Méthodes d'apprentissage automatique pour des applications robotiques dans un contexte industriel : étude de cas du tri robotisé

Joris Guerin

▶ **To cite this version:**

Joris Guerin. Méthodes d'apprentissage automatique pour des applications robotiques dans un contexte industriel : étude de cas du tri robotisé. Signal and Image processing. Ecole nationale supérieure d'arts et métiers - ENSAM, 2018. English. NNT : 2018ENAM0045 . tel-04351493

HAL Id: tel-04351493

https://pastel.hal.science/tel-04351493v1

Submitted on 18 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

2018-ENAM-0045

École doctorale n° 432 : Science des Métiers de l'ingénieur

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

l'École Nationale Supérieure d'Arts et Métiers

Spécialité " Génie Informatique "

présentée et soutenue publiquement par

Joris GUÉRIN

le 10 Décembre 2018

Méthodes d'apprentissage automatique pour des applications robotiques dans un contexte industriel : étude de cas du tri robotisé

-

Machine learning improvements for robotic applications in industrial context: Case study of autonomous sorting

Directeur de thèse : **Olivier GIBARU**
Co-encadrement de la thèse : **Stéphane THIERY, Éric NYIRI**

Jury

M. Ivan LAPTEV, Directeur de recherche, WILLOW, INRIA Paris	Président
M. Olivier PIETQUIN, Professeur des universités, Google Brain Paris	Rapporteur
M. Jean-Pierre GAZEAU, Ingénieur de recherche, Pprime, Université de Poitiers	Rapporteur
M. Lorenzo NATALE, Maître de conférences, iCub, Instituto Italiano di Tecnologia	Examinateur
M. Byron BOOTS, Maître de conférences, IRIM, Georgia Institute of Technology	Examinateur
M. Olivier GIBARU, Professeur des universités, LISPEN, Arts et Métiers ParisTech - Lille	Examinateur
M. Stéphane THIERY, Maître de conférences, LISPEN, Arts et Métiers ParisTech - Lille	Invité
M. Éric NYIRI, Maître de conférences, LISPEN, Arts et Métiers ParisTech - Lille	Invité

T H È S E

Arts et Métiers ParisTech - Campus de Lille
Laboratoire d'Ingénierie des Systèmes Physiques et Numériques

À Baba

Remerciements

Cette thèse a été réalisée au centre des Arts et Métiers ParisTech de Lille, au sein du Laboratoire d'Ingénierie des Systèmes Physiques et Numériques (LISPEN). J'ai également passé sept mois dans le Robot Learning Lab (RLL) affilié au Center for Machine Learning (CML) et à l'Institute for Robotics and Intelligent Machines (IRIM), à Georgia Tech, Atlanta, USA. Ce projet a reçu des soutiens financiers du projet ColRobot (collaborative robotics for assembly and kitting in smart manufacturing, European Union's 2020 research and innovation program under grant agreement No.688807), de Fulbright Haut-de-France, ainsi que de la Fondation Arts et Métiers (convention No. 8130-01).

En premier lieu, je tiens à remercier **Ivan Laptev**, directeur de recherche dans l'équipe WILLOW à l'INRIA de Paris, d'avoir présider le jury de ma soutenance de thèse. Je remercie également **Olivier Pietquin**, staff research scientist à Google Brain Paris et **Jean-Pierre Gazeau**, ingénieur de recherche à l'institut PPrime (UPR CNRS 3346) de l'université de Poitiers, pour avoir rapporté ma thèse. Les commentaires constructifs de leurs deux rapports très détaillés m'ont permis d'améliorer mon travail. Je tiens aussi à remercier **Lorenzo Natale**, ingénieur de recherche dans le département iCub à l'Istituto Italiano di Tecnologia (IIT), d'avoir pris part à mon jury de thèse.

