest mAP. Among the NVIDIA boards, clearly, the Xavier AGX dominates the TX2, which dominates the Jetson Nano. The Xilinx platforms have a very stable power consumption for all considered networks, and dominate the i7-7700 in terms of efficiency and inference latency. The i7-7700 is the least efficient board, but it is also the one with better sequential performance, leading to a smaller post-processing latency variance. Moreover, it has comparable performance with respect to the Jetson Nano.

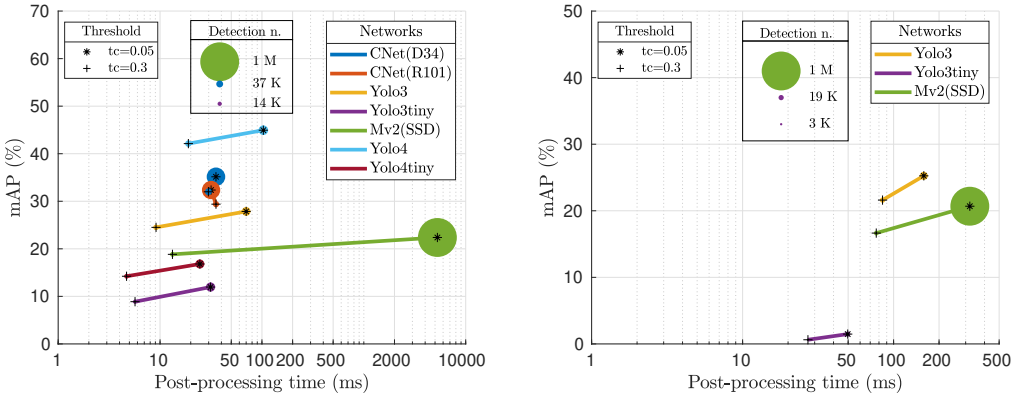
When considering multi-stream, Xilinx boards have better results in terms of inference with respect to all the other boards but suffer the pre-/post- processing phases; opposite behavior can be found in NVIDIA boards, regardless of the data type or of the selected board. When considering end-to-end latency and total throughput, Xavier AGX is again the winning board. However, when dealing with multi-image detection, to obtain a good throughput boost it is recommended using (i) light models (e.g. YoloV3-tiny, YoloV4-tiny or MobileNetv2-SDDLite), (ii) small input size (the smaller the better, but recalling that the mAP will decrease with the size).

Finally, considering data types, FP16 only negligibly deteriorates the accuracy of the considered networks, but obtaining much better real-time performance, always laying on the Pareto-optimality curve. A significant accuracy deterioration is instead experienced with INT8. Therefore, FP16 represents our recommended choice for platforms that support this precision.

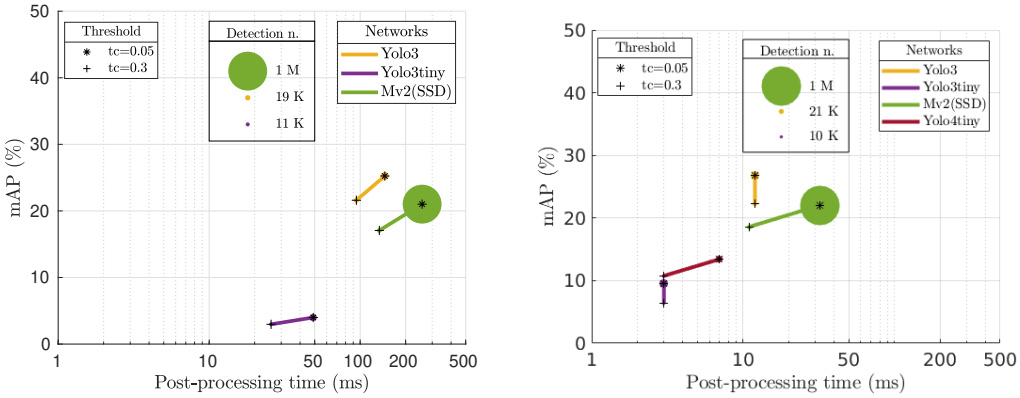
Although ODCNNs and embedded platforms have come a long way in terms of performance over the years, the methodological analysis for a fair intra- and inter-networks/platforms comparison highlighted in this chapter remains a valid and useful approach to follow.



(A) Xavier AGX (FP32) (B) Jetson TX2 (FP32)



(C) Jetson Nano (FP32) (D) XCZU9EG (INT8)



(E) XCZU7EV (INT8) (F) i7-7700 (FP32)

FIGURE 3.1: Comparison of mAP (y-axis) versus worst case post-processing latency (x-axis) using confidence threshold $t_c = 0.05$ $(*)$ or $t_c = 0.3$ $(+)$. The radius of the point is proportional to the number of detections.

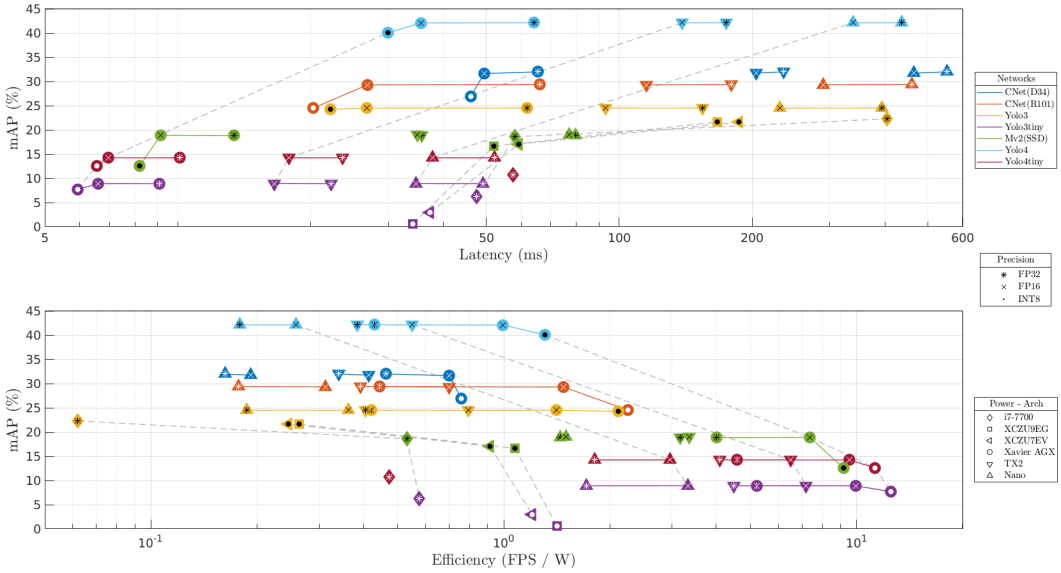


FIGURE 3.2: Comparison of the different networks on the three platforms considering average-case latency and power.

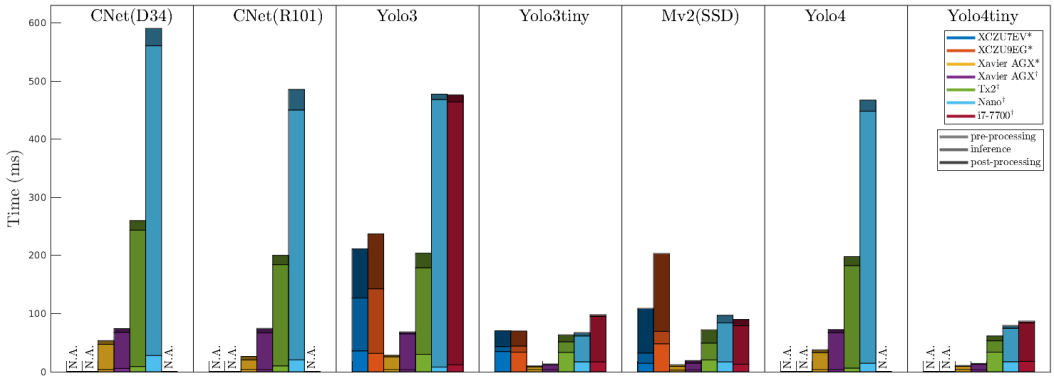


FIGURE 3.3: Worst-case execution time divided in pre-processing, inference and post-processing with respect to the end-to-end latency. $*$ stands for INT8, † for FP32.

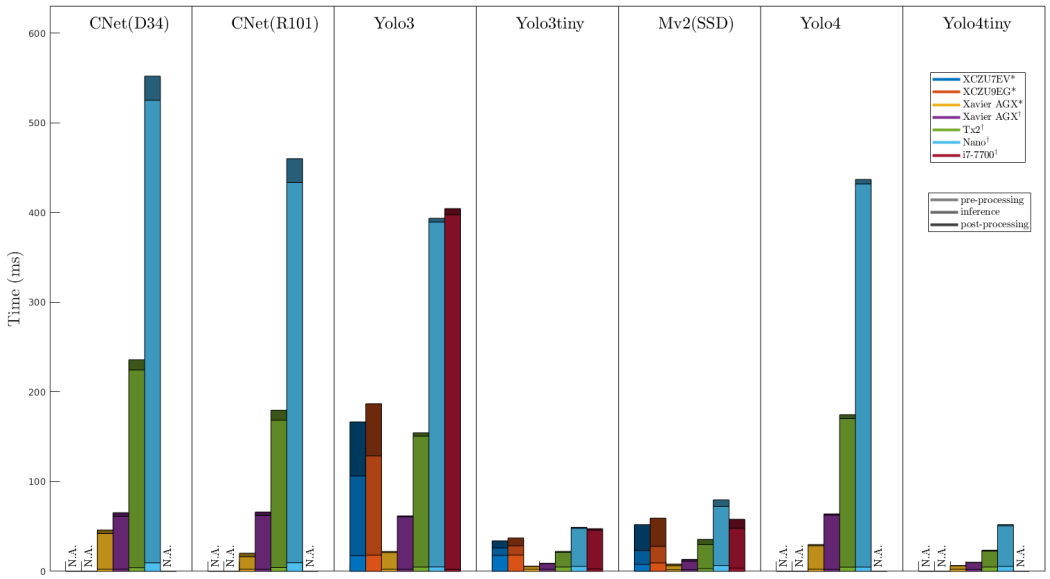


FIGURE 3.4: Average-case execution time divided in pre-processing, inference and post-processing with respect to the end-to-end latency. $*$ stands for INT8, † for FP32.

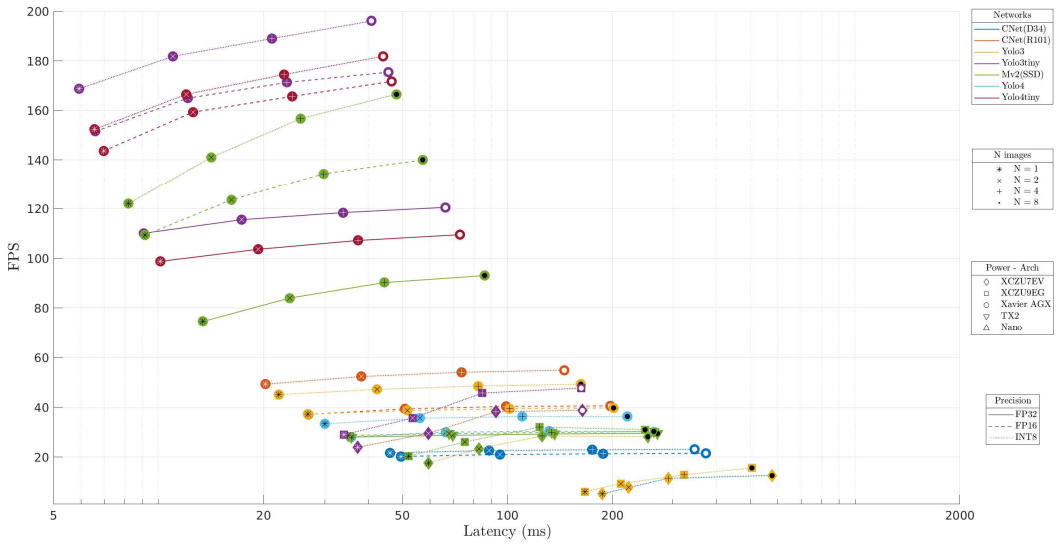


FIGURE 3.5: FPS vs end-to-end latency on average for $N \in \{1, 2, 4, 8\}$ images. Results such that $FPS_{N=8}/FPS_{N=1} < 0.05$ are omitted.

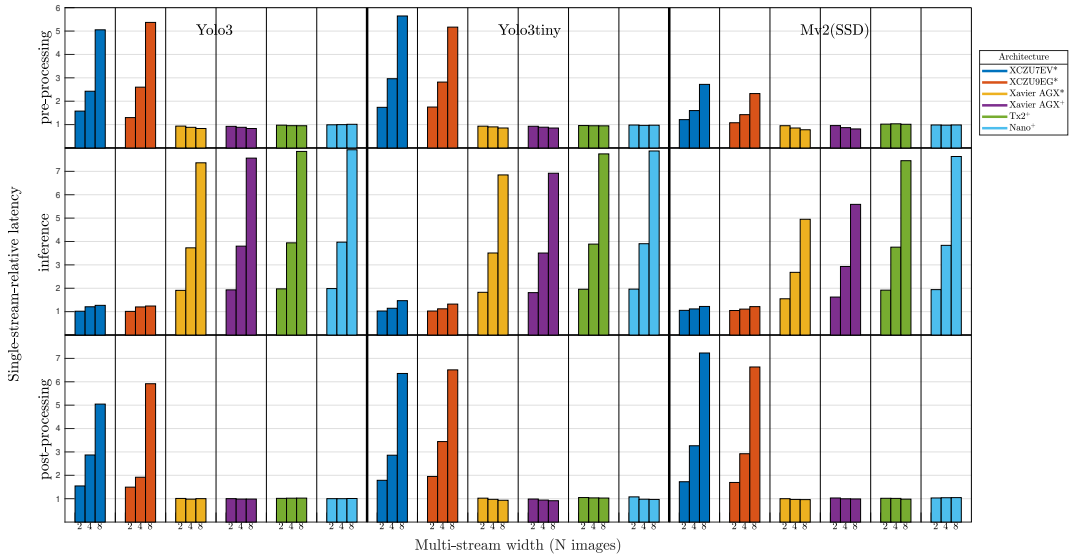


FIGURE 3.6: Multi-streaming speed up: N-images single-stream average latency relative to $N = 1$, for $N \in \{2, 4, 8\}$. Pre-processing (top), inference (middle) and post-processing (bottom). $*$ stands for INT8, $+$ for FP16.

Chapter 4

6D pose estimation explainability

In the 6D pose estimation domains, recent learning-based techniques that use CNN, based on RGB data, have proven to achieve very high performance scores in the 6D object pose estimation, achieving state-of-the-art results. These types of methods generally require huge amounts of labeled training examples and pay the cost of being extremely data-driven. While for other computer vision tasks the dataset acquisition and labeling are easy to obtain, resorting to manual labeling, this is not the case for 6D object pose, since identifying ground truth translations and rotations from real images is not easily feasible for a human annotator. The research community has resorted to either relying on large datasets obtained from a photorealistic simulation [32] or to smaller datasets of real-world images labeled by relying on fiduciary markers [29, 5].

In this Chapter two interrelated aspects will be discussed. On the one hand, the real-world datasets will be analyzed, for those real-world datasets that rely on markers to extract the object pose ground truth. The interesting aspect is to analyze if the presence of easily recognizable shapes of markers might bring about a bias in the learning procedure, resulting in an improved success in the 6D pose estimation. In this sense, LineMod dataset [29], the most common one in the 6D object pose problem, will be investigated. On the other hand, an attempt will be made to understand which methods could be subject to such bias.

A brief methods review will be addressed, and then EfficientPose [6] will be analyzed since *i*) it achieves the best state-of-the-art performance on LineMod [1], and *ii*) it meets the architectural standards for suffering from bias. The aim of this chapter is to present an analytical analysis of the problem previously highlighted and, considering a specific use case, pose some general questions about constructing datasets for the 6D pose estimation task and determining the most suitable methods for the application-specific scenario.

Annotation induced Bias Commonly used 6D pose estimation methodologies, as introduced above, use the LineMod dataset [29]. This dataset, described in Section 2.2.2, is labeled by fixing the target objects to a rectangular board that is surrounded by ArUco markers [21]. The ground truth object pose is then retrieved by deploying geometric algorithms which use markers to recover first the board's pose in camera coordinates, and, successively, the object's pose in the same coordinate system. Utilizing simple image processing algorithms and geometry, the tags are utilized to recover the board's pose in camera coordinates, and finally the object pose is then calculated by applying the known fixed offset from the board's coordinates system to the object's origin.

[1] https://paperswithcode.com/sota/6d-pose-estimation-on-linemod

The ArUco chessboard used to recover the ground truth pose is visible in the training and evaluation images. Therefore, it is interesting to research the effect that the background and markers have on a similar model to the one detailed in Section 4.1 when predicting the 6D pose from the whole image. Additionally, the arrangement of the objects on the board could cause the model to learn a shortcut or induce unintended behavior. As LineMod is a dataset for single-object, only the pose for the object in the middle of the board is provided for training. The hypothesis is that some models could leverage the aspect of other visible objects to infer the 6D pose for the target object in an unintended way.

When assessing the efficacy of a proposed method based on LineMod, or similar datasets, it is important to consider its generalization capabilities. This entails evaluating the performance of the method when applied to different scenarios and practical applications, such as robotic manipulation or object tracking for trajectory planning purposes. Furthermore, factors such as changing backgrounds or the lack of ArUco markers or other types of markers should also be taken into account.

To this end, the following factors should be taken into account:

- How beneficial is it to have the target object positioned in the center of the board?
- To what extent is the method affected by a static or semi-static background?
- Does the network utilize 6D pose information from visible markers and the objects surrounding it?

Methodology overview In the 6D pose estimation problem, the methodologies proposed in the literature can be divided into two different groups as described in Section 2.2.4. Some methods perform the calculation over the entire image frame to directly infer the depicted objects' pose, while others must be performed a number of times equal to the number of objects in the scene. The former can be considered more general and complex, while the latter take a simpler approach, dividing the problem into subproblems. For the latter, the computational time increases as the number of depicted objects increases.

In order to identify the 6D pose estimation methods that perform better in real-time applications, considering the possible biases mentioned above, some considerations about methods should be made.

Basically, to define a method as real-time, the architecture complexity plays one of the main roles. Currently, there is no real real-time community for the 6D pose estimation domain. In any case, a 6D pose estimation method that depends on the number of objects appearing in the scene is clearly not suitable for real-time domains. Going in this direction, two-stage methods have been chosen not to be considered. Also, methods that work on cropped images may not be suitable for this study.

Regarding the possible annotation bias introduced in real-world datasets, it is intuitive that some methods working with the croped scenes depicting the target object are less susceptible to the ArUco markers' presence in the RGB image than the full RGB-based methods.

For this reason, a qualitative and quantitative analysis of EfficientPose [6] will be presented, which was chosen as an ideal candidate for attempting to answer the

aforementioned questions due to its state-of-the-art performance on LineMod [2] and its ability to operate on the full image. The purpose of this analysis is to illustrate a possible process to assess the generalization properties of a model, which is an essential requirement for any real-world application.

In addition, the significance of selecting a proper dataset when training new models will be emphasized below. In fact, relying on a dataset that introduces some bias could lead to deceptive outcomes. Besides the ArUco marker bias discussed in this chapter, it is important to consider another significant one which is related to the bad division of the training and validation/test sets. In many real-world datasets, such as LineMod, the validation set comprises a random selection of frames from the dataset. Consequently, the neural network is trained on some frames and subsequently evaluated on some contiguous frames that are presumably similar to those seen during training, with the target object's pose being identical or nearly identical. Instead, A fair approach would be to split the training and validation sets sharply to ensure that the samples are significantly different.

The work presented in this chapter could further be situated within the field of 6D pose explainability, an area which, to the best of my knowledge, has not been discussed previously. Given the prevalence of CNNs in computer vision and their tremendous power, it is essential to use interpretable models which can explain their predictions. This is important for identifying failure modes, enabling researchers to concentrate on the most promising directions. Furthermore, to ensure the reliability of CNNs in real-world applications, it is necessary to set up appropriate confidence and trust.

4.1 Overview

LineMod As mentioned in chapter 2, LineMod [30] is the most common dataset used both for training and evaluating the 6d methods in the literature (depicted in Figure 4.1).
LineMod is a real dataset consisting of 15 classes or object models, each of which has ∼1200/1300 different views. The objects are not occluded, located around the image center, and easily distinguishable. In particular, for each object class, a single instance of the item is placed on the working plane's center. The working plane is bounded by custom-made aruco markers, typically used to calibrate the camera which were used to provide the object pose ground truth. Inside this kind of chessboard are placed several other objects surrounding the target one, that cause only mild occlusion, and which could indirectly contribute to distinguishing the target object's pose. Some other objects are placed outside the chessboard and placed haphazardly on the desk. The ∼1200/1300 images are collected by shooting the same camera at different angles to differentiate the view from one acquisition to the next. In this way, from one image to another, a change in pose is referred to both the target object and also to all other objects, including the ArUco chessboard.

EfficientPose EfficientPose [6], as described in Section 2.2.4, is an extension of a widely used 2D detector, EfficientDet (ED) [94], based on the popular convolutional backbone EfficientNet [93]. In a single shot, the architecture is able to predict the class, the 2D bounding box, rotation, and translation of one or more objects, given

[2] https://paperswithcode.com/sota/6d-pose-estimation-on-linemod

FIGURE 4.1: Samples from the LineMod dataset

an RGB image as input. In detail, two analogous to the classification and bounding box regression subnetworks are added to EfficientDet, modeled after the classification and bounding-box regression of the original model. The rotation subnetwork predicts the rotation vector $\mathbf{r} \in \mathbb{R}^3$, in an axis-angle representation. Its architecture is similar to the class and bounding box regression, with the addition of an iterative refinement module. The final rotation is then the sum $\mathbf{r} = \mathbf{r}_{init} + \mathbf{\Delta r}$ where $\mathbf{r}_{init}$ is the initial estimate for the rotation, while $\mathbf{\Delta_r}$ is the iterative refinement module, given as output of separable convolutional layers, group normalization and activation functions. The translation network on the other side shares a similar structure. Instead of regressing directly (t_x, t_y, t_z), the t-architecture predicts separately (c_x, c_y), which represents the object center in the image, and t_z. After this, t_x and t_y are obtained from (c_x, c_y) and fixed camera parameters, as done in [106].

A transformation loss function is added to the original ED loss function, with a regularization parameter λ. The final loss function is then composed by:

- Smooth L1 as regression loss function
- Focal loss as classification loss function
- Transformation loss as pose loss function. This is added to the original EfficienDet loss function and is described as follows:

$$\mathcal{L}_{tr} = \frac{1}{m} \sum_{x \in \mathcal{M}} \|\hat{\mathbf{P}}\mathbf{x} - \overline{\mathbf{P}}\mathbf{x}\| \qquad \text{if } \mathcal{M} \text{ not symmetric}$$

$$\mathcal{L}_{tr} = \frac{1}{m} \sum_{x_1 \in \mathcal{M}} \min_{x_2 \in \mathcal{M}} \|\hat{\mathbf{P}}\mathbf{x_1} - \overline{\mathbf{P}}\mathbf{x_2}\| \qquad \text{if } \mathcal{M} \text{ symmetric}$$

where $\overline{\mathbf{P}}$ is the ground truth pose and $\hat{\mathbf{P}}$ is the estimated pose.

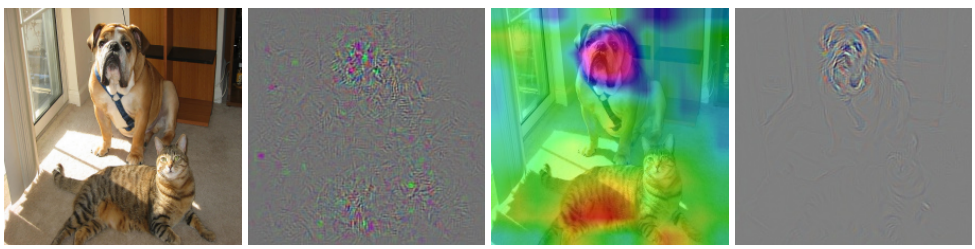


FIGURE 4.2: Output of saliency methods for the class Mastiff. Source [67]

Saliency Maps Interpretable Machine Learning (IML) [63] has experienced rapid growth in the Machine Learning domain, primarily focusing on elucidating the process behind a model's specific prediction. While some models, such as Decision

Trees and Rule Based Classifiers, are inherently interpretable, DNNs cannot be interpreted in the same manner and are generally considered to be a black-box predictor after training. Pixel attribution is a family of attribution methods designed to interpret the prediction of models that operate on images, with the aim to produce a saliency map which highlights the importance of individual (or groups of) pixels in the input image for the model's specific output. One of the simple methods to generate saliency maps, introduced in [84] is sometimes referred to as Vanilla-Gradient, and it consists of computing the gradient of the output prediction with respect to the input image. More formally,

Given $s = \Psi(I)$, where $\Psi(\cdot)$ denotes the (trained) neural network, the gradient $\frac{\partial \Psi}{\partial I}$ can be easily obtained via standard backpropagation. Grad-CAM [82] is a more recent method to produce saliency maps: differently from the vanilla gradient, the gradient of the output y_s, that is the class score before the softmax, is computed with respect to the output feature map A^k of a convolutional layer (ideally, the last one before the global average pooling of traditional classification models). The gradient tensor is then averaged across the spatial dimensions indexed by i, j to obtain a single vector $\alpha_k^s \in R^c$ where c denotes the number of channels:

$$\alpha_k^s = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_s}{\partial A_{ij}^k} \tag{4.1}$$

Finally, the saliency maps are obtained by weighting each channel of the feature map A^k with the corresponding value of α_k:

$$L_{GC} = ReLU\left(\sum_k a_k A^k\right) \tag{4.2}$$

4.2 Methodology

The potential for bias to be introduced into the model due to the presence of visible artifacts, particularly markers employed in the data collection process, is to be explored by this research. The LineMod dataset, a well-known non-synthetic dataset, and the EfficientPose model, which is currently the most effective fully-convolutional network for LineMod, will be focused on for experimentation. A mixed evaluation of qualitative observations, utilizing the attribution methods discussed in Section 4.1, and quantitative experimentation, involving the evaluation of results obtained from modified versions of the LineMod dataset, is proposed to assess the validity of the hypothesis.
The proposed methodology can be briefly summarized as follows:

- The pre-processing of LineMod by deliberately masking the visible markers and the use of the new version of the dataset for the training of EfficientPose is proposed. The 6D pose estimation task is then compared between the late and the original training of EP.
- A generalization of pixel attribution methods for a regression problem is also introduced, and it is shown that the saliency maps produced with the extended version of Grad-CAM support the hypothesis that the model's predictions are contingent upon the presence of fiduciary markers on LineMod.

In the remainder of this section the building blocks of the proposed methodology will be explored, then in Section 4.3 the conducted analysis is proposed and discussed in detail.

4.2.1 Dataset Masking strategy

In this analysis, two datasets have been used. One is the original LineMod dataset. The other is a modified LineMod Dataset, where ArUco markers are covered with black squares to verify if the ArUco markers impact pose predictability. The process of deleting ArUco markers is based on geometrical tools and on the objects' ground truth. Considering the 3D origin point $[0, 0, 0]$ as the object's pivot, when no trans-

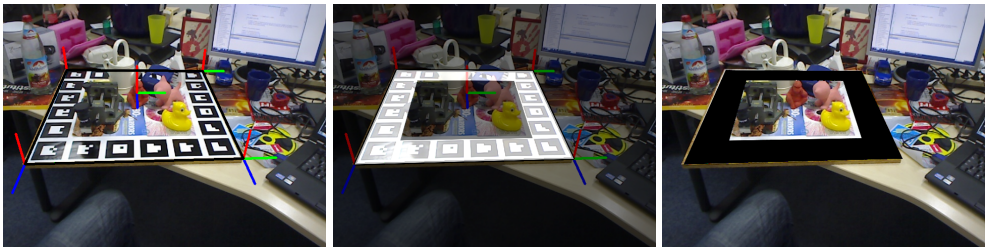


FIGURE 4.3: Three steps of our geometric procedure in order to remove ArUco markers.

lation and rotation are applied, the positions of the black square's four corners can be identified. The four corners correspond to the upper-left, upper-right, lower-left and lower-right corners of the ArUco chessboard. Each corner is represented by a 3D point defined as $[\pm\delta_x, \pm\delta_y, \pm\delta_z]$, where δ_x, δ_y, δ_z are the axis offsets between the corners and the origin. Corners are identified only once for each object. Then, for each object image, the corresponding rotation and translation are applied to accurately project the black square at its correct image position (see Figure 4.3). At this stage, it is sure that the target object is not obscured, through the use of the target object's mask image provided in the original LineMod dataset. ArUco-Free LM Dataset adopts a simple strategy in order to show and discuss how the ArUco markers influence final results. However, in the ArUco-Free Linemod Dataset, another bias is introduced when a black square is positioned in alignment with the object's pose, resulting in a similar but narrow bias. Across frames, the network may be able to infer the object's pose directly from the black square. Nonetheless, due to the nature of the square, a fixed view of the square corresponds to four different possible rotations. Additional, advanced methods could be utilized such as masking the ArUco markers with random patterns (varying in form and hue) rather than using a black square, or utilizing image restoration techniques, such as image inpainting methods [64].

In addition, ArUco masks were used also for computing a density map, able to highlight their presence. The images in Figure 4.4 indicate for each object which areas are ArUco markers concentrations. It is evident that these markers are not equally distributed, instead they focus on the same areas.

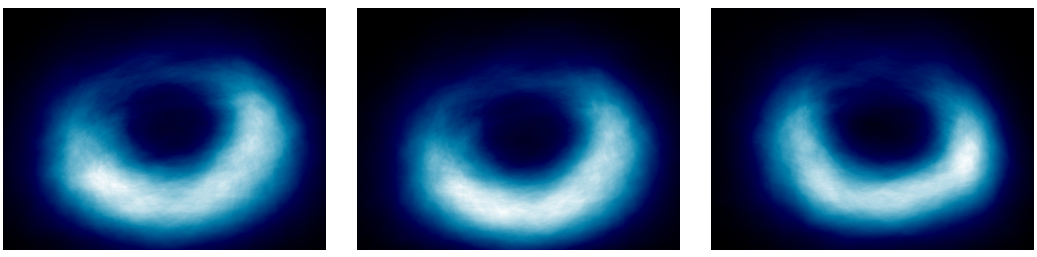


FIGURE 4.4: Density maps based on masks. From left to right: object 1, object 5, object 11.

4.2.2 Saliency maps

Inspired by the domain of interpretable AI, qualitative analysis was sought to interpret and extract information about how the network learns. This is a fundamental phase in order to assess the capabilities of 6D methods in other real-world scenarios, since metrics are limited in providing insight into a network's comprehension.
Saliency methods are mainly developed for multi-class classification problems [77], and are thus sometimes referred to as Class Activation Maps (CAM) methods. At inference time, classification models output a probability score for each possible class, and the highest scoring class is selected as the predicted one. However, in this study, a regression problem is investigated: this is pertinent as the gradient-based methods that are planned to be used, Vanilla Saliency [84] and Grad-CAM [82], suppress the negative part of the gradient since it corresponds to a decrease in the score for the class of interest. In a scenario similar to ours (i.e, regression of rotation values in $[-\pi, \pi]$), the magnitude of the gradient is clearly of interest. Hereafter, the exact formulation of the saliency methods used in the work is formalized.

Vanilla Saliency It is straightforward to adapt the vanilla saliency for the regression. The Rotation head of EfficientPose outputs a tensor $\overline{R} \in \mathbb{R}^{N\times 3}$ of N candidate regressions. In a single-object scenario, as in this case, the rotation vector $\mathbf{r}$ for the target object is recovered as the one with the highest confidence. An identical approach can be adopted also for the translation regression. The gradient of $\mathbf{r}$ computed with respect to the input image I is a tensor with the same shape as the input image, which is reduced to a single channel by averaging. Unlike the original formulation, both the negative and positive values of the gradient are retained for the reasons mentioned above. For visualization, the saliency map (single-channel) is normalized to the interval $[0, 255]$ using min-max.

Grad-CAM Adapting Grad-CAM to this problem is far more challenging. The original formulation operates on the feature map produced by the last convolutional layer of a classification model; however, the structure of the regression head of EP (simplified in Figure 4.5) makes it difficult to choose the correct feature map. Therefore, a feature map obtained as the combination of the three convolutional layers that precede the output of the initial prediction $\bar{r}_{init}$ is opted to be used, as the refinement module is used to predict small additive offsets to the initial prediction, which may be Identity mappings if the initial prediction does not require refinement.

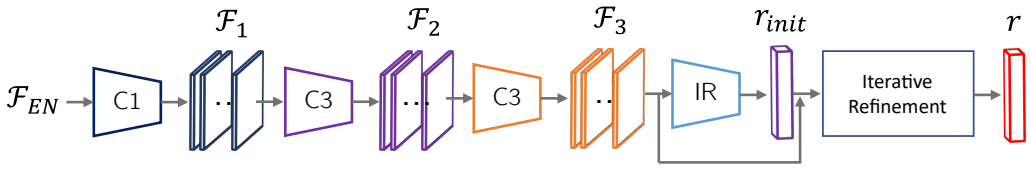


FIGURE 4.5: Rotation subnetwork for EfficientPose. The Translation subnetworks share an identical structure.

Let $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3 \in \mathbb{R}^{\overline{W}\times\overline{H}\times\overline{C}}$, with $\overline{W}, \overline{H}, \overline{C} \in \mathbb{N}$ be the three intermediate feature maps, an aggregated featuremap $\mathcal{F}_t \in \mathbb{R}^{\overline{W}\times\overline{H}\times 3\overline{C}} = \{\mathcal{F}_1 \oplus \mathcal{F}_2 \oplus \mathcal{F}_3\}$ are constructed, where $\oplus$ denotes the concatenation on the channel axis. Equation 4.1 is adapted by replacing A^k with $\mathcal{F}_t$ and replacing the average pooling with L_2 norm, in order to avoid opposing gradient values to elide each other in the sum. The new

formulation for the pooled gradient becomes:

$$\alpha_t = \sqrt{\sum_{i=1}^{\overline{W}} \sum_{j=1}^{\overline{H}} \left(\frac{\partial \mathbf{r}}{\partial \mathcal{F}_t^{ij}} \right)^2} \in \mathbb{R}^{3\overline{C}}$$

The computation of the saliency maps is obtained by weighting each channel of $\mathcal{F}_t$ with the corresponding value of α_k and accumulating over the channel axis to obtain a single matrix. Differently from Equation 4.2 the ReLU is removed and the absolute value of the weighed feature map is instead taken:

$$L_{GC} = \sum_{t=1}^{3\overline{C}} |\alpha_t \mathcal{F}_t|$$

The produced saliency map is normalized and interpolated to the input image shape for visualization. The formulation, derived from simple but solid mathematical observations, results in an ideal generalization of Grad-CAM to our case.