Je remercie **Olivier Gibaru**, professeur aux Arts et Métiers de Lille, qui a dirigé ma thèse et m'a permis de conduire mes recherches dans les meilleures conditions. De même, je remercie **Éric Nyiri**, maître de conférences aux Arts et Métiers de Lille, qui m'a apporté un soutien technique et scientifique essentiel tout au long de ma thèse. J'adresse également des remerciements très spéciaux à mon encadrant et ami **Stéphane Thiery**, maître de conférences aux Arts et Métiers de Lille, qui s'est fortement impliqué, tant humainement et scientifiquement, durant mon doctorat. Enfin, je

remercie **Byron Boots**, assistant professor à Georgia Tech, pour m'avoir fait beaucoup avancer grâce à ses précieux conseils.

J'adresse des remerciements à l'équipe d'ingénieurs du LISPEN, **Jorge Palos**, **Adrien Florit** et **Vincent Deschodt** qui m'ont épaulé dans chacune des expériences pratiques que j'ai dû mettre en place.

Je remercie l'ensemble de mes amis d'Atlanta, **Yash Chitalia**, **Marina Alaricheva**, **Shray Bansal** et **Sasha Lambert** qui ont fait de mes mois américains une période enrichissante, de laquelle je garde d'excellents souvenirs. J'adresse également mes plus tendres remerciements à **Regina Nobre**, qui m'apporte beaucoup au quotidien et qui m'a soutenu durant les moments plus difficiles de cette thèse.

De même, j'adresse toute ma gratitude à l'ensemble des doctorants des Arts et Métiers, ainsi que tous mes autres amis de Lille. Grâce à vous, j'ai passé trois années qui comptent parmi mes plus belles. En particulier, je remercie mes compères **Emmanuel Cottanceau**, **Gil Boye de Sousa** et **Pierre Besset**, ainsi que **Tiago Moraes**, **Franck Hernoux**, **David Busson**, **Arthur Givois**, **Guillaume Baille**, **Carine Charles**, **Juliette Morin**, **Thomas Pidancier** et **Nadim El Hayek**.

Je conclue ces remerciements par mon frère, **Mathias Guérin**, mes parents, **Valérie et Philippe Guérin**, ainsi que l'ensemble des membres de ma famille qui m'ont toujours soutenu et encouragé à suivre le chemin qui me correspond. Enfin, je profite de cette section de remerciement pour témoigner toute ma gratitude et mon amour à mon grand-père, **Daniel Guérin**, sans qui je ne serais pas la personne que je suis aujourd'hui.

Abstract

Thanks to their flexible mechanical design, modern industrial robots can be programmed for different tasks without physical modification. In addition, they are highly instrumented and should be able to be responsive to their environment. However, the use of robots in industry is still restricted to repeatable tasks with low level of adaptability. In an industrial context, it is essential to program robots that can autonomously adapt to different applications and are robust to changes in working conditions. The machine learning framework for robot programming is well suited to design such kinds of adaptive and robust applications. Hence, in this thesis, several machine learning contributions are presented, aiming at designing smarter robotic applications, with a broader operational range. The methods developed are centered on autonomous sorting, but may be useful to address problems in many other subfields of robotics. Throughout this thesis, we propose new approaches to image clustering, optimal view selection, trajectory learning and stereo localization, with the objective of designing more universal robotic sorting applications.

Keywords: Robotic sorting, Image clustering, Transfer learning, Optimal view - selection, Trajectory learning, Autonomous dataset generation.

Contents

List of Figures

List of Tables

Glossary

Agg
: Agglomerative clustering (En.)
Regroupement hiérarchique ascendant (Fr.)

BLC
: Background/Lighting Conditions (En.)
Conditions d'arrière plan et d'éclairage (Fr.)

BN
: Batch Normalization (En.)
Normalisation par lot (Fr.)

BNet
: Best Network (En.)
Meilleur réseau (Fr.)

BoF
: Bag of Features (En.)
Sac de caractéristiques (Fr.)

CC
: Concatenate and Cluster (En.)
Concaténer et partitionner (Fr.)

CNN
: Convolutional Neural Network (En.)
Réseau neuronal convolutif (Fr.)

CP
: Clustering Problem (En.)
Problème de partitionnement (Fr.)

CVKM
: Coefficient of Variation weighted K-Means (En.)
K-moyennes pondéré par le coefficient de variation (Fr.)

DEC
: Deep Embedded Clustering (En.)
Partitionnement intégré profond (Fr.)