4.2.3 Evaluated Task metrics

Two metrics, described in detail in Section 2.2.3, are used to test the networks:

- **ADD**: the average distance computes the mean distance between each point of the 3D model obtained by the pose matrices $\hat{\mathbf{P}} = [\mathbf{R_{est}}, \mathbf{t_{est}}; \mathbf{0}, 1]$ and $\overline{\mathbf{P}} = [\mathbf{R_{gt}}, \mathbf{t_{gt}}; \mathbf{0}, 1]$

  Given the model $\mathcal{M}$, the estimated pose $\hat{\mathbf{P}}$ and the ground truth $\overline{\mathbf{P}}$

  $$e_{ADD} = avg_{x \in \mathcal{M}} \|\hat{\mathbf{P}}\mathbf{x} - \overline{\mathbf{P}}\mathbf{x}\|$$

- **ADI**: the average closest point distance computes the mean distance between each point of the 3D estimated model and its closest neighbor on the ground truth model:

  $$e_{ADI} = avg_{\mathbf{x_1} \in \mathcal{M}} \min_{\mathbf{x_2} \in \mathcal{M}} \|\hat{\mathbf{P}}\mathbf{x_1} - \overline{\mathbf{P}}\mathbf{x_2}\|$$

  It is preferred if the model $\mathcal{M}$ has indistinguishable views.

- **Criterion of Correctness** The estimated pose is considered correct if $e < \theta_{AD} = k_m d$
  where k_m constant generally equal to 0.1, d = object diameter

4.2.4 Proposed Evaluations

For the experimental analysis reported here, the rotation subtask of EP is investigated, since the translation regression is based on an identical structure, as introduced in Section 4.1, the same principles are agnostic to the regression subtask. The results, both qualitatively and quantitatively, obtained by two architecturally identical instances of EP will been compared: the original model trained on the official LineMod dataset and an alternative version trained on the ArUco-Free dataset introduced in Section 4.2.1. Since the focus is on single-class EP, the evaluation procedure is independently performed for three distinct representative object classes from LineMod.

To summarize, a total of 6 distinct versions of EP are trained: the standard version of EP ($\phi = 0$) is picked, and trained on subsets of objects 1 (Ape), 5 (Can) and 11 (Glue) of both original (LM) and ArUco-Free (AF-LM) datasets. The Ape and Can objects are asymmetrical: the former is one of the smallest objects and the latter is one of the largest objects in the dataset. The glue, on the other hand, is a symmetrical object. For each trained model, the 6D-pose metrics (Section 4.2.3) and saliency maps (Section 4.2.2) are computed on both the validation subsets of LM and AF-LM for the corresponding object. In the following section, the complete analysis is provided.

4.3 Results and Discussion

4.3.1 Quantitative analysis

Three objects are considered: one with symmetric views (object 11, glue) and the other two asymmetric (object 1, ape and object 5, can). Considering ADD(-S) as a metric, the following rule is used:

$$ADD(-S) = \begin{cases} ADI, & \text{if } obj \text{ sym} \\ ADD, & \text{if } obj \text{ asym} \end{cases}$$

Object 1: the ape Table 4.1 shows how EfficientPose performs on the two different datasets. It can be observed that in both training (with and without ArUco markers), the networks learn from the background. In fact, if the test dataset is different from the train one, the pose estimation accuracy collapses. The cause of this outcome can also be attributed to the bias discussed at the beginning of this chapter regarding the unfair division of the training and validation sets. Figure 4.6 compares estimated (blue) with ground truth (green) bounding boxes, with weights learned on the Original LM Dataset. When ArUco-Free Dataset is used, in some cases the rotation is wrong, in others the object is not even detected. Probably, when the ArUco markers are covered, the network still learns from the black square around the object, as stated earlier in the Section 4.2.1. Object 1 achieves the worst performances on both datasets.

	Test		
Original LM training	Original LM 0.8771	ArUco-Free LM **0.0162**	**0.4467**
ArUco-Free LM training	ArUco-Free LM **0.8848**	Original LM 0.0	0.4424

TABLE 4.1: This is object 1. ADD is computed on the two datasets with the two types of weights available (trained with $\phi = 0$). Then, in the right column, for each training, the two test results' averages are computed, in order to observe which experiment performs better both on the test images of the same image type seen during training, and on the test images of the type never seen.

Object 5: the can For object 5, the can, the trained EfficientPose weights with $\phi = 0$ on the Original LM Dataset and on the ArUco-Free Dataset are used. The object is not symmetric, therefore ADD is used. For this object, we noticed that the results are similar to the ones of object 1, and confirm the thesis of background-induced bias.

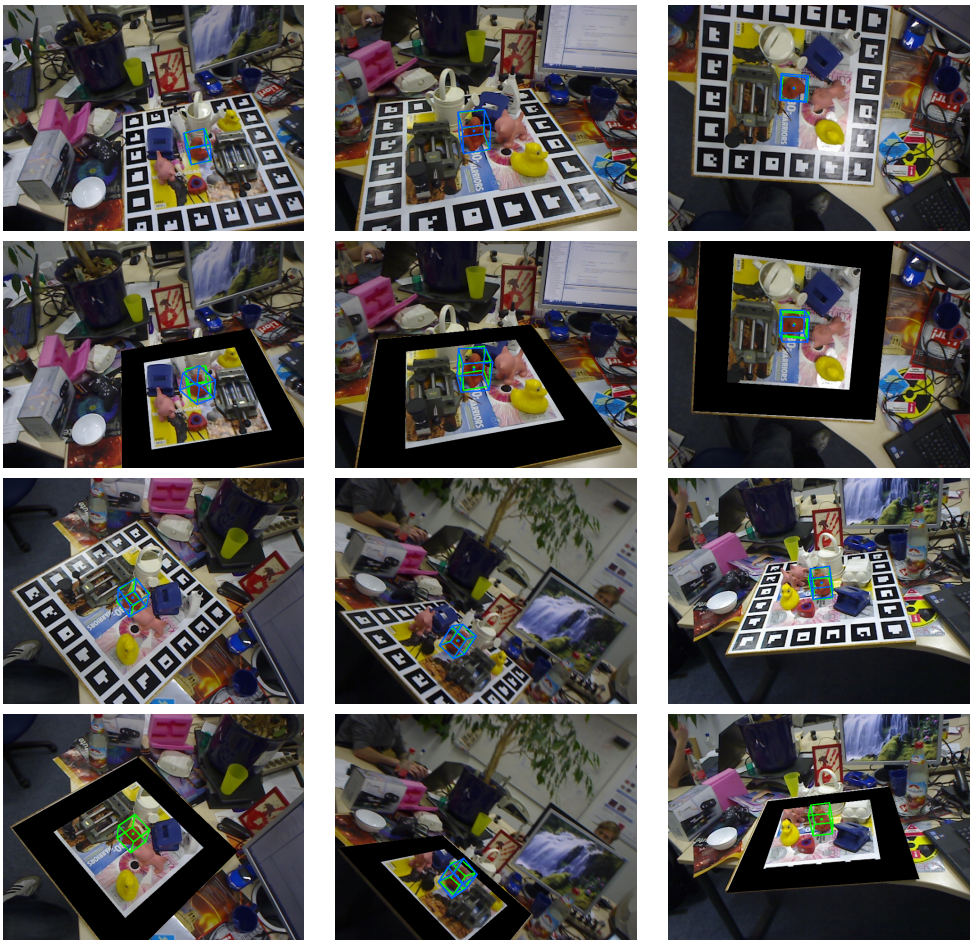


FIGURE 4.6: These images are used during the test phase on object 1 with weights downloaded from the available Efficient Pose training ($\phi = 0$). The green box represents the ground truth pose, while the blue one represents the estimated pose. On the first and third rows there are test images from the original LineMod dataset, while on the second and fourth rows there are the correspondent images from ArUco-Free dataset. We observed that the boxes are all wrong and, in some cases, the network doesn't even detect the object.

However, in this case, EfficientPose with our ArUco-Free Dataset performs better in both cases (Original LM and ArUco-Free LM datasets), as shown in Table 4.2.

Object 11: the glue The third object analyzed is a symmetric object, for this reason, ADI results are shown in Table 4.3. They are higher since ADI is more relaxed than ADD. In this case, while performances on the same dataset of the training (first column of Table 4.3) are almost perfect, with an accuracy of 100%, the training without ArUco performs better than the other with the opposite dataset (second column of Table 4.3). Therefore, the average accuracy is better. From these metrics it is evident that the network learns from the background. However, accuracies don't give us information about which background areas are more relevant than others. To explain better these results, the saliency maps are used in the next section.

In Table 4.4, both mAP and ADD(-S) metrics are reported for the considered objects. The rows specify the tests performed, with details of the training set and the test set on which the metrics are calculated. In the first and third rows, the two

Test			
Original LM training	Original LM 0.9852	ArUco-Free LM 0.0315	0.5083
ArUco-Free LM training	ArUco-Free LM **0.9921**	Original LM **0.1673**	**0.5797**

TABLE 4.2: This is object 5. ADD is computed on the two datasets with the two types of weights available(trained with $\phi = 0$). Then, in the right column, for each training, the averages of the two test results are computed, in order to observe which experiment performs better both on the test images of the same type of the images seen during training, and on the test images of the type never seen.

Test			
Original LM training	Original LM **1.0000**	ArUco-Free LM 0.3031	0.6516
ArUco-Free LM training	ArUco-Free LM **1.0000**	Original LM **0.4083**	**0.7042**

TABLE 4.3: This is object 11. ADI is computed on the two datasets with the two types of weights available (trained with $\phi = 0$). Then, in the right column, for each training, the two test results' averages are computed, in order to observe which experiment performs better both on the test images of the same image type seen during training, and on the test images of the type never seen.

different trained models are tested on the test set of the same type as their training set. the ADD values for object 1 and object 5, and ADI for object 11 respectively, are comparable by observing these two experiments, while the mAP is always equal to one. On the other hand, for the second and fourth rows, in which the two experiments are evaluated on two test set following different distribution with respect to the corresponding training set, some changes can be observed. In general, the model trained on LineMod ArUco-Free performs better than the model trained on original LineMod in terms of both mAP and ADD. For object 1, the model trained on Original LineMod gets the lowest mAP value and a very low ADD value, while the model trained on ArUco-Free LineMod achieves the highest mAP value, but its ADD is equal to 0. As described in Equation 2.1.3, the mean Average Precision measures both classification and localization performance. In this case, a high mAP value represents good performance of the two EP subnetworks, for the classification and bounding-box tasks. However, the predictive capability of the rotation subnetwork is very poor, as can be deduced from the low ADD(-S) values.

4.3.2 Qualitative analysis

Figure 4.7 shows Vanilla Saliency maps. They are computed with respect to two Original LM images. The weights are given by EfficientPose code for object 1, with $\phi = 0$. The map represents the gradient magnitude for each pixel. Since EP has a first common feature extraction phase and, then, is divided into different subnetworks with their own output (classification, bounding boxes, rotation, and translation), two saliencies for each image have been compared. The 'rotation' image represents vanilla saliency map based on the rotation subnetwork, while the 'classification' image comes from the classification subnetwork. The differences for object 1 in this

	mAP & ADD(-S) results					
	Obj 1		Obj 5		Obj 11	
Original LM training Original LM test	mAP 1.0	ADD 0.8771	mAP 1.0	ADD 0.9852	mAP 1.0	ADI 1.0
Original LM training ArUco-Free LM test	mAP 0.8295	ADD 0.0162	mAP 1.0	ADD 0.0315	mAP 0.9739	ADI 0.3031
ArUco-Free LM training ArUco-Free LM test	mAP 1.0000	ADD 0.8848	mAP 1.0	ADD 0.9921	mAP 1.0	ADI 1.0
ArUco-Free LM training Original LM test	mAP 0.999	ADD 0.0	mAP 1.0	ADD 0.1673	mAP 0.9962	ADI 0.4083

TABLE 4.4: mAP and ADD(-S) are reported for each experiment. On the rows, the couples of the training dataset and the test dataset are reported. The first two rows belong to training with Original LineMod dataset, while the other two rows belong to training with the ArUco-Free LineMod dataset (trained with $\phi = 0$). The results are reported for the three objects tested.

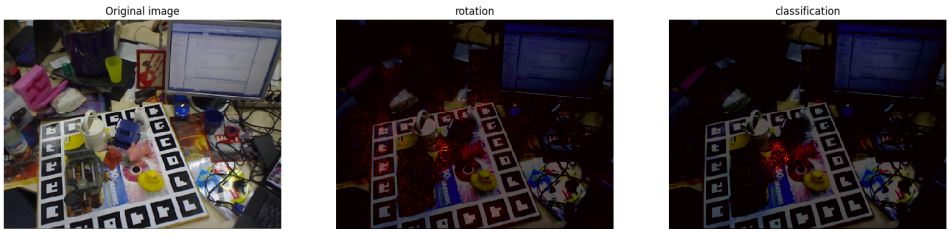


FIGURE 4.7: Vanilla Saliency applied on two images of the Original LineMod dataset. Weights are trained on the same dataset. The gradient is computed given two different outputs: in the second image it is based on the regressor, in the third image it is based on the classificator.

image are significant. In fact, while the saliency for classification is grouped into the principal object, the ape, instead the rotation saliency is more scattered and goes also on the marker chessboard. It probably means what is expected: that, for the position output, a bias is induced by background, which plays a fundamental role. In order to improve our study with a more sophisticated and explainable saliency method, the saliency maps with GradCAM method are also computed. Moreover, sometimes Vanilla Saliency could have a saturation problem, as shown by [83] and [63].

In Figure 4.8, different results for the three objects can be observed. The weights are learned on the Original LM, while the tests are computed on both datasets. For example, for object 1 (on the left), pixels with higher saliency values are, on the Original LM (top), distributed on the object and the markers, whereas on the ArUco-Free LM, they do not put focus on markers, in fact they are covered. It means that, when ArUco values collapse to zero, they are not anymore interesting for the rotation estimation.

For object 5, in the center, the focus is on a large area which includes also the can. Therefore, the network uses not only the principal object, but also the background one, to estimate the final pose. This is possible since background objects do not change their position with respect to the can.

Similar to the ape behavior, for object 11's pose estimation EP focuses on marker chessboard, not even observing the principal object. When markers are zeroed,

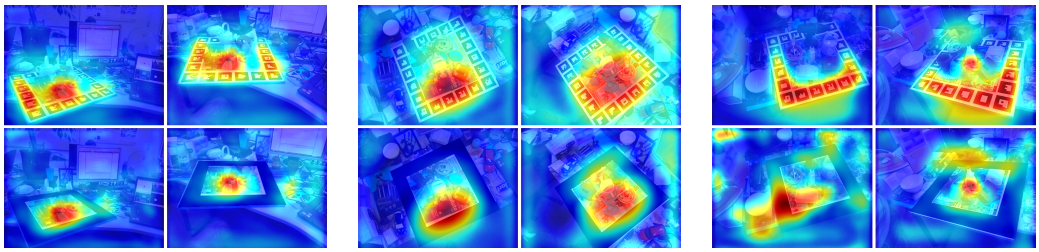


FIGURE 4.8: On the top: saliency maps with weights provided by EfficientPose on the original LineMod Dataset. On the bottom: saliency maps with the same weights, applied to our ArUco-Free LineMod dataset. From left to right: object 1, object 5, object 11.

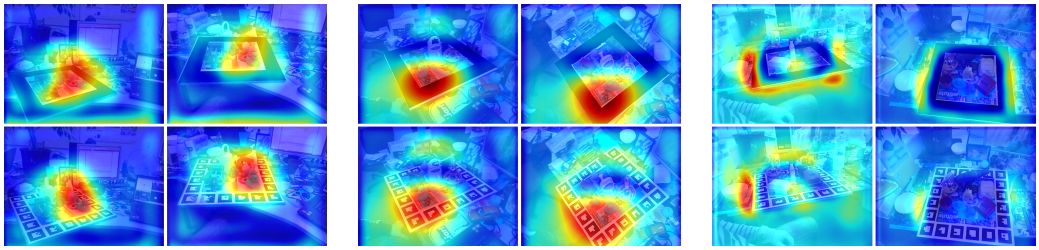


FIGURE 4.9: On the bottom: saliency maps with weights learned on the ArUco-Free LineMod Dataset, applied to the Original LM dataset. On the top: saliency maps with the same weights, applied to our ArUco-Free LineMod dataset. From left to right: object 1, object 5, object 11.

saliency is distributed in not well-defined areas. These images prove the fact that there is a background-bias during EfficientPose training with ArUco markers.

Figure 4.9 represents saliency maps obtained from our weigths, learned on the ArUco-Free LM. These results are quite different from the previous ones.

For object 1 the saliency is not on the markers, however, the network doesn't focus on the ape, looking for information in other background objects. Attention does not change a lot: it means that markers are not the central keypoints for pose estimation. Nevertheless, the network on the Original LM dataset has the worst performance accuracy.

Also, in object 5 (in the center) images saliency maps seem to remain the same for both tests.

An interesting phenomenon happens with object 11 (on the right). The most plausible interpretation is that weights on the ArUco-Free dataset predict the pose based on the square edges. When these edges are not so strongly highlighted due to the presence of ArUco markers, the prediction is wrong. For this reason, covering the ArUco markers is useful for better understanding the network, but it's not enough for obtaining a more generalized dataset.

An interesting observation concerns object 11 compared to the other two. The saliency maps for this object show quite different behavior in both Figure 4.8 and Figure 4.9. The network focuses on both marker chessboard and black square edges. A plausible motivation can be traced back to the object's symmetry. In this case, the network would not be able to estimate the correct object's pose except by learning from the background.

A remarkable result emerges from the saliency maps calculated for the model trained on the Original LM dataset. For the original LineMod images, the saliency

maps focus on both the ArUco markers and the area close to the target object, as described above and depicted in Figure 4.8 (top). In contrast, for the ArUco-Free LineMod images, the saliency maps are sometimes zeroed out. This observation can be explained by the fact that, for ArUco-Free LineMod images, some pixels were zeroed for each RGB channel. The zeroing of these pixels, which refer to the ArUco markers considered essential for the EP model, causes the gradients to be zeroed as well. It is as if these null values made that part of the hyperplane on which the gradients were calculated constant. Indeed, the gradient is zero for any constant value, as well as for a minimum or maximum point. Furthermore, for these specific images, the last activations before the last convolutional layer used for the Grad-CAM calculation have been checked and found to be nonzero.

4.3.3 Consequences

Finally, some advice for avoiding background-induced bias in 6-DoF object pose estimation are provided. Although in real-time applications it is preferable to use single-shot full-frame methods to provide 6D pose estimation of different objects and instances with a negligible increase in inference cost, testing them on real-world datasets, like LineMod, typically used to evaluate which method performs best, as detailed above, could lead to misleading results. In this case, with those datasets, networks are preferable which first detect the object, and then predict the pose of the cropped image. On the other hand, the dataset choice is perhaps more fundamental. To avoid biasing the learning method, it is important to change the background from training frames. In particular, different lights, views and background objects are helpful to overcome this issue. For example, HOPE [97], as written in section 2.2.2, presents images in different scenarios, changing also the lights. In addition, an alternative could be to change the markers to avoid repetitive appearances across consecutive frames. Furthermore, as previously stated, for validating the models, the target dataset should consist of a noticeably distinct training set and validation set.

The analysis explained is limited to one dataset, LineMod, and focuses on EfficientPose, but it is generalizable to other dataset-network of the same type: datasets that use fiduciary markers to retrieve the ground truth information, and full-RGB-based methods.

Chapter 5

Underwater 6D pose estimation

This chapter presents a novel and challenging scenario. In collaboration with the Technology Innovation Institute (TII) and Spinitalia s.r.l., the focus is on a use case involving object manipulation in underwater environments. This work examines the 6D pose estimation problem for underwater scenarios, a challenging topic that requires careful consideration.
The entire use case includes several parts. First, an autonomous Unmanned Surface Vehicle (USV) [68] travels on the sea surface, avoiding obstacles and following a specific path. Then, a Long-Range Autonomous Underwater Vehicle (LRAUV), anchored to the USV, detaches and is tasked with scanning the seabed for objects. Finally, an Autonomous Underwater Vehicle (AUV), an underwater drone, departs from the USV to approach the objects, and a robotic arm, the manipulator of the AUV, retrieves the objects.

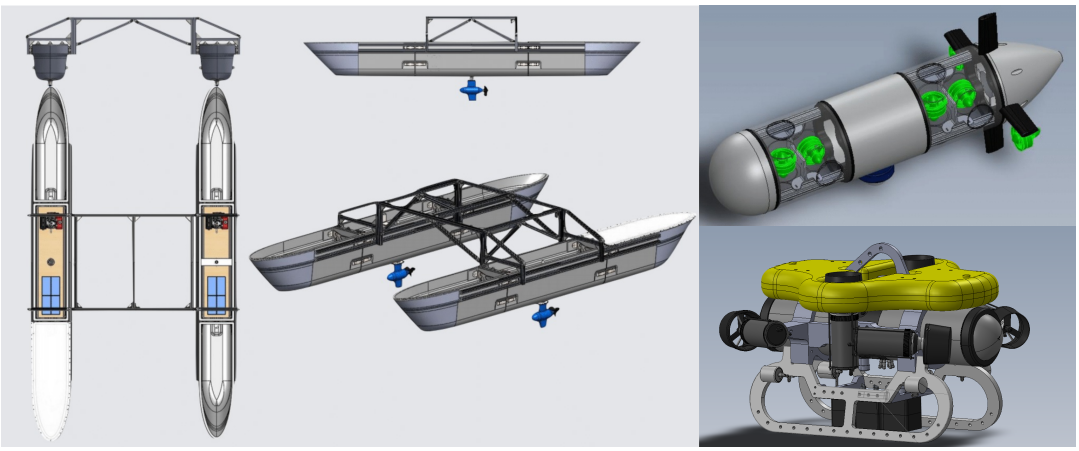


FIGURE 5.1: Underwater use case. The Unmanned Surface Vehicle (USV) is shown on the left, the Long-Range Autonomous Underwater Vehicle (LRAUV) on the top right, and the Autonomous Underwater Vehicle (AUV) on the bottom right.

This scenario requires the use of Artificial Intelligence (AI) solutions. For example, an object detector is utilized for obstacle avoidance by the USV, while a path planner is employed to adjust its optimal trajectory. Additionally, sonar sensors are used to scan and detect objects on the seafloor. Finally, the 6D pose estimation is used to identify and retrieve objects from the seabed by the AUV's manipulator.

In this treatment, only the 6D pose estimation task will be analyzed.

5.1 Problem statement

The 6D pose estimation problem, as described in Chapter 2, is one of the most recent and challenging problems in artificial intelligence. It involves predicting the 6-dimensional pose detection of an object in a 3D space, going to accurately estimate the position, orientation, and size of the object in a scene. 6D pose estimation is a challenging problem because of the complexity of the scene, the variability of the object, and the need to estimate the pose from multiple views.

Common applications of 6D pose estimation include autonomous vehicles, robotics, augmented reality and medical imaging. Various algorithms, such as deep learning and computer vision techniques, are used to accurately estimate 6D poses. These algorithms are used to detect object features in the scene, such as edges, corners and textures, in order to accurately estimate the pose of the object.

In underwater scenarios, and also in the one adopted in this chapter, the 6D pose estimation problem complexity is higher. The more difficulties regard the camera limitations, the challenging object and environment domains and the inherent limitations of the DNN models.

Object domain As described earlier, in Section 2.2.1, many challenging problems are commonly known in 6D pose estimation scenarios. In particular, the most common ones involve symmetric objects, occlusion and self-occlusion scenarios, truncated object recognition, and cluttered environments. In general, these challenging domains affect the 6D pose estimator performances due to the increased complexity introduced. An object without its discriminating parts is more difficult to recognize, such as a handle hidden behind a cup, a bottle that is symmetric with respect to its z axis, or if only a small part is visible because the object is submerged or outside of the scene edges.

The various datasets, and consequently methods, proposed in the literature attempt to address one or more of these open problems. Regarding instead other real world object domains, such as reflective and transparent objects, those without texture, and all cases where visible features are sparse, very few datasets and methods exist.

In underwater scenarios, and certainly in the one addressed in this project, most of these domains are investigated. Specifically, the objects used are textureless, monochromatic and mostly symmetrical. Due to the environmental characteristics, the objects are subject to poor visual conditions that make their recognition even more difficult. The objects can be partially submerged in the seafloor and partially illuminated. Directly irradiated surfaces are visible, while side surfaces may be in the dark. Therefore, occlusions, self-occlusions and truncations must be also managed.

Camera limitation In common datasets for 6D pose estimation, the scenarios analyzed represent industrial objects or toys placed in protected, enclosed rooms or environments. Data acquisitions are typically made with a camera circling around objects, such as in LineMod. Usually, light conditions are constant and not taken into account. However, in the underwater environment, the lighting conditions are very unfavorable and need to be taken into consideration. Two types of light sources are allowed: the natural one, the sun, which filters the water fluid, and the artificial one, placed near the camera sensor that has a limited operating range. Hence, the light source illuminates a small part of the scene, while the remaining environment

is blurred. The camera sensor must be placed in a water-proof transparent box and, once the box is submerged in water, it suffers distortion due to different fluid densities.

Environment domain As mentioned above, in common 6D pose datasets, the heterogeneity of the environment background is not taken into account. Generally, the background can be considered constant; it is almost the same throughout the dataset. Depending on the dataset and method, in some cases it may help to generalize the scenarios investigated. It is precisely the example of LineMod and EfficientPose, where a more diversified environment should have helped the method to generalize. As shown in Chapter 4, this is not the case. Other methods, instead, that use the image crop of the target objects, can not benefit from this attempt to generalize. In real underwater scenarios, however, the background could play an important role because of its high variability. The underwater environment can be very heterogeneous. In some cases, the seabed might be rough or agitated, reducing visibility and compromising the frame quality. It can be not visible because it is too deep, simulating a dark scenario. The seabed can be sandy or have marine flora, which greatly diversifies the background. The water can be clear or cloudy. Some floating particles may occlude objects or be reflective.

Method's challenge The ideal method for our scenario should cover all the aforementioned domains, regarding the object, light conditions and environment constraints. The Deep Neural Network should be robust enough and reach high performances with low-resolution and low-quality RGB images. Feature-based methods are not applicable because they are more sensitive to image quality, and full-frame approaches are preferable because they are agnostic to the number of objects in the scene. Due to the textureless objects, poor light conditions and intrinsic environment characteristics, the low quality of the visual features compromised most of the known methods.

These are the reasons that make underwater scenarios, and then this project, so interesting.

In the following, in Section 5.2 an overview of the existing data collection methods in underwater tasks is explained, then the considered objects are presented, and subsequently, the newly created datasets are discussed. In Section 5.3 the methodology choices regarding the 6D pose estimator are motivated. Finally, in Section 5.4 the experimental results are shown. A method comparison is done. The YoloV4 and Augmented Autoencoder performances are detailed for quantitative and qualitative results, and the on-edge performances are presented.

5.2 Datasets

In order to create a dataset for training and testing a pose estimator, in addition to the usual best practice regarding balancing on classes, background generalization and general case coverage techniques, in underwater scenarios some domain-specific constraints should be considered. For real data acquisition, there are some limitations related to the environment itself, such as the location of objects, their records, and label annotation techniques. To create a real dataset for the 6D pose estimation task, the frame RGB or RGBD must be associated with the corresponding

6D labels. The labels must describe both the translations and rotations, a very expensive piece of information to collect. Real data acquisition and also data labeling is complicated, and in some cases impossible. In a controlled environment, such as a swimming pool or tank, where the working conditions are easier, real data labeling can be also possible. However, in other real scenarios, like a real lake, they are more complicated and, consequently, a more general acquisition method should be considered. Therefore, since underwater data acquisition presents many difficulties, so various types of sensors are used and many state-of-the-art ones have been recently developed to facilitate data extraction. Collecting depth information requires the use of specialized devices such as stereo cameras, LiDAR, lasers, SONAR, or fusion sensors that combine different types of sensors.

As mentioned in Section 2.2.2, public datasets are available for the 6D pose estimation problem, and many of them are based on RGB images. To obtain more insights about the object's pose, particularly for the third component of translation, depth information can be used. Unfortunately, there are few real-world datasets that handle 6D pose estimation in dry environments and almost none for underwater environments. In addition, there is a lack of trained models for submerged objects, as they require data collection in hard-to-reach locations and can be difficult to perform. In underwater environments, only a few works, described in the next section, have been published on dataset generation or real-data collection.

5.2.1 Existing data collection methods

Autonomous object manipulation in underwater scenarios is a very complex task, still little studied, but with a growing interest in recent years. The first step is definitely to create datasets for this task, and indeed the need for underwater datasets has been addressed by some research.

Jeon et al. [39] recognized the challenges to obtain underwater labeled data. Acquiring underwater object data is more complex than acquiring data from ground environments. Even when the data is gathered, manual annotation is expensive and errors due to human mistakes are inevitable. Additionally, underwater camera images should have various variations including intensity degradation and color distortion. Jeon et al. proposed an approach for making a synthetic dataset using a 3D CAD model. Their approach automatically annotates view points, bounding boxes, and segmentation labels for use in underwater environments. They also introduced a simple deep learning pipeline to validate their data generation method, achieving promising results. This pipeline consists to apply a Mask R-CNN network, which detects the object's mask, bounding boxes and class, and then a simple orientation estimation network, combined with DenseNet [36], Dense block and Fully Connected network, is applied to extract the quaternion representation of the object's orientation from the cropped images resulting from the Mask R-CNN.

Billings et al. [3] introduced a fisheye image dataset, called UWHandles, collected in natural underwater environments, that contains three types of grab handles from different natural environments. UWHandles has both 6D object pose and 2D bounding box annotations. AprilTag fiducials were used to obtain ground truth camera poses in the image sequences. The data were collected by placing a remotely-operated vehicle (ROV) on the seafloor near the objects and dispersing the fiducial markers in a manner similar to the method used in this letter. The handles also had fiducial markers attached to mounting plates at the base of the handle.

Previous research efforts are still insufficient to address the 6D pose estimation problem in a general way. Such research is very specific to subdomains of the problem.

Our scenario's goal is to develop a training dataset for CAD and RGB sources, and then evaluate the results on RGB source only, to predict multiple underwater objects' poses. It is therefore necessary for the training dataset to be both diverse and accurate. To collect the data, getting also the depth information, a stereo camera is used.

5.2.2 Objects of interest

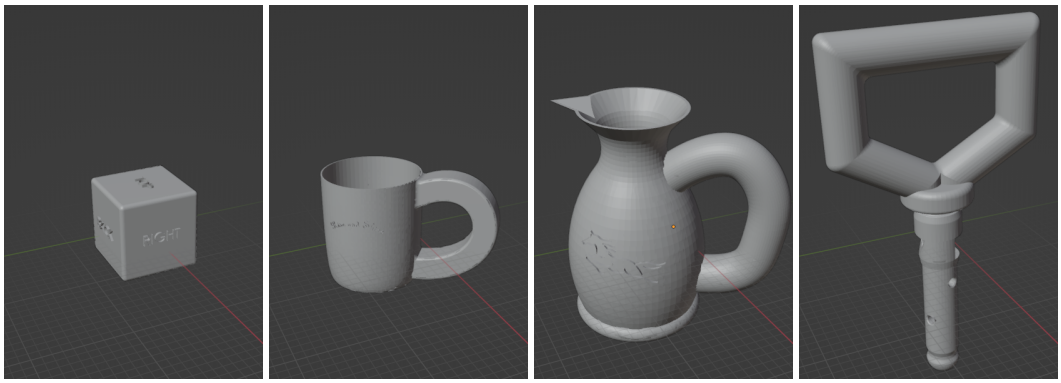


FIGURE 5.2: 3D objects: box, cup, jug and hotstab.

In the studied scenario, four objects are considered, depicted in Figure 5.2. The chosen object categories represent common use cases for underwater intervention tasks. The box, hotstab and cup are considered symmetrical, while the jug is not. Symmetry means that there are two or more equal views that correspond to different poses (and clearly distinguishable) of the object itself. The hotstab is symmetrical for each pose and for its corresponding pose to 180° over its z axis, the box is symmetrical for 360° in every its axes, and the cup is only symmetrical for those angles of the x, y, z axes where the handle is hidden. In a second version of the object models, an embossed text is placed on some objects surfaces, for cup, jug, and box objects as depicted in Figure 5.2. The text, of the same color and material as the objects, should help the 6d pose estimator distinguish symmetry.

The objects were printed on a 3D printer with a gray and rather dark matt plastic material. Hence the objects can be considered textureless.

	diameter	min_x	min_y	min_z	$size_x$	$size_y$	$size_z$
box	183.5974	-51.18766	-54.2683	-59.69242	106	106	106
cup	291.7259	-111.3848	-65.5387	-72.14373	219.7151	126.8602	143.9991
jug	440.0625	-141.406	-83.63377	-156.1391	286.3683	177.3268	283.202
hotstab	542.8955	-56.0085	-125.035	-247.5889	111.9975	250.6408	468.3709

TABLE 5.1: Objects information expressed in mm.

The four objects have different sizes, listed in Table 5.1, to cover scenarios with different object shapes, which pose a challenge for deep learning methods.

Among the 4 objects under analysis, the box is the only convex one, while the cup, jug and hotstab are concave. A solid is called convex if, given two points on its surface, all points of the two target points' conjunctive segment remain within the

solid. This is the case for the box object. For others, however, such as the cup or jug, some elements such as the handle make them concave: taken one point on the handle and one on the main building, some points of the conjunctive segment can be outside the object.