DH

Denavit-Hartenberg model (En.)
Modèle de Denavit-Hartenberg (Fr.)

DMVC

Deep Multi-View Clustering (En.)
Partitionnement multi-vues profond (Fr.)

EC

Ensemble Clustering (En.)
Partitionnement de données ensembliste (Fr.)

FC

Fully Connected (En.)
Couche entièrement connectée (Fr.)

FM(I)

Fowlkes-Mallows Index (En.)
Indice de Fowlkes-Mallows (Fr.)

GC

Geometrical Center (En.)
Centre géométrique (Fr.)

GPS

Guided Policy Search (En.)
Recherche de stratégie guidée (Fr.)

GRKM

Gap-Ratio weighted K-Means (En.)
K-moyennes pondéré par le rapport des écarts (Fr.)

IB

Information Bottleneck (En.)
Goulot d'étranglement de l'information (Fr.)

IC

Image Clustering (En.)
Partitionnement d'images (Fr.)

ICA

Independent Component Analysis (En.)
Analyse en composantes indépendantes (Fr.)

IDEC

Improved Deep Embedded Clustering (En.)
Partitionnement intégré profond amélioré (Fr.)

iLQR/iLQG

Iterative Linear Quadratic Regulator/Gaussian control (En.)
Contrôle lineaire-quadratique (Gaussien) itératif (Fr.)

JULE

Joint Unsupervised LEarning of representations and clusters (En.)
Apprentissage non-supervisé et simultané de caractéristiques et groupes (Fr.)

KL

Kullback-Leibler divergence (En.)
Divergence de Kullback-Leibler (Fr.)

KM

K-Means clustering (En.)
Partitionnement en k-moyennes (Fr.)

LNet

Leading Network (En.)
Réseau de tête (Fr.)

LQR/LQG

Linear Quadratic Regulator/Gaussian control (En.)
Contrôle lineaire-quadratique (Gaussien) (Fr.)

MC

Monte Carlo (En.)
Monte Carlo (Fr.)

MDP

Markov Decision Process (En.)
Processus de décision Markovien (Fr.)

ML

Machine Learning (En.)
Apprentissage automatique (Fr.)

MLP

Multi-Layer Perceptron (En.)
Perceptron multicouche (Fr.)

MV

Multi-View (En.)
Multi-vues (Fr.)

MVC

Multi-View Clustering (En.)
Partitionnement multi-vues (Fr.)

MVEC
: Multi-View Ensemble Clustering (En.)
Partitionnement ensembliste multi-vues (Fr.)

NMI
: Normalized Mutual Information (En.)
Information mutuelle normalisée (Fr.)

NN
: Neural Network (En.)
Réseau neuronal (Fr.)

OC
: Optical Center (En.)
Centre optique (Fr.)

PUR
: Clustering Purity (En.)
Pureté de partitionnement (Fr.)

ReLu
: Rectified Linear units (En.)
Unité de rectification linéaire (Fr.)

RGB
: Red-Green-Blue color code (En.)
Encodage couleur rouge-vert-bleu (Fr.)

RGBD
: Red-Green-Blue-Distance (En.)
Rouge-vert-bleu-distance (Fr.)

RL
: Reinforcement Learning (En.)
Apprentissage par renforcement (Fr.)

SVS
: Semantic View Selection (En.)
Sélection de vue sémantique (Fr.)

URS
: Unsupervised Robotic Sorting (En.)
Tri robotique non-supervisé (Fr.)

WNet
: Worst Network (En.)
Pire réseau (Fr.)

Part I

Introduction

Chapter 1

Towards more generic robotic applications

Abstract

This chapter introduces the context and organization of the thesis. Over the past two decades, robotics research has undergone important changes driven by new industrial perspectives. The objectives are now to provide robots with skills that work in a variety of situations and are robust to environment changes. Machine Learning is a particularly well adapted framework for such kind of applications. In this thesis, we propose to rethink the standard robotic sorting application within this context. Usually, to decide where to sort objects, a robotic sorting system needs to solve either an instance retrieval (known object) or a supervised classification (predefined set of classes) problem. In this chapter, we introduce a new decision making module, where the robotic system chooses how to sort the objects in an unsupervised way. This approach generalizes common robotic sorting to any new set of previously unseen objects. This application is called Unsupervised Robotic Sorting (URS) and entails most of our contributions. Furthermore, other robotic skills for autonomous trajectory learning and 3D stereo localization are developed with similar motivations.