Intuitively, considering a pose invariant metric, a convex object can be considered as more simple one than a concave object and it might be more easily distinguishable in the pose. Depending on the benchmark metrics, a convex object could achieve better performances, as shown in Section 5.4.3. In our scenario, a further distinction can be made in order to categorize objects as more or less distinguishable by pose.

Considering a simple object like a box, where the three axes' shapes are equal, and for example the COU metric, defined in 2.2.3, where only the segmentation masks are considered to evaluate the goodness of a predicted pose, any object rotation on any axis produces nearly the same segmentation mask. On the other hand, for a more complex object, like a hotstab, a little rotation variation produces an equal variation in its corresponding mask.

In other words, given an image and the object selection rectangle, pose invariant metrics achieve greater results when more pixels of the cropped image belong to the object than to the background.

From a practical point of view, a new measure will be useful: the 6D-sphericity degree. Given a target object and the smallest sphere in which that object is completely inscribed for each of its possible rotations, the object's 6D-sphericity is the ratio between its volume and the sphere's volume. Intuitively, if the target object is a sphere, the 6D-sphericity will be one. For the considered objects, the box, cup, jug, and hotstab, respectively, have decreasing values of 6D-sphericity.The COU metric will reach higher values the more the object is 6D-spherical: the segmentation masks will be similar.

5.2.3 Proposed dataset

Our dataset comprises RGB-annotated frames of four different object categories, seen from different points of view, alongside the 3D models. The chosen object categories, described above, were selected to reflect common use cases for underwater intervention tasks. This dataset is meant to continuously grow with various object representations that are useful for underwater intervention purposes and to foster the development of robust underwater object pose estimation methods.

The dataset consists of synthetic and real data:

- Data generation: starting from the Computer-Aided-Design (CAD) models, the RGBD images have been generated,
- Data acquisition: using 3D printed objects, records of several environmental types have been collected.

Data generation Due to the higher cost of real-world data recording, to overcome the low data heterogeneity, intended both for sufficiently large data of color intensity and distortion, and for variation with respect to the underwater environment type, synthetic data play a central role to try to apply deep learning techniques.

Through synthetic data, we potentially can generate infinitive images, applying all the data augmentation techniques, while ensuring the exact ground-truth label.

Specifically, using Unity, we are able to generalize between different background images, each of which is augmented by several techniques:

- Regarding the light, it is possible to add light distortion to simulate reflections, change the light source's position, change the light breadth and intensity. Color lights change randomly, with values chosen in specific ranges to avoid black lights. In detail: a light color is defined by hue (in a range of (0, 265)), saturation (in a range of (0, 128)) and value/brightness (which is constantly equal to 1). There are $265 \times 128 \times 1$ possible light colors.
- Regarding the background plane, it is possible to rotate, translate, and tilt. It is also possible to switch between 2D and 3D backgrounds. For the 3D backgrounds also the proximity and amplitude of terrain gradients can be changed, while for the 2D one a similar effect can be reached with a simply visible effect. The background plane can be placed at various distances with respect to the point of view. Different homogeneous and heterogeneous backgrounds can be also simulated.
- Regarding the objects, it is possible to force them to be inside the scene, not occluded and not submerged. On the other hand, also occlusions, truncations, and background immersions are permitted. The objects can be placed onto the background or can be fluctuating. Moreover, the objects can be placed at different distances with respect to the camera position, they can be placed at the same distance between themselves, or at different distances to simulate the perspective. The objects can have shadows.

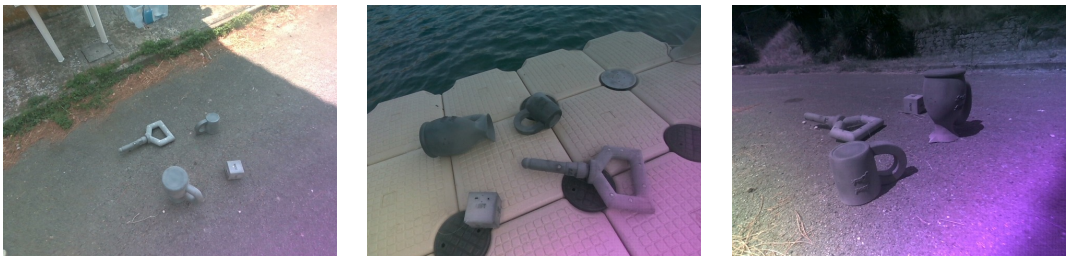
Some of the results are depicted in Figure 5.3.

FIGURE 5.3: These are some examples of underwater Unity-simulated scenes.

Data collection Even though the simulated dataset reaches promising results, to improve the performances of an artificial neural network (ANN), the real-world samples are usually determinants to better learn and perform in real-world domains.

3D-printed objects made it possible to harvest data in realistic scenes. The dataset is collected using a d455 realsense camera inside a waterproof container, designed by Spinitalia s.r.l., which simultaneously captured color and depth images at 640×480 resolution at 30 frames per second. Video sequences have been recorded for each of the four object categories in 10 different setups, both individually and in

various category combinations. The different scenes have been recorded across two countries, in a variety of real environments both underwater 5.4b and dry 5.4a.



(A) Real world underwater scenes.

(B) Real world dry scenes.

FIGURE 5.4: These are some examples of real images acquired in different scenarios. (A) images present dry backgrounds: on the left objects are on the asphalt, on the centre they are on a pier, on the right they are on the asphalt during night. (B) images present underwater backgrounds: they are all acquired at Nemi Lake (RM) with different degrees of depth.

In our underwater scenes, challenging visibility conditions expected at sea are considered, including object occlusions, heterogeneous backgrounds, shadow or direct light conditions, natural or artificial light, and night recordings. Figure 5.4 provides examples of real scenarios. For each recording, the object has been freely rotated within the camera's field of view at varying distances up to $3m$ from the camera.

To obtain the ground truth of the object pose, an AprilTag [66] bundle has been designed, which maps to the geometric center of the object of interest.

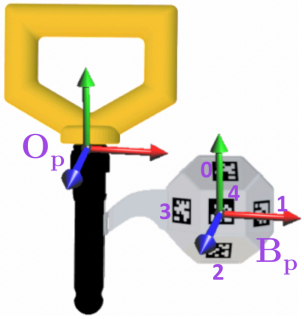


FIGURE 5.5: Example of object pose ground truth estimation.

An example is illustrated in Fig. 5.5. Namely, given that the object base frame by its 3D model, O_p, was known and the bundle pose, B_p, was detectable, the transform between the object and the bundle, T_B^O, was calculated offline.

The AprilTag presence in a scene permits to derive the ground truth pose of the object attached to it, thanks to methods that can detect the tags and determine their pose.

The scene acquisitions have been made both with and without the AprilTag.

For the dataset with the AprilTag, as just described, the exact translations and rotations of the depicted objects are associated with each image. In this way, a real 6D pose estimation dataset can be collected. However, the AprilTag presence in the scene could alter some deep learning methods' performances, as described in Chapter 4. Moreover, the AprilTags are applied to the object's appendixes. The objects are then altered in shape and size. However, in order not to use the AprilTag bundle, if the 6D labels are set in a human full-hand method the risk of bad label data is very high.

On the other hand, the dataset without AprilTag is useful both to create a real test set, and to overcome the possible aforementioned problems due to the AprilTag presence in the scene. Such test set is able to perfectly simulate a real scenario, in which the AprilTags are not present. The AprilTag-less datasets will not be annotated with the translation and rotation ground truth information. The real-world acquisitions, despite are not 6D pose labeled, are however useful to create a real dataset for the 2D detection tasks. With automatic 2D labeling tools, widely used in recent years, such as LabelIMG [1], and the simple annotation procedure of such information, optimal labels can be obtained from real-world images.

In our scenario, three stages have been considered in the 2D annotation process. First, approximately 1% of the data across scenes was labeled by a human annotator. Second, an auto-labeling tool, based on Yolov4 [4], was designed and used to find the bounding boxes for the remaining data through the 1% of manually annotated data. Finally, to ensure the quality of the annotated region of interest, three different human annotators checked the created bounding boxes in a sequential fashion. The real-world 2D annotated dataset has 87,100 real RGB images.

This type of dataset is especially important for dealing with an unexplored or semi-explored domain, such as underwater. A specific data format would be required for any of the 6D pose estimation methods investigated, even at the experimental stage alone. On the other hand, with a dataset labeled for 2D detection tasks, all the 2D detectors and other methods based on the same type of dataset, can be easily approached.

5.3 Methodology choices

To address the problem of 6d pose estimation in underwater scenarios, from the method's point of view, it is first necessary to distinguish and identify the problems to be overcome. As mentioned before, in underwater environments some light conditions make more sensitive deep learning models. These concern both environment's intrinsic characteristics and the camera limitations, which result in low quality images. The chosen image resolution is 640×480 to meet the latency constraints: the whole 6D pose pipeline should be run onto an embedded board placed on the AUV. Low image resolution afflicts deep learning methods even more.

Moreover, 3D object models without textures and their characteristics of symmetry, occlusion, self-occlusion and truncation make learning the pose more arduous. For these reasons, the adoption of the major part of existing methods is not the best choice.

[1] https://github.com/heartexlabs/labelImg

Feature-based approaches would risk very low performance and other methods might be more suitable. To overcome the problem listed before, we chose to adopt a holistic two-stage method, based on a YoloV4 [4] for the object detection task and an Augmented Autoencoder (AAE) [88] for the rotation estimation.

YoloV4: object detector Regarding the object detection task, and hence the choice to use the YoloV4 detector, the motivations concern the results shown in Chapter 3. YoloV4 is a network that performs very well in heterogeneous environments, with an excellent tradeoff between accuracy and latency. In the underwater scenario under analysis, the need of selecting a real-time and rather lightweight deep learning solution is a requirement imposed by hardware constraints. Indeed, YoloV4 allows the desired FPS requirements to be achieved while at the same time respecting the power constraints. In underwater scenarios, with the robotic arm mounted on the AUV, physical space is right limited. In this case only a compact embedded board is allowed.

Another reason for choosing YoloV4, and also for dividing the whole task into two different tasks (object detection and rotation estimation) is due to the ease of the OD data collection procedure. For object detectors, which are more investigated networks, there are several automatic labeling solutions, and yet even for manual labeling methods, the definition of only the 2D bounding boxes containing an object is easier and less prone to human error than the full 6D labeling task. 2D labeling is also cheaper and therefore more data can be collected.

Moreover, in underwater light-poor conditions and noisy acquisition conditions, the YoloV4 performs better than other singe-shot methods. YoloV4 permits to distinguishes also both near and far objects, for both sunny and shadow environments (see Table 5.2 the mAP values).

Although there are other methods that are newer and better performing than YoloV4, an already well-known network has been chosen. Often the installation and the first use of new deep methods require a lot of initial effort. Methods are not yet established and may hide some bugs. Additionally, some methods, such as YoloV6 and YoloV7, have not yet been published by the time the pipeline was consolidated. For YoloV4, the training process is well-defined and supported by OpenCV for inference. In our case, the detector has never been the bottleneck, and for only four objects there is no need to improve performance. The mAP achieved is already close to 1, so there is no need to further enhance performance.

Augmented Autoencoder: rotation estimator For the 6D pose task, the choice of AAE is driven by similar reasons. The authors [88], in addition to the method, also introduced the Domain Randomization (DR) strategy. DR consists of training a model on rendered views with several data augmentation techniques. This data generator makes it possible to train the network without real data, and instead use only the 3D CAD model. During the network's pre-processing stage, the CAD model is used directly to create a training dataset. Therefore, a labeled 6D dataset is not necessary, which facilitates its use.

Assuming to have sufficient 6D annotated data, the applicability of other well-known 6D pose methods could not be safe. As described in Section 2.2.4, the features-based methods need to extract the 2D-3D correspondences from the image and 3D model. On one hand, the images theoretically should be captured with ad high resolution, in constant light conditions, and with a fine-graned characterization regarding the depicted object of interest. On the other hand, instead, the 3D model used

should have the texture associated with it. The texture permits giving more visual details of the objects, such as color, shadows and shapes. With texture-less objects instead, the detail accuracy degree is not sufficient to apply this kind of method.

In Domain Randomization, the considered data augmentation techniques are random lighting conditions, different backgrounds, different levels of saturation, contrast, hue, square occlusion, and gaussian blurring. Such training permits to generalize on different backgrounds and also to real images, meeting the project requirements on the environment generalization. In our case, we specified to use custom real background images in addition to VOC images to better fit our scenario. Based directly on the CAD model, even if 3D printed objects have poor visual features, AAE is able to fit our scenarios better than other methods. In addition, the method directly addresses the problem of symmetries, occlusion and self-occlusion. This is due to the property of representing the orientation not on a fixed parametrization, but on the appearance of an object. The disadvantage of this choice is that the AAE does not scale with respect to the number of instances in the scene, but it still achieves good results with the required input resolution while maintaining relatively high quality.

Utilizing pose refinement techniques or other post-processing methods would not produce satisfactory outcomes due to the absence of reliable visual characteristics. The ICP algorithm, which depends on depth data, is not feasible in this scenario owing to the inherent constraints of the sensors in capturing adequate depth information in an underwater setting. Poor depth data will generate an untrustworthy point cloud.

Other holistic methods, both for one-shot and two-stage methods, require a real-world annotated dataset and may suffer from instability. EfficienPose(EP), for example, was tested in a preliminary phase but did not obtain satisfactory results. This may be explained by the findings in Chapter 4. In our scenario, several factors make the EP unsuitable for 6D pose estimation. The distance range of the objects varies from a few centimeters to 3 meters, which is significantly different from the LineMod dataset where the objects are close to a restricted distance range. Additionally, the objects appear in all corners of the scene, rather than being close to the center of the image as in LineMod. Furthermore, the objects are textureless and symmetrical, unlike the majority of objects in LineMod which are colored and convex. Lastly, there are no other useful objects or visual tags present that could help the 6D method distinguish the poses, thus making EP unsuitable.

The YoloV4 and AAE pipeline adoption came out from a comparison with other two state-of-the-art methodologies: EfficientPose and Yolo-6D. The results are shown in Section 5.4.1.

5.4 Results

In this Section, the experiments and results will be discussed. To evaluate the goodness of a model, some metrics explained in Section 2.2.3 and Section 2.1.3 have been adopted. In particular, the ADD and ADI metrics are used to evaluate asymmetric (jug) and symmetric (box, cup and hotstab) objects respectively. For both these metrics, the estimated pose is considered correct if $e < \theta = k_m d$
where k_m constant is generally equal to 0.1, d = object diameter. In Section 5.4.3 three different k_m values will be considered: $k_m = 0.1, 0.2, 0.3$. Regarding the detection metric, the AP metric is used in order to evaluate the goodness of the detection and classification parts. Due to the ambiguity of the considered objects, also the COU

and VSD metrics are taken into consideration. The COU error function is ambiguity invariant; however, since it relies on the projection of the model, it provides only limited information about the accuracy of the object surface alignment. In this direction, also the VSD error function can be used with ambiguity poses. It works with the 2D masks of the object model's visible part in order to compare directly pixels.

5.4.1 Method comparison

In this Section a DNN model comparison is proposed. Three different neural networks are chosen to cover the range of methodologies present in the state-of-the-art that differ both in problem approach and learning strategy:

- EfficientPose [6]. It has been chosen because reaches the SOTA performances in 6D pose estimation problem [2]. As described in the previous Chapters, it is a full-frame, single-shot method. It is multi-instance and multi-object invariant, as the number of objects in the image increases, the computation time does not change.
- Yolo-6D [95]. As described in Section 2.2.4, Yolo-6D has a feature-based approach to firstly detect the 3D bounding box vertices from the image, and secondly, apply the PnP algorithm [48] to estimate the object pose in $O(n)$. The network is multi-instance invariant for the 3D detection task. It requires a custom labeled dataset, but its format is similar to that of darknet (Yolo family).
- Augmented Autoencoder [88]. It is a one-shot method for the rotation estimation task that works on cropped images. It does not scale with respect to the number of objects and instances in the scene. On the other hand, it does not require a custom labeled dataset and directly addresses the symmetry and occlusion problems. In this case, YoloV4 has been chosen to perform the 2D object detection task and extract the cropped images for the objects' regions of interest.

The results shown below are referred to only one object class dataset in order to make a fair analysis. Indeed, the AAE is trained on a single object instance. In multi-object scenarios, as described before, AAE does not scale with the number of objects. EP and Yolo-6D instead could work also for multi-class dataset, but to compare with AAE they are trained on the single target object. The chosen target object is the hotstab, which is the most significant one in underwater scenarios between the available objects.

The method comparison is made on three different datasets, each of them referring to an approach. EfficientPose needs to have a dataset in a LineMod-like format, with a specific directory structure and with specific ground truth information. Yolo-6D instead, needs to have labels in a darknet-like format. With respect to the YoloV4 label, Yolo-6D needs to have a ground truth with the 3D bounding box corners. Finally, Augmented Autoencoder does not need any kind of dataset, but it directly generates the training images from the CAD model. The datasets used in EfficientPose and Yolo-6D are generated with Unity with the same degrees of freedom, regarding distance range (from 50cm to 3m), light condition range, and all the techniques described in Section 5.2.3. For AAE the VOC images are used as background in the Domain Randomization strategy, instead of a generated dataset with

[2] https://paperswithcode.com/sota/6d-pose-estimation-on-linemod

Unity. However, for the YoloV4, used for the 2D object detection sub-task, a similar dataset generated in the same way as the other two methods is used.

Although the three methods are trained on these different datasets, they can be considered comparable thanks to the same generation procedure used to create the corresponding dataset. Moreover, the methods are evaluated on the same validation dataset on which the metrics are computed.

Setup EfficientPose is trained on a dataset containing 2500 images, for 500 epochs and a learning rate fixed to $1e^{-4}$. The hyperparameter Φ has been set to zero, in order to use the most lightweight network version of the EfficientDet backbones family.

Yolo-6D is trained on a dataset containing 2500 images for 10000 epochs with an initial learning rate of 0.0001 and batch size of 32. The learning rate has been changed in $0.001, 0.0001, 0.0001$ for the steps $1000, 60000, 70000$.

YoloV4 is trained on a dataset with 10000 images for 50000 epochs. The batch size has been set to 64 with a subdivision of 64. The learning rate starting from 0.001 has been decreased to a factor of ten for the steps $20000, 25000$.

Augmented Autoencoder is trained for a render resolution of 640×480 in order to decrease the network parameter, using directly the CAD model and generating 20000 training images through the Domain Randomization strategy

The results shown below are referred to only the generated-like evaluation dataset and not to the real-world one.

	ADD	ADI	mAP
EfficientPose	1.32%	8.65%	0.7521
Yolo-6D	9.58%	36.71%	0.77
YoloV4+AAE	**11.4%**	**44.0%**	**0.99**

TABLE 5.2: The table shows the results of EfficientPose, Yolo-6D and YoloV4+Augmented Autoencoder pipeline in terms of ADD, ADI and mAP metrics for the hotstab object. The ADD and ADI are computed with a threshold $k_m = 0.1$. The mAP is computed with an Intersection-over-Union threshold of 0.5.

Comparison In Table 5.2, the results of ADD, ADI and mAP metrics are reported. The considered object is partially symmetric, hence the ADI is the most significant metric compared to the others. Also ADD is reported for comprehensiveness. ADD and ADI describe the estimation performance of network laying, but also marginally considering the detection part. The mAP metric is referred to only the 2D detection part: the detection subnetworks for EP, the 2D detection deriving from the 3D BBs projection for Yolo-6D, while in the YoloV4 and AAE pipeline is referred only to the YoloV4 network. A good 2D detector could improve the final pose estimation performance. The considered threshold in ADD and ADI computation is $k_m = 0.1$. This threshold is referred to the 3D model's diameter at which a 6D pose is considered correct. If the average distance between the 3D model points transformed with the GT pose and estimated pose, respectively, is lower than this threshold the pose is considered to be correct. The threshold used in mAP computation instead is referred to the Intersection-over-Union threshold of 0.5 computed in all precision-recall function points. See details explained in Section 2.1.3.

YoloV4 and AAE pipeline reaches the best porfomances across all metrics, compared with the others two methods. This indicates that the YoloV4's 2D object detection is extremely accurate and provides the AAE with well-cropped images that

accurately encapsulate the objects, allowing it to accurately predict the objects' pose. On the other hand, EfficientPose and Yolo-6D reach lower mAP values and as the accuracy in object detection decreases, then the pose is also more difficult to predict. The motivation can be various:

- the EfficientPose model chosen has the $\Phi = 0$ hyperparameter. This architecture is the most lightweight among all EfficientDet models and it reaches lower performances compared to the others;
- in contrast, Yolo-6D is based on YoloV2, which achieves much lower performance than its descendant YoloV4.

The large gap in ADI values between EfficientPose and Yolo-6D, despite the fact that the mAP values are very similar, can be attributed to the experiments shown in the previous chapter. As mentioned in Chapter 4, EfficientPose performs very well on the LineMod dataset because, in addition to the fact of ArUco presence, the target objects are placed near the image center and clearly distinguishable, and the distance range between the camera sensor and the target object is more restricted than in our case. Such good performance given by the use of a simplified dataset covered all EfficientPose's subnetworks, but mainly the rotation one, which is why that subnetwork was analyzed in the previous chapter. Indeed, looking at the results it can be seen that the BBs are positioned correctly in the images, with quite correct translation, but rotated incorrectly (top-left in Figure 5.6). An example of EP, Yolo-6D and YoloV4+AAE predictions is shown in Figure 5.6.

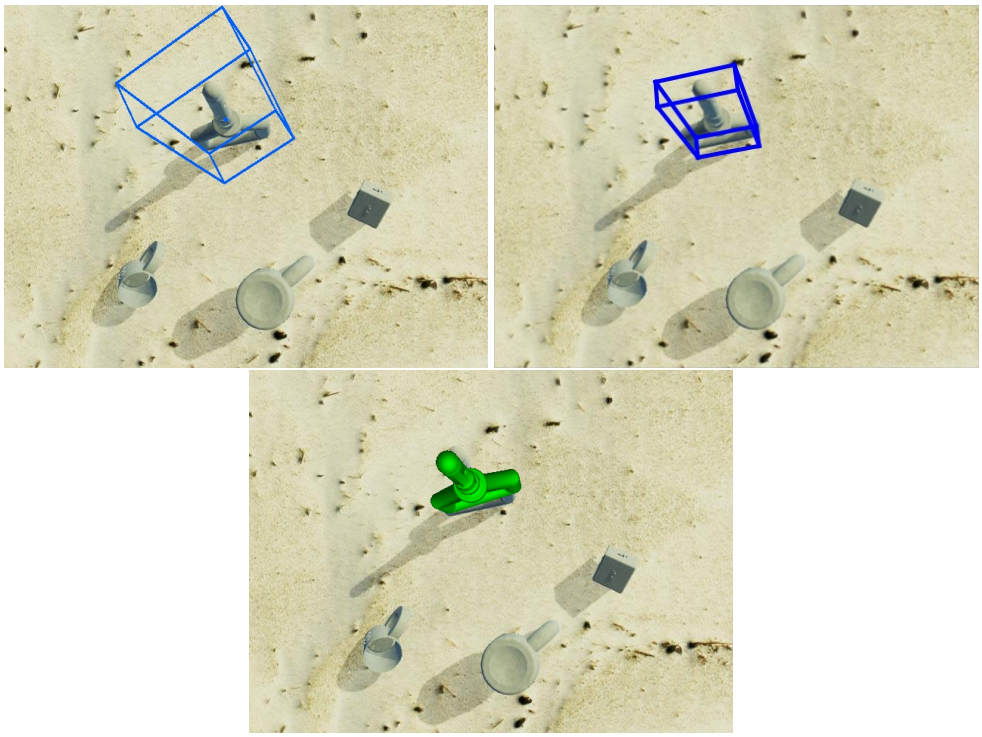


FIGURE 5.6: Methods comparison results. At the top left, the prediction of EfficientPose is displayed, Yolo6D at the top right, and YoloV4+AAE at the bottom.

In addition, looking at both Table 5.2 and Table 4.4, it can be observed a 20% decrease in mAP values, justifiable by the increase in problem complexity, and a more

90% decrease in ADI values. The 90% decrease represents the low representative power of the EP's rotation subnetwork.

On the other hand, the small gap in ADI values between Yolo-6D and the AAE-YoloV4 pipeline can be motivated by the PnP method used in pose estimation for the Yolo-6D network. The dataset used to calculate the metrics is a dataset generated with Unity. Although some data augmentation techniques have been used to generalize the synthetic dataset to reach the quality of the real underwater dataset, it is quite clean compared to the real scenario. Thus, in this case, the PnP algorithm achieves good performance, but in a real-world underwater scenario, where the quality of frames decreases, a drop in performance is expected.

The YoloV4-AAE pipeline results shown in Table 5.2 are collected without any kind of post-processing. In AAE, the native use of ICP refinement is planned to improve the pose estimation. In our scenarios, using a d455 realsense in an underwater environment, the depth information collected would be of poor quality, further invalidating the pose estimation itself. For this reason, not only the result didn't improve, but also the prediction took more time and, then, efficiency decreased.

5.4.2 Experimental setup

In the following, only the YoloV4-AAE pipeline results are reported. The adopted solution is composed by:

- YoloV4. It is trained for multi object detection to cover all four objects simultaneously: box, cup, jug, and hotstab. The dataset is composed of 10000 images for the training set and 2500 images for the validation set, generated via unity with sandy background and with the data augmentation technique described in Section 5.2.3. Every image contains the four objects in the scene with their BB ground truths. The network hyperparameters used are: an input network resolution of 512×512, batch size and subdivision at 64, learning rate of 0.001 with a decreasing scale factor of ten in steps $20000, 25000$, maximum number of epochs of 50000.

- Augmented Autoencoder. One network for each object model, for a total of four distinct AAE models. Each model is trained for a render resolution of 640×480 in order to decrease the network parameter and meet the physical latency constraints. The training is made using directly the CAD model and generating 20000 training images through the Domain Randomization strategy starting from VOC images as a background. For each object model, an independent exploration of the best hyperparameters has been carried out to customize training on the target object and achieve the best performance for it. AAE training presents many hyperparameters, as written in the paper [88]. They could be divided into two groups: the ones which modify the structure and optimization of the network (learning rate, latent space dimension, batch normalization, etc.) and the ones which act on the data augmentation (occlusion percentage, inversion, multiplication, drop, gaussian blurring addition, etc.). Because of the large number of hyperparameter configurations, it is not possible to explore all of them, so heuristics are needed to choose only the most promising configurations. The results are shown in Table 5.3.

-	Box	Cup	Jug	Hotstab
Data Augmentation Hyperparameters				
Perspective Transform				
Crop And Pad	✓	✓		✓
Affine	✓	✓	✓	✓
Coarse Dropout	✓	✓		✓
Gaussian Blur			✓	✓
Invert	✓	✓	✓	✓
Multiply	✓	✓	✓	✓
Contrast Normalization			✓	✓
Square Occlusion	0.6	0.4	0.4	0.4
Architecture-Hyperparameters				
Learning Rate	$2e-4$	$2e-4$	$2e-4$	$2e-4$
Optimizer	Adam	Adam	Adam	Adam
Latent Space Dimension	256	256	128	256
Epochs	50000	70000	70000	70000
Batch Size	32	32	64	64

TABLE 5.3: The chosen hyperparameters for each object's training.

Objects	$k_m = 0.1$		$k_m = 0.2$		$k_m = 0.3$	
Box	ADD 1,0%	**ADI** 21,6%	ADD 1,4	**ADI** 51,0%	ADD 2,0	**ADI** 60,4%
Cup	ADD 19,8%	**ADI** 55,0%	ADD 37,2	**ADI** 77,8%	ADD 49,2	**ADI** 85,8%
Jug	**ADD** 29,2	ADI 72,2%	**ADD** 56,6	ADI 86,6%	**ADD** 68,6	ADI 93,0%
Hotstab	ADD 11,4	**ADI** 44,0%	ADD 40,4	**ADI** 64,4%	ADD 56,0	**ADI** 73,8%
Average Perc.:	ADD 15,35	ADI 48,2%	ADD 33,90	ADI 69,95%	ADD 43,95	ADI 78,25%

TABLE 5.4: Table of recall percentages based on the err_{ADI} and err_{ADD} for different threshold values k_m.

5.4.3 Performances

Quantitative results In the considered datasets, three objects (box, cup and hotstab) present symmetric views, while one object (jug) is asymmetrical, therefore e_{ADI} is chosen for the first three objects and e_{ADD} is chosen for the last one. Results for both e_{ADI} and e_{ADD} with different thresholds of correctness are represented in Table 5.4.

As described in Section 5.2.2, the box is symmetrical for 360° in each of its axes, the cup is only symmetrical for those angles of the x, y, z axes where the handle is hidden, and the hotstab is symmetrical for each pose and for its corresponding pose to 180° over its z axis. In other words, the box is completely symmetric on all axes, the hotstab is symmetric only on the z axis, and the cup is symmetric only for those rotations where the handle is not visible. In this order, moving from the most symmetrical object to the least symmetrical object, even varying by threshold k_m, gives increasing ADI results. For the jug, however, being an asymmetrical object, the ADD metric has been used. In fact, its ADD results are higher than the others.As written

in Chapter 2, ADI yields relatively small errors even for views that are distinguishable, and is thus more permissive than ADD, ADI is in fact the lower bound of ADD as written in [31]). The objects evaluated with ADI are therefore advantaged.

The limitations of these metrics are that they are not pose-ambiguity invariant. Consequently, the four taken objects are very disadvantaged, since they present many ambiguous poses. This is an unusual dataset, due to simple objects which complicate the problem. For objects such as boxes, cups, and hotstabs, this is due to their symmetrical nature. For instance, the box object has three symmetrical axes and it is texture-less. This makes it difficult for the network to distinguish one face of the box from the other. In the second version of the object model, a unique texture has been placed on each face of the object. For example, the box has inscriptions on each of its faces, which are the same color as the object and are not enough to differentiate the pose of the object itself. The jug, on the other hand, not being a symmetrical object, is easily distinguishable in poses when the handle is visible. For poses in which the handle is hidden, only the spout of the jug can be used to distinguish the pose, making the task more arduous. Despite these disadvantages, the AAE is able to predict a good rotation, which allows the object to overlap with the one depicted in the image, but not with the correct object face.

Objects	COU	VSD
Box	94.6%	24.4%
Cup	94.2%	32.6%
Jug	79.6%	45.2%
Hotstab	20.2%	10.8%
Average Percentages:	72.15%	28.25%

TABLE 5.5: Table of COU and VSD recall percentages based on the err_{COU} and err_{VSD} respectively. In both metrics the threshold θ is set to 0.3.

To give a correctness indicator of visual object matching, other two metrics can be considered. In Table 5.5, the Complement over Union (COU) and Visible Surface Discrepancy (VSD) results are presented.

The COU is based solely on object segmentation masks and does not take into consideration the poses or distances of the objects, unlike ADD and ADI metrics. Intuitively, regardless of the pose, a convex object will be able to obtain higher COU recall values than a concave object. As defined in section 5.2.2, following our definition of 6D-sphericity, the greater the object's 6D-sphericity, the more likely it is to have a higher COU value. Given a cropped image, derived from the predicted bounding box, and considering that it depicts an almost 6D-spherical object, then the main part of the cropped image will refer to the object. The COU metric, working on the object's segmentation mask and not considering the pose at all.

For example, the box, which is an almost completely 6D-spherical object with respect to those under analysis, achieves the higher COU recall value, nevertheless, as shown in Table 5.4, its ADD value is very small (with $k_m = 0.1$ the ADD values is 1%). The AAE is unable to predict the correct pose, but, as mentioned before, is able to predict a good rotation, which allows the rendered object to overlap with the one depicted in the image, but not with the correct box face. Even the cup and jug objects, minus the handles, can be considered roughly 6D-spherical as well, while the hotstab can be considered not 6D-spherical at all. Notwithstanding the hotstab achieves good results in pose estimation (5.4), due to its shape, a small AAE's prediction mistake produces a COU value degradation.

Similar considerations can be made by observing the VSD metric, which, as described in Section 2.2.3, considers the 2D corresponding masks of the object model' visible parts and the distance matrices. VSD is inherently invariant under pose ambiguity. As shown in Equation 2.2.3, if the pixel p is in the intersection of the two visibility masks and the distance d is less than a misalignment tolerance τ, then the error for that pixel is d/τ; otherwise, the matching cost takes the maximum value 1. The hostab model achieves the worst performance, also caused by small errors in AAE prediction. On the other hand, the jug, being asymmetrical, gets a better result than the other objects.

Qualitative results These observations can be further seen in Figure 5.7. YoloV4 predictions are shown on the left and pose estimates and corresponding object renderings are shown on the right. Remarkably, even though the light conditions and water clarity are very unfavorable, the results are very good. In Figure 5.7a all the predicted rotation seems correct. However, the cup has a wrong rotation, it is overturned. In this case, the ADD metric reaches a very low value, while the COU, one of the highest. In Figure 5.7b, despite the box's rotation being completely wrong, independently of its symmetry, the COU will have an high value. In Figure 5.7c instead, the hotstab has a good rotation and a slightly wrong translation. Due to its handle, which does not match with the corresponding one depicted in the image, the COU and VSD values will be low.

For the sake of completeness, also the worst results are shown in Figure 5.8. YoloV4 predictions are shown on the left and pose estimates and corresponding object renderings are shown on the right. The objects are placed at a distance close to the 3 meters limit to test the method's robustness. YoloV4 is mainly responsible for the poor results. When YoloV4 misses or fails to detect an object correctly, the Augmented Autoencoder cannot remedy it in any way. In Figure 5.8a, the YoloV4 detects a box instead of hotstab and the AAE, which needs the object class and the cropped image, only tries to estimate the box rotation. Therefore, the major responsibility lies with the YoloV4 detections, but also the AAE fails to predict some poses, such as the jug pose in Figure 5.8c. These scenarios are very disadvantageous, objects are placed far away, cluttered, and environmental conditions unfavorable.