1.1 Machine learning for adaptive robotic skills

1.1.1 History of industrial robotics

In 1962, General Motors installed serial robots to carry out spot welding and die-castings extraction in their New Jersey's plant. These machines, developed by Unimation, were the first robots to successfully perform industrial tasks and mark the beginning of a new era in industrial production. Industrial robots are multifunctional machines which can be programmed to carry out different tasks. They differ from more standard specialized machines in that they can be reprogrammed for a new task without physical modification, and indeed, during three decades, many Unimation robots were sold worldwide to execute a variety of tasks. This reprogrammability presents the huge advantage of reducing the costs and times of mechanical design for new industrial applications, which rapidly lead to the emergence of many new robot manufacturer. The main tasks robots were dealing with in the end of the 20^{th} century were *assembly* (([Chen and Burdick, 1995]), ([Grunes et al., 1995]), ([Bonert et al., 2000])), *welding* (([Sicard and Levine, 1988]), ([Kim et al., 1998])) and *sorting* (see Section 1.2.1.1). A more complete study about the history of industrial robotics can be found in both ([Westerlund, 2000]) and ([Wallén, 2008]).

This important development of robotics in industrial factories has brought many new interesting problems to the research community (([Sciavicco and Siciliano, 2000]), ([Groover et al., 1986]), ([Nof, 1999]), ([Mair, 1988])). During 40 years, research in robotics was mostly driven by two problems of utmost importance for industry:

- How to define an optimal trajectory to accomplish a given task? (mechanics)
- How to ensure that the given trajectories are correctly followed? (control)

All these years of research in automation lead to impressive production plants where robots perform complex tasks at high velocity and with high precision (Figure 1.1).

However, this way of approaching robotics also has its limits as the robots can only perform highly repeatable tasks. In addition, such high velocity and non-adaptive applications constrain the robots to be isolated in secured environments, preventing interaction with humans. Hence, implementing a new application requires to physically build the robotic cell as well as programming the robot, which is a complex task requiring highly qualified engineers. These limitations mitigate the advantage of industrial robots over special machines since a huge amount of work is still required for a new implementation, thus limiting the use of robots to tasks dealing with large production volumes in order to be profitable.

Figure 1.1: Robotic assembly cell in a Volkswagen plant in Slovakia.

1.1.2 A new robotics context

In the past 20 years, robotics research has undergone important transformations driven by an increasing industrial demand for more flexible robots. Indeed, in order to diminish drudgery of work while improving precision, and to some extent increasing productivity, the diversity of tasks involving robots in industry has increased significantly. Some examples of these new tasks include bin picking and sorting (([Liu et al., 2012]), ([Lukka et al., 2014])), visual inspection and metrology (([de Sousa et al., 2017]), ([Jayaweera and Webb, 2010])), machining (([Olabi et al., 2010]), ([Chen and Dong, 2013])), kitting (([Balakirsky et al., 2013]), ([Banerjee et al., 2015])), etc.

Together with the number of different tasks, the flexibility required within the context of each task has also increased. The growing demand for mass customization ([Zawadzki and Żywicki, 2016]) together with the apparition of new industries in the robotics market (aerospace, energy, etc.) has lead to a new industrial context called Industry 4.0 ([Bahrin et al., 2016]) or smart factory ([Lucke et al., 2008]). This new context is defined by, among other things, smaller production volumes and higher responsiveness of the production plants. To illustrate the need for more adaptive robotics applications within this context, we can consider the simple example of a screwing operation. On the one hand, in a large automotive robotics assembly line, a robot needs to drive a given number of screws that are always at the same location, and repeat this operation an important number of times. This operation is programmed by finding an optimal trajectory and a control strategy to follow such trajectory. In some sense, implementing such a robotic screwing operation does not really differ from implementing a welding or painting operation. On the other hand,

we can consider the context of satellites production in the aerospace industry, which also requires many screwing operations but are only produced in "batches" of one. The task thus becomes "screw what you encounter on the structure", which is much more challenging as cinematic alone is not sufficient and needs to be coupled with sharp perception skills. This thesis is part of the Colrobot european project[1], which also illustrates well the new context of industrial robotics.