5.4.4 On edge

The project constraints, as described before, dictated the choices of the real-time, lightweight deep learning solutions and the input camera source, declining it by the selected resolution, FPS frequency and so on. The YoloV4 is the ideal choice for the 2D object detection sub-task since it can achieve the desired FPS while still adhering to the power constraints. The Augmented Autoencoder, on the other hand, is the ideal choice for the 6D pose estimation task. It is the best trade-off between accuracy and latency, as written in Section 5.3. Given that only four objects are considered in our scenario, AAE has been chosen over other solutions, despite not being able to scale with the number of objects.

Since the robotic arm mounted on the AUV requires a compact embedded board due to the limited physical space, Nvidia Xavier AGX was chosen for its better results than other Nvidia embedded boards or Xilinx boards, as shown in Chapter 3. The analysis indicates that Xavier AGX is the clear winner in almost all aspects, achieving the best power efficiency, highest mAP, lowest end-to-end latency, and highest total throughput.

(A)

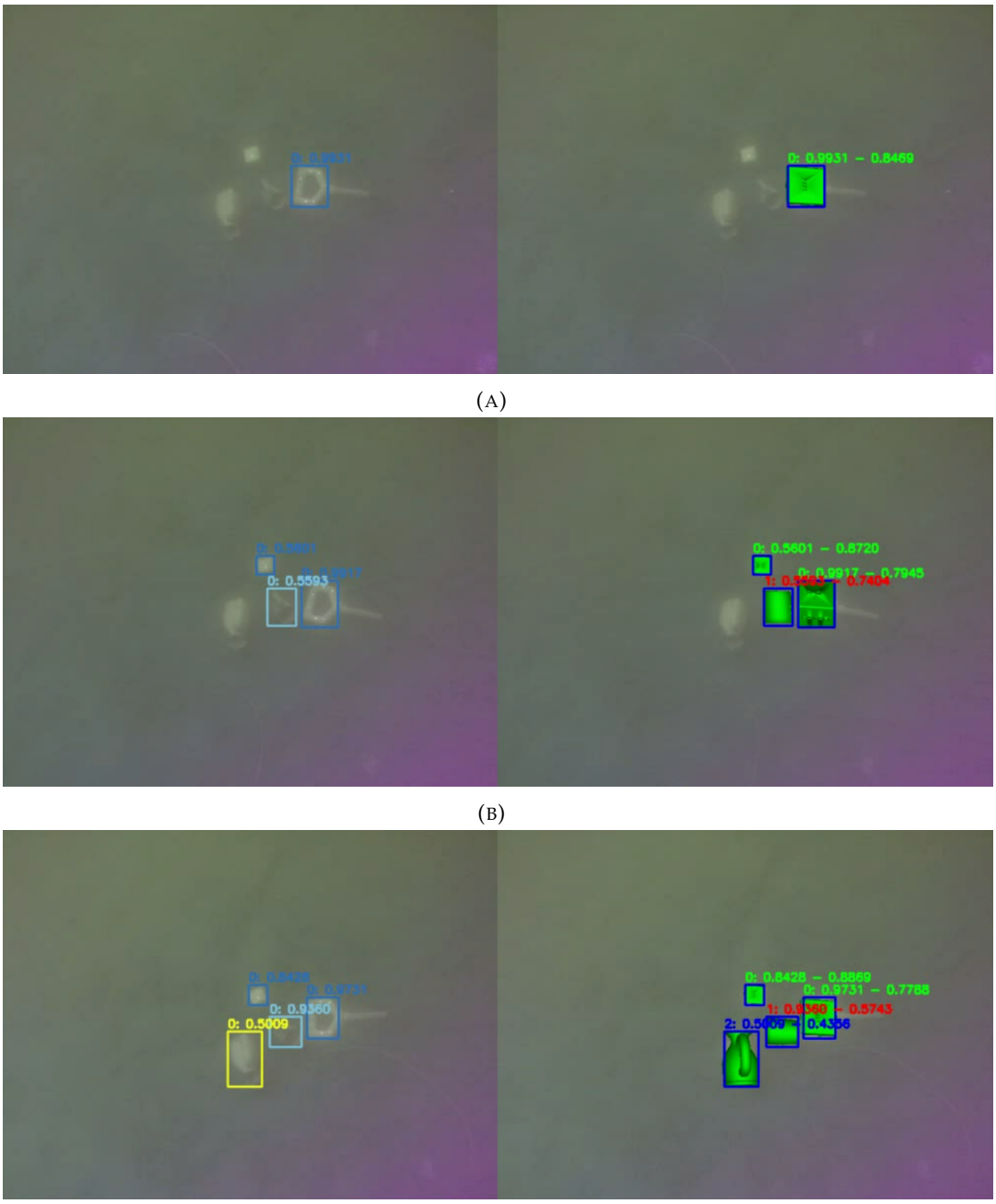
(B)

(C)

FIGURE 5.7: Selection of frame in real-world video recordings. These data are collected in Nami lake, Rome. YoloV4 detections are shown on the left, and the Augmented Autoencoder results are shown on the right. The green objects are the CAD models that have been rendered on the images with the predicted rotation and translation.

Therefore, an Nvidia Xavier AGX is considered in this section to report the end-to-end times of DNN solutions.

Setup About the platform, the Xavier AGX is configured with an operative system Ubuntu 20.04.5 LTS, with Nvidia Jetpack 5.0.2. To maximize NVIDIA Xavier AGX's performance, the power mode has been set to *MAXN* and *jetson_clocks* has



(A)

(B)

(C)

FIGURE 5.8: Selection of frame showing the worst results in real-world video recordings. These data are collected in Nami lake, Rome. YoloV4 detections are shown on the left, and the Augmented Autoencoder results are shown on the right. The green objects are the CAD models that have been rendered on the images with the predicted rotation and translation.

been launched before the tests. The Jetson Clock is a feature of the NVIDIA embedded platform that enables the user to adjust the GPU clock frequency and memory clock frequency. This allows users to optimize their system for maximum performance. Additionally, the NVIDIA embedded platform also offers a maximum N power mode which is designed to provide the highest performance available.

The NVIDIA Xavier AGX embedded platform supports three types of data precision: FP32, FP16, and INT8. Data precision refers to the number of bits used to represent data. FP32, or single-precision floating point, is the most precise type available and is used for applications that require higher precision and accuracy. FP16, or half-precision floating point, is a less precise type and is used for applications that require less precision or accuracy. INT8, or 8-bit integer, is the least precise type and is used for applications that require the least precision or accuracy.

In this preliminary test, the single-precision floating point FP32 is considered.

As for the networks, YoloV4 and Augmented Autoencoder, they have been developed with python3.7. Yolov4 running with the OpenCV API, while AAE with Tensorflow 2.6. As shown in Section 3.1.1, some frameworks, such as TensorRT, are written in CUDA to optimize the inference on GPU of the deep learning models. These optimizations are not currently being considered, but will be studied at a later stage.

In the following the end-to-end latency, FPS, and the details for the three phases of pre-processing, inference and post-processing are reported for YoloV4 and Augmented autoencoder networks.

		pre	inf	post	tot
single object	min	3,09	68,00	209,25	280,35
	max	15,87	100,80	224,22	340,89
	avg	4,93	72,23	215,02	292,19
multi objects	min	3,25	68,10	212,46	283,80
	max	16,27	97,72	247,72	361,72
	avg	6,33	80,38	219,18	305,90

TABLE 5.6: Inference times (in milliseconds) for the YoloV4 phases: pre-processing, inference and post-processing. These results are taken from 500 images, so this table shows the statistics of 500 inference times. The first time has been discarded not to consider the weight load and memory levels initialization. The rows show both YoloV4 running on a single class and Yolov4 running with multi objects. In particular, for the first experiment, only an hotstab is depicted in every image. In contrast, in the second experiment, the box, cup, jug and hotstab are always present in each scene.

In Table 5.6 the YoloV4's times are shown for two different experiments:

- The first experiment reports statistics over 499 images, depicting only one object, the hotstab. In each image there is only one instance of the hotstab and no other objects are present in the scene.
- The second experiment reports statistics over 499 images, depicting four objects per scene. Each scene shows one instance of each object type, the box, cup, jug and hotstab respectively.

In both experiments, 500 images are considered, but due to the weight load and memory levels initialization, which could alter the statistics, the first times have been discarded. Indeed, limited to the inference phase, considering the first time acquisition, the values for the two experiments are $3,727.60$ and $3,851.41$ respectively.

From the table 5.6, looking at the last column showing the total times, one can see a negligible increase in time between the two experiments, with one instance and with four objects.

Independently of the experiment, post-processing is the heavier phase in terms of latency. The discrepancy between these results and the ones shown in Figure 3.3 and Figure 3.4 is due to the different frameworks used, and different optimizations of some operations. In the previous work described in Chapter 3, a huge effort had been made to speed up as much as possible both the inference phase and the pre and post-processing phases, which are routinely run on CPUs. In these preliminary tests, the difference is just that.

			pre	inf	post	tot
single object	Hotstab	min	61,40	61,37	0,02	122,79
		max	79,21	77,59	0,08	156,89
		avg	67,32	66,99	0,03	134,33
multi objects	Box	min	64,40	66,40	0,02	130,81
		max	107,24	123,31	0,16	230,71
		avg	71,86	73,01	0,02	144,89
	Cup	min	64,07	65,30	0,02	129,39
		max	134,07	107,46	0,07	241,60
		avg	72,17	72,61	0,02	144,80
	Jug	min	65,27	65,92	0,01	131,21
		max	96,30	109,42	0,20	205,92
		avg	71,85	72,97	0,02	144,84
	Hotstab	min	64,24	65,61	0,01	129,86
		max	99,72	96,66	0,08	196,45
		avg	71,74	73,17	0,02	144,94
	Total	min	272,11	274,63	0,06	546,80
		max	388,75	393,71	0,28	782,74
		avg	287,76	291,90	0,10	579,76

TABLE 5.7: Inference times (in milliseconds) for the Augmented Autoencoder phases: pre-processing, inference and post-processing. These results are taken from 500 images, so this table shows the statistics of 500 inference times. The first time has been discarded not to consider the weight load and initialization memory levels. On the rows are shown both single object scenario and multi objects scenario. In particular, in the first experiment, only an AAE model runs over an image, while in the second experiment, four AAE models run on the same image, one for each object: box, cup, jug and hotstab. In the second group of rows also the total time, referred to as the time spent on an image is shown.

Regarding the 6D pose estimation part, the Augmented Autoencoder, as described before, works with single class. In the second experiment, the one with 4 objects per scene, four different AAE networks are required, one for each object. In Table 5.7 indeed, for the second experiment also the statistics of each AAE model are reported. In contrast to the Yolov4, the time for each phase of the scenario with four objects grows linearly with respect to the times with one object.

For the AAE models, the pre-processing and inference phases are equally heavy in terms of latency. As described earlier, optimization techniques, such as CPU parallelization and/or transferring computation to GPUs, are also not adopted here.

In Table 5.8, the end-to-end latency and FPS for YoloV4, Augmented Autoencoder and the sum of the two NNs are summarized. With a single object per scene, exactly two frames per second can be processed in the worst case. On the other

		YoloV4		AAE		TOT	
		tot	FPS	tot	FPS	tot	FPS
single object	min	280,35	3,57	122,79	8,14	403,13	2,48
	max	340,89	2,93	156,89	6,37	497,78	2,01
	avg	292,19	3,42	134,33	7,44	426,51	2,34
multi objects	min	283,80	3,52	546,80	1,83	830,61	1,20
	max	361,72	2,76	782,74	1,28	1.144,46	0,87
	avg	305,90	3,27	579,76	1,72	885,66	1,13

TABLE 5.8: This table shows the total inference times, as the sum of pre-processing, inference and post-processing times, for both single object and multi objects scenarios with the corresponding FPS value. The first two columns with tot labels are equal of the corresponding columns shown in Table 5.6 and 5.7. On the right, the sum of these two columns is computed, and then the total FPSs are shown.

hand, the corresponding worst case in a multi-object scenario does not allow even one frame per second.

Chapter 6

Conclusions and Open Problems

6.1 Conclusions

Deep learning methods have made significant advancements in recent years, allowing for more accurate and efficient solutions to a variety of complex problems. However, in the application field, it is important to consider more than just the efficiency and accuracy of deep methods. Unexpected bias may be introduced depending on the use case, resulting in unexpected outcomes.

The focus of this thesis, due to the vast variety of sub-fields in AI, is limited to the treatment of the 2D object detection and then the 6D pose estimation with particular interest on real-time scenarios.

This study wants to give some insights of the two problems under analysis in order to make a further step into the street of progress.

Chapter 2 This Chapter gives a well-organized survey of object detection and 6D pose estimation problems. A definition of a mathematical problem is provided for both topics, describing the goals and challenges of the two topics. A description of the history and the details of the datasets in the literature is provided. In addition, for each problem discussed, a formal definition of known metrics is presented, describing the cases in which they can be applied. Furthermore, a detailed review of the most common methods has been made available. For the object detection task, the described methodologies refer to the real-time subfield: real-time object detector. On the other hand, for the 6D pose estimation task, the presented methodologies refer only to RGB-based methods. A brief presentation of the GPGPU and FPGA architecture is described at the end of the chapter.

Chapter 3 This Chapter proposed a fair comparison of the heterogeneous embedded platforms and object detection convolutional neural networks, as a result of a collaboration with Tetrapak s.p.a.. The results were twofold: for the company, it was a matter of finding the best combination of platform and networks for its needs for anomaly detection in the production chain; from an academic perspective, it was a matter of finding the best trade-off for selecting the best object detector versus the best embedded platform from the different and metric-driven point of views. In particular, in addition to analyzing Tetrapak's industrial PC, already present in their production chain, also a lot of boards for both the GPGPU and FPGA platform families are considered into the analysis. For the GPGPU Nvidia Jetson family, the Xavier AGX, TX2 and Nano have been considered. Instead, for the FPGA Xilinx family, the XCZU7EV and XCZU9EG have been analyzed. Regarding the Neural Networks, a set of seven different real-time object detectors have been analyzed: YoloV3 and

its tiny version, Mobilenetv2-SSDLite, two versions of CenterNet differing in backbone architecture, ResNet101 and DLA34 respectively, and finally YoloV4 and its tiny version. To completely fair compare boards and methods, different floating point data representations can also be considered. For example, the Xilinx boards support only INT8 representation. On the other hand, the industrial PC supports only the FP32 precision. However, Nvidia TX2 and Nano support both FP32 and FP16 representation, while Xavier AGX support also the INT8 in addition to the other NVIDIA boards. The discrepancy of the data representation between boards affects obviously the memory usage to represent the methods and also its processing speed. The processing speed affects the power consumption of a given task. In addition, the method's accuracy and end-to-end latency are affected by different representation precision. End-to-end latency is the set of pre-processing, inference and post-processing times. Obviously, the hardware platform itself affects all the metrics described above.

The conducted study has demonstrated Nvidia Xavier AGX as the unquestionable winner of the compared boards. Xavier AGX is the best embedded platform in terms of end-to-end latency, and throughput. For the networks, YoloV4 is the best network among those under analysis.

Chapter 4 This Chapter is focused on one of the less explored themes in the 6D pose estimation domain: explainability. The best performing 6D method, EfficientPose(EP) [6] [1], and the most widely used state-of-the-art dataset, LineMod (LM) [30], are analysed. Thanks to some techniques in state-of-the-art for understanding the decision-making processes, for example saliency maps such us Vanilla-Gradient [84] and Grad-CAM [82], our experiments uncover a surprising bias introduced in LineMod.

EP is particularly suitable in our study for its operation. This is a full RGB-based end-to-end 6D method and differs from most others in that it does not use the cropped image. In fact, the bias discovered relates to the data acquisition methodology itself; where the objects are placed on a desk, and insight a custom chessboard delimitated from ArUco tags. The presence of those markers is a common technique in 6D data acquisition in order to derive the exact ground truth objects' pose. However, as shown in the Chapter, the presence of ArUco markers in the scenes distorts the method's predicting capabilities.

Furthermore, in LineMod also another bias is discovered. Acquiring real-world data in static or semi-static scenes entails a simpler scenario for the candidate method. Such method, in order to identify the pose of a target object, could also see the surrounding objects that rotate and translate in the same manner as the target one.

These are the reason why the EfficientPose, which works with the entire frame, is particularly prone to these biases.

The severity of these biases, especially in a dataset like LineMod, which has become the de facto standard in the 6D pose literature datasets, resides in the fact that it distorts the rankings of the best performing 6D methods, such as EfficientPose, which is evidence of this. On these datasets, the methods' expressive capacity can be falsely evaluated.

In this Chapter, we also introduced a new dataset without the ArUco markers, showing better results. In the future, our idea is to improve our data generalization technique in order to present a new bias-less dataset similar to LineMod.

[1] https://paperswithcode.com/sota/6d-pose-estimation-on-linemod

Chapter 5 Finally, in this Chapter a new use case is presented: the underwater one. The new, almost completely unknown, underwater domain has been investigated thanks to a collaboration with the Technology Innovation Institute (TII). The goal of the project was to develop a 6D pose estimation pipeline to pick objects on the seabed with robotic harm mounted on an underwater drone. This use case was a sub-task of a bigger use case that involves several other artificial intelligence components, such as autonomous driving, obstacle avoidance, trajectory computation, and so on. We compared several state-of-the-art methods and finally proposed a robust pipeline, consisting into split the problem into two different tasks: object detection via YoloV4 and pose estimation via Augmented Autoencoder. This choice is widely motivated, because of the difficult object domain, mostly symmetrical and not easily distinguishable in pose, the novel challenging problems related to the heterogeneous environment, light conditions and low visibility states.

The proposed methodologies are largely evaluated with the common 6D metrics, considering symmetries and asymmetries developed ones.

The remarkable contribution is also in the dataset's field. We collected a new real-world underwater dataset for the four considered objects and also we developed a new tool to collect synthetic, but near realistic, data via Unity.

Finally, the end-to-end latency and FPS, with details for each sub-phase of pre-processing, inference and post-processing, are detailed. The tests are computed on an Nvidia Xavier AGX board to give a first contribution to the real-time 6D pose research area.

Finally, I put into practice the knowledge acquired in the previous chapters by implementing a robotic manipulator project in underwater scenario that utilizes the concepts of 6D pose estimation and real-time computing. The results achieved have been satisfactory, and this study stands to be a first step for community growth.

6.2 Open Problems

While working in depth on the various issues just described, some not intuitive challenging problems are discovered, regarding both methodological approaches and missing questions.

Fairness comparison For what concerns the Real-Time Computing, the discrepancy of platforms used to evaluate or develop deep learning methods is one of the most common issues in the field of artificial intelligence. Different platforms can have different hardware and software configurations, which can lead to discrepancies in results when comparing the performance of AI methods on those platforms. This can make it difficult to draw fair comparisons between different AI methods on different platforms.

Regarding the hardware sphere, different architecture types, such as x86, aarch64, power9 and so on, could compute the same simple processor operations in different ways, changing a bit the final results, especially in deep methods where there are a lot o operations and a little error can be propagated a lot of times. As mentioned before, also different data representations could obviously alter the final accuracy results.

At the same time, also the software stack affects the performance computation. In this case, different operating systems, different DNN framework versions, or simply different library versions could alter the results.

The use of different hardware and software configurations across various platforms for evaluating AI methods can create discrepancies in the results. For instance, when using deep learning methods, the development programming language is typically Python and the solutions are usually run in one environment. However, the versions of Anaconda and other packages available on different platforms may vary, making it difficult to maintain the same configuration across platforms, resulting in an unfair comparison of results

Therefore, a new deep learning method developed today, with updated versions of software and new architectures, could find advantages over older methods, being better in both accuracy and latencies.

Explainability Another important issue, especially when working with companies, is that of explainability. For a long time, deep learning methods were viewed as black boxes. With the motivation of being data-driven methods, we were limited to seeing the performance achieved. Today, partly because of the many biases introduced, knowing how these kinds of methods work is increasingly important. It was primarily through the knowledge provided by saliency maps that we were able to identify the bias described in Chapter 4.

The importance of this lies not only in checking that no bias is introduced into the learning process, but especially in understanding and improving the predictive ability of the methods. Knowing how the learning process can be affected would also allow specific and precise improvements to be made to a part of it, reducing the computational costs and workload of designers of a new network. Since training a network is also a very costly operation in terms of resources and time, one could identify that minimal subset of input data that would achieve the same or improved performance.

Category-level 6d pose estimation Although the research field of 6D pose estimation is quite recent, there are both RGB-based and RGBD-based subfields that have achieved good results for some particular working scenarios. Some methods are capable of learning multiple objects at once, such as EfficientPose [6], while others need a trained model for each individual object, such as Augmented Autoencoder [88]. However, their greatest limitation is that they must have the CAD of the object in both training and testing, which makes them unable to recognize the pose of an unknown object or, more simply, a meta-cup, a cup other than the one they are able to recognize and for which they have the CAD model. These methods belong to the family defined as instance-level 6D pose estimation. Instance-level 6D pose estimation approaches have numerous limitations. As mentioned above, they cannot recognize objects other than known objects; therefore, they do not work with unseen objects and meta-objects. These methods are very dependent on the dataset. They can estimate only the objects present in the dataset. The effort to learn a new object is enormous: both the CAD model and the dataset are needed.

In the literature, some novel methods are proposed in order to generalize the object concept. For example, MegaPose [46] tries to estimate the 6D pose of a novel object, although starting from classical CAD-based training.

Furthermore, recently a paper [104] proposed in 2019 the first method of category-level 6d pose estimation, giving birth to this new challenging subfield. The goal is to create a technique for general settings and objects that have never been seen before and do not have CAD models. They overcome this problem by defining a shared canonical representation for all possible object instances within a category, called

Normalized Object Coordinate Space. In addition, they introduce the first category-level 6d pose estimation dataset-generation approach, called Context-Aware MixEd ReAlity (CAMERA). Few other works have tried to solve the same challenge, following the same idea of NOCS, or completely changing approach [10] [114] [14] [53] [9]. However, they mostly focused on RGB-D inputs, since RGB features are sensitive to color variations. Therefore, this subfield remains an open challenge, especially for RGB image inputs, but it is advancing rapidly and has the potential to enable a wide range of new applications in areas such as augmented reality, robotics, and 3D scene understanding. It should be included in a wider and broader purpose, that an algorithm in the future will be able to generalize the concept of 6D pose estimation for every object.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Paul J Besl and Neil D McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. Spie. 1992, pp. 586–606.

[3] Gideon Billings and Matthew Johnson-Roberson. "SilhoNet-fisheye: Adaptation of a ROI based object pose estimation network to monocular fisheye images". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4241–4248.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[5] Eric Brachmann et al. "Learning 6d object pose estimation using 3d object coordinates". In: *European conference on computer vision*. Springer. 2014, pp. 536–551.

[6] Yannick Bukschat and Marcus Vetter. "EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach". In: *arXiv preprint arXiv:2011.04307* (2020).

[7] Zhe Cao, Yaser Sheikh, and Natasha Kholgade Banerjee. "Real-time scalable 6DOF pose estimation for textureless objects". In: *2016 IEEE International conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 2441–2448.

[8] Jiasi Chen and Xukan Ran. "Deep learning with edge computing: A review". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[9] Wei Chen et al. "Fs-net: Fast shape-based network for category-level 6d object pose estimation with decoupled rotation mechanism". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 1581–1590.

[10] Xu Chen et al. "Category level object pose estimation via neural analysis-by-synthesis". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI 16*. Springer. 2020, pp. 139–156.

[11] Dmitry Chetverikov et al. "The trimmed iterative closest point algorithm". In: *2002 International Conference on Pattern Recognition*. Vol. 3. IEEE. 2002, pp. 545–548.

[12] Edwin KP Chong and Stanislaw H Zak. *An introduction to optimization*. Vol. 75. John Wiley & Sons, 2013.

[13] Jifeng Dai et al. "Deformable convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 764–773.

[14] Yan Di et al. “Gpv-pose: Category-level object pose estimation via geometry-guided point-wise voting”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 6781–6791.

[15] Xiaohan Ding et al. “Repvgg: Making vgg-style convnets great again”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 13733–13742.

[16] Bertram Drost et al. “Introducing mvtec itodd-a dataset for 3d object recognition in industry”. In: *Proceedings of the IEEE international conference on computer vision workshops*. 2017, pp. 2200–2208.

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural architecture search: A survey”. In: *arXiv preprint arXiv:1808.05377* (2018).

[18] Scott Ettinger et al. “Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9710–9719.

[19] Mark Everingham et al. “The pascal visual object classes challenge: A retrospective”. In: *International journal of computer vision* 111.1 (2015), pp. 98–136.

[20] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88 (2010), pp. 303–338.

[21] Sergio Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292.

[22] Zheng Ge et al. “Yolox: Exceeding yolo series in 2021”. In: *arXiv preprint arXiv:2107.08430* (2021).

[23] Daniel Glasner et al. “aware object detection and pose estimation”. In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 1275–1282.

[24] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[25] Kaiming He et al. “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[26] Kaiming He et al. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), pp. 1904–1916.

[27] Xin He, Kaiyong Zhao, and Xiaowen Chu. *AutoML: A Survey of the State-of-the-Art*. 2019. arXiv: 1908.00709 [cs.LG].

[28] Stefan Hinterstoisser et al. “Gradient response maps for real-time detection of textureless objects”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.5 (2011), pp. 876–888.

[29] Stefan Hinterstoisser et al. “Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes”. In: *Asian conference on computer vision*. Springer. 2012, pp. 548–562.

[30] Stefan Hinterstoisser et al. “Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes”. In: *2011 international conference on computer vision*. IEEE. 2011, pp. 858–865.

[31] Tomáš Hodaň, Jiří Matas, and Štěpán Obdržálek. "On evaluation of 6D object pose estimation". In: *European Conference on Computer Vision*. Springer. 2016, pp. 606–619.

[32] Tomáš Hodan et al. "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects". In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 880–888.

[33] Jonas Dominik Homburg et al. "Constraint Exploration of Convolutional Network Architectures with Neuroevolution". In: *Advances in Computational Intelligence*. Ed. by Ignacio Rojas, Gonzalo Joya, and Andreu Catala. Cham: Springer International Publishing, 2019, pp. 735–746. ISBN: 978-3-030-20518-8.

[34] Sabir Hossain and Deok-jin Lee. "Deep Learning-Based Real-Time Multiple-Object Detection and Tracking from Aerial Imagery via a Flying Robot with GPU-Based Embedded Devices". In: *Sensors* 19.15 (2019), p. 3371.

[35] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv:1704.04861* (2017).

[36] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[37] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. "Comparing images using the Hausdorff distance". In: *IEEE Transactions on pattern analysis and machine intelligence* 15.9 (1993), pp. 850–863.

[38] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[39] MyungHwan Jeon et al. "Underwater object detection and pose estimation using deep learning". In: *IFAC-PapersOnLine* 52.21 (2019), pp. 78–81.

[40] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[41] Licheng Jiao et al. "A Survey of Deep Learning-Based Object Detection". In: *IEEE Access* 7 (2019), pp. 128837–128868.

[42] Roman Kaskman et al. "Homebreweddb: Rgb-d dataset for 6d pose estimation of 3d objects". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019, pp. 0–0.

[43] Alex Kendall, Matthew Grimes, and Roberto Cipolla. "Posenet: A convolutional network for real-time 6-dof camera relocalization". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2938–2946.

[44] Chloe Eunhyang Kim et al. "A comparison of embedded deep learning methods for person detection". In: *arXiv preprint arXiv:1812.03451* (2018).

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[46] Yann Labbé et al. "MegaPose: 6D Pose Estimation of Novel Objects via Render & Compare". In: *arXiv preprint arXiv:2212.06870* (2022).

[47] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[48] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem". In: *International journal of computer vision* 81.2 (2009), pp. 155–166.

[49] Chuyi Li et al. "YOLOv6: A single-stage object detection framework for industrial applications". In: *arXiv preprint arXiv:2209.02976* (2022).

[50] Shiqi Li, Chi Xu, and Ming Xie. "A robust O (n) solution to the perspective-n-point problem". In: *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2012), pp. 1444–1450.

[51] Zhigang Li, Gu Wang, and Xiangyang Ji. "Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 7678–7687.

[52] Joerg Liebelt, Cordelia Schmid, and Klaus Schertler. "independent object class detection using 3d feature maps". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8.

[53] Haitao Lin et al. "SAR-Net: shape alignment and recovery network for category-level 6D object pose and size estimation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 6707–6717.

[54] Tsung-Yi Lin et al. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[55] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[56] Li Liu et al. "Deep learning for generic object detection: A survey". In: *International Journal of Computer Vision* 128.2 (2020), pp. 261–318.

[57] Shu Liu et al. "Path aggregation network for instance segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8759–8768.

[58] Wei Liu et al. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[59] Xiang Long et al. "PP-YOLO: An effective and efficient implementation of object detector". In: *arXiv preprint arXiv:2007.12099* (2020).

[60] Paolo Meloni et al. "Neuraghe: Exploiting cpu-fpga synergies for efficient and flexible cnn inference acceleration on zynq socs". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11.3 (2018), pp. 1–24.

[61] Frank Michel et al. "Global hypothesis generation for 6D object pose estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 462–471.

[62] Diganta Misra. "Mish: A self regularized non-monotonic neural activation function". In: *arXiv preprint arXiv:1908.08681* (2019).

[63] Christoph Molnar. "A guide for making black box models explainable". In: *URL: https://christophm. github. io/interpretable-ml-book* (2018).

[64] Kamyar Nazeri et al. "Edgeconnect: Generative image inpainting with adversarial edge learning". In: *arXiv preprint arXiv:1901.00212* (2019).

[65] Alejandro Newell, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation". In: *European conference on computer vision*. Springer. 2016, pp. 483–499.

[66] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 3400–3407.

[67] Utku Ozbulak. *PyTorch CNN Visualizations*. https://github.com/utkuozbulak/pytorch-cnn-visualizations. 2019.

[68] Èric Pairet et al. "Nukhada USV: a Robot for Autonomous Surveying and Support to Underwater Operations". In: *OCEANS 2022 - Chennai*. 2022, pp. 1–6. DOI: 10.1109/OCEANSChennai45887.2022.9775538.

[69] Shiye Pan and Xinmei Wang. "A Survey on Perspective-n-Point Problem". In: *2021 40th Chinese Control Conference (CCC)*. IEEE. 2021, pp. 2396–2401.

[70] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[71] Georgios Pavlakos et al. "6-dof object pose from semantic keypoints". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 2011–2018.

[72] Sida Peng et al. "Pvnet: Pixel-wise voting network for 6dof pose estimation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4561–4570.

[73] Mahdi Rad and Vincent Lepetit. "Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3828–3836.

[74] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[75] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[76] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[77] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[78] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[79] Davide Sapienza et al. "Deep Image Prior for medical image denoising, a study about parameter initialization". In: *Frontiers in Applied Mathematics and Statistics* 8 (2022).

[80] Davide Sapienza et al. "Deep learning-assisted analysis of automobiles handling performances". In: *Communications in Applied and Industrial Mathematics* 13.1 (2022), pp. 78–95.

[81] Carmelo Scribano et al. "All You Can Embed: Natural Language based Vehicle Retrieval with Spatio-Temporal Transformers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 4253–4262.

[82] Ramprasaath R Selvaraju et al. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[83] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *International conference on machine learning*. PMLR. 2017, pp. 3145–3153.

[84] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *arXiv preprint arXiv:1312.6034* (2013).

[85] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[86] Min Sun et al. "Depth-encoded hough voting for joint object detection and shape recovery". In: *European Conference on Computer Vision*. Springer. 2010, pp. 658–671.

[87] Pei Sun et al. "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[88] Martin Sundermeyer et al. "Implicit 3d orientation learning for 6d object detection from rgb images". In: *Proceedings of the european conference on computer vision (ECCV)*. 2018, pp. 699–715.

[89] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[90] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[91] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[92] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[93] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[94] Mingxing Tan, Ruoming Pang, and Quoc V Le. "Efficientdet: Scalable and efficient object detection". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10781–10790.

[95] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. "Real-time seamless single shot 6d object pose prediction". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 292–301.

[96] Zhi Tian et al. "Fully convolutional one-stage 3D object detection on LiDAR range images". In: *arXiv preprint arXiv:2205.13764* (2022).

[97] Stephen Tyree et al. "6-DoF pose estimation of household objects for robotic manipulation: An accessible dataset and benchmark". In: *arXiv preprint arXiv:2203.05701* (2022).

[98] Yaman Umuroglu et al. "Finn: A framework for fast, scalable binarized neural network inference". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2017, pp. 65–74.

[99] Micaela Verucchi et al. "A systematic assessment of embedded neural networks for object detection". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE. 2020, pp. 937–944.

[100] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." In: *Journal of machine learning research* 11.12 (2010).

[101] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *arXiv preprint arXiv:2207.02696* (2022).

[102] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. *Designing Network Design Strategies Through Gradient Path Analysis*. 2022. DOI: 10.48550/ARXIV.2211.04800. URL: https://arxiv.org/abs/2211.04800.

[103] Chien-Yao Wang et al. "CSPNet: A new backbone that can enhance learning capability of CNN". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 390–391.

[104] He Wang et al. "Normalized object coordinate space for category-level 6d object pose and size estimation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2642–2651.

[105] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. "Recent advances in deep learning for object detection". In: *Neurocomputing* (2020).

[106] Yu Xiang et al. "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes". In: *arXiv preprint arXiv:1711.00199* (2017).

[107] Saining Xie et al. "Aggregated residual transformations for deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[108] Saining Xie et al. "Exploring randomly wired neural networks for image recognition". In: *arXiv preprint arXiv:1904.01569* (2019).

[109] Xiaowei Xu et al. "Dac-sdc low power object detection challenge for uav applications". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[110] Fisher Yu et al. "Bdd100k: A diverse driving dataset for heterogeneous multitask learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2636–2645.

[111] Fisher Yu et al. "Deep layer aggregation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2403–2412.

[112] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. "Objects as points". In: *arXiv preprint arXiv:1904.07850* (2019).

[113] Xizhou Zhu et al. "Deformable convnets v2: More deformable, better results". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9308–9316.

[114] Lu Zou et al. "6d-vit: Category-level 6d object pose estimation via transformer-based instance representation learning". In: *IEEE Transactions on Image Processing* 31 (2022), pp. 6907–6921.

[115] Zhengxia Zou et al. “Object detection in 20 years: A survey”. In: *arXiv preprint arXiv:1905.05055* (2019).