1.1.3 Machine learning and robotics

Because of these new expectations, instead of directly dealing with applications, current research focuses on providing robots with diverse skills, which can be combined to achieve various goals. To be functional, such skills must either be robust to a variety of conditions, such as different poses of manipulated objects and different environments, or be easy to reconfigure by unqualified operators. Learning methods[2] are well suited to develop such robotic bricks and are being widely adopted in recent research. Some examples of robust robotic skills which were successfully learned are scene understanding ([Shi et al., 2016]), grasping (([Bohg et al., 2014]), ([Lenz et al., 2015])), object recognition ([Eitel et al., 2015a]), motion planning ([Dong et al., 2016]), localization and mapping ([Kim and Eustice, 2015]), trajectory generation ([Levine et al., 2015b]), etc. To obtain the desired robust behaviors, a robot must be highly equipped with sophisticated sensors. Processing such complex information and link it to robot movements is a difficult task that in many cases can only be solved with learning methods. In the meantime, the machine learning framework is also well adapted to design applications that are easily reconfigurable by unqualified human workers, as suggested by the path followed by the field of human-robot collaboration (([Tsarouchi et al., 2017]), ([Munzer et al., 2017])).

This thesis fits within this adaptable and customizable robotics context. The different contributions that are presented in this manuscript aim at:

- providing the community with new algorithmic tools and datasets to develop such kind of robust robotics skills,
- demonstrating feasibility of the proposed approaches on real robotic systems.

The main focus of this work, which constitutes most of this thesis, is on the development of object understanding and sorting applications that are context independent

[1] `https://www.youtube.com/watch?v=8zpYzVEw-Io`

[2] A beginner's introduction to what is machine learning can be found in Appendix A.

and robust. These topics are studied within the context of Unsupervised Robotic Sorting, which is introduced in the next section. Along the same line, we also propose a more generic method for trajectory learning, which is easier to set up by non experts and to embed into more complex learning frameworks. Our last contribution consists in defining a methodology for autonomous dataset construction for object stereo localization. These contributions as well as the thesis organization are better detailed in Section 1.3 and more specific bibliographical elements are discussed in each chapter.

1.2 Unsupervised Robotic Sorting

In this section, we propose a broader introduction of the unsupervised robotic sorting application, which motivates many of the contributions of this thesis.

1.2.1 Context

1.2.1.1 Autonomous sorting

The problem of automatic sorting has a long history in industry, with the first tomatoes sorting system dating back to the 70's ([Husome et al., 1978]). Since then, it has received a lot of attention, with important focus on combining computer vision and robotic manipulators to solve the pick-and-place task ([Mouli and Raju, 2013]).

Although it was among the first robotics tasks, designing a pick-and-place application is challenging and requires solving multiple subtasks. If the objects to be sorted are cluttered, the system first needs to segment the scene and identify the different objects ([Ecins et al., 2016]). Then, the system must use various sensors, such as 2D or 3D cameras, color sensors or bar code readers, to gather data about the different objects. These data are used to find a grasping strategy for the object ([Bohg et al., 2014]) and to decide how to classify it. Naturally, all along this process, smart motion planning and control is also required ([Siciliano and Khatib, 2016]).

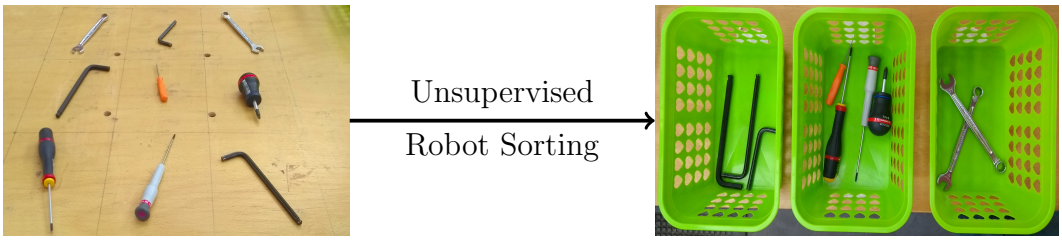
Although every bricks of the pick-and-place pipeline are interesting and challenging problems, this thesis focuses more on the decision making subproblem, i.e. how to decide where to place an object after grasping it? Other skills that can also be useful for robotic sorting are studied in the final part of the manuscript.

1.2.1.2 Decision making in autonomous sorting

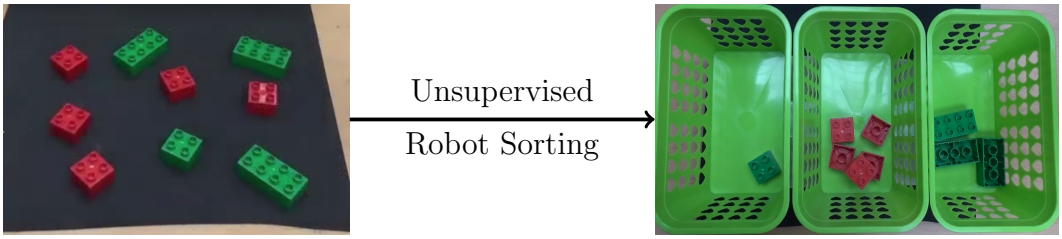
Previous implementations of autonomous sorting dealt with objects which are either known or belong to some predefined classes. In machine learning terminology, one could say that the robot intelligence usually resides in solving either an instance retrieval or a supervised classification problem. Different approaches in solving such decision-making problems differ in the kind of sensing devices used, as well as the algorithmic choices.

Instance retrieval Several implementations have considered the problem of recognizing each object as a member of a known database, and sorting it according to an associated predefined rule. In ([Giannoccaro et al., 2013]), RFID markers are used to recognize and sort objects. An implementation of a surgical tools sorting application using barcode reading and template matching to find and sort objects from a cluttered scene is proposed in ([Tan et al., 2015]). Finally, in the application presented in ([Dragusu et al., 2012]), real-world objects (usb, glue-stick, etc.) are sorted with a serial manipulator using template matching.

Supervised classification Many robotics supervised classification applications sort objects based on simple rules such as thresholding, applied to simple features. For example, (([Szabo and Lie, 2012]), ([Shum et al., 2016]) and ([Nkomo and Collier, 2012])) all use color features extracted from either images or color sensors, ([Pereira et al., 2014]) extend these approaches by adding shape features. In ([Gupta and Sukhatme, 2012]), robot manipulation is used to unclutter Duplo bricks, which are then sorted according to their length and color. In ([Singh et al., 2016]), a classical computer vision framework is implemented to carry out faulty parts removal. A robotic system able to recycle construction wastes using features provided by metal detectors and sensors sensitive to visual wavelengths is presented in ([Lukka et al., 2014]). Recent papers have started to use Convolutional Neural Networks (CNN) to carry out supervised classification in robotic sorting. The authors of ([Eitel et al., 2015b]) have used a CNN on RGBD images to find the class of an object and sort it. In ([Zhihong et al., 2017]), a fast R-CNN proposal network is combined with a pretrained VGG16 CNN architecture to jointly localize and sort objects with a robot manipulator.



(a) Tools in a work station



(b) Lego bricks of different colors and shapes

Figure 1.2: Two different instances of the unsupervised robotic sorting problem.

1.2.2 Problem definition

Chapters 2 to 6 of this manuscript are about a new kind of pick-and-place application, consisting in sorting unknown objects, which do not belong to any predefined class. Given a set of previously unseen objects, the robotic system needs to sort them such that objects stored together are similar to each other and different from other objects. Such problem definition is the definition of a clustering problem (see Chapter 2 for more details) in which the outputs consist in physically sorted objects, hence, we call this problem Unsupervised Robotic Sorting (URS).

The above statement of the URS problem neither defines the notion of similarity nor the number of groups to be made. Different definitions of these concepts defines different instances of the URS problem. This thesis mainly addresses the URS problem in which the number of groups is imposed by the number of storage spaces available and the notion of similarity is defined by "how a human would solve the problem". Figure 1.2 shows the inputs and associated expected outputs for two practical examples of URS. It is important to note that the notion of reproducing human decision is highly subjective as the optimal classification can vary between individuals. More specifically, in this thesis we aim at reproducing the decision which would be mostly adopted by a group of human experts at the classification task at hand. For example, specialist doctors for unsupervised classification of medical images, qualified workers for industrial classification tasks, etc. In practice, we validate

our algorithms by trying to reproduce the classifications of several standard supervised datasets without using the labels. These datasets are created by human experts and can thus be used as proxies for the expert populations described above. More details on the validation methods can be found in the relevant chapters.

1.3 Contribution and thesis organization

The rest of this thesis is organized as follows. Chapters 2 to 6 deal with the URS problem introduced previously. Chapter 2 complements the introduction of URS by proposing a first implementation. This unsupervised sorting application is based on color and shape features and leads to the development of a new K-means clustering algorithm to handle such kind of data when they are noisy. The contributions of Chapter 2 were published as a conference paper at CCSEIT 2017 ([Guérin et al., 2017]).

In Part II of this thesis, we propose to study the problem of Image Clustering (IC), which is a required skill to implement URS with a higher level of abstraction. Hence, Chapter 3 proposes a benchmark study on solving IC by transferring knowledge from deep Convolutional Neural Networks (CNN) pretrained on large and versatile datasets. Then, Chapter 4 builds on the results from Chapter 3 to propose an ensemble clustering approach to leverage information from several such CNNs. This is shown to improve state-of-the-art results at image clustering. The studies in Chapters 3 and 4 were published in the proceedings of two conferences, respectively AIFU 2018 ([Guérin et al., 2018c]) and BMVC 2018 ([Guérin and Boots, 2018]).

After studying image clustering on standard datasets, we propose to study practical URS implementations in Part III. In Chapter 5, we introduce a first implementation and evaluate its robustness to various external factors. This study suggests that the results are highly dependant on the point of view under which the objects are observed, which motivates the development of an optimal view selection method in Chapter 6. This contribution consists in creating a large multi-view dataset and use to train a neural network to choose optimal camera poses. This approach is shown to work better than fixed camera poses. The physical implementation of URS proposed in Chapter 5 has been published in the International Journal of Artificial Intelligence and Applications ([Guérin et al., 2018d]), and the developments on optimal view selection from Chapter 6 can be found in the proceedings of IROS 2018 ([Guérin et al., 2018a]).

The contributions about object understanding proposed in Part II and III are important skill for many autonomous robotic tasks. However, many other abilities are required to reach full autonomy, as discussed in Section 1.2.1.1 in the context of URS. Hence, in Part IV, we propose to explore two of such skills. On the one hand, Chapter 7 presents an adaptation of a popular trajectory learning method. By learning a quadratic model of the cost function instead of computing it analytically, this method is made independent of both the system's model and the cost function's definition, thus making the system easier to program in more situations. This contribution resulted in the publication of two conference papers: ([Guérin et al., 2017]) and ([Guérin et al., 2016]), which were presented respectively at ACD 2016 and IECON 2016. On the other hand, Chapter 8 proposes a methodology to autonomously build datasets for 3D stereo localization using a robot manipulator. One such dataset is built as a proof of concept and made publicly available. A short paper on this thematic can be found in the proceedings of IECON 2018 ([Guérin et al., 2018b]).

Finally, Part V proposes a summary of our contributions and draws perspectives for future work. All the different research items introduced in this thesis share the same objective of developing industrial robotic application with a broader validity range. We believe that it can potentially lead to easier and more accessible robot programming, and by extension help the democratization of robotics.

Chapter 2

First example of URS: clustering from color and shape features - the Gap-Ratio K-means algorithm

Abstract

The most standard robotics sorting setting consists in using color and shape features to carry out supervised classification. To illustrate the Unsupervised Robotics Sorting (URS) problem, we propose to extend this use case to the unsupervised setting. This application presents various challenges: 1. Color and shape features are, in general, of a different order of magnitude. 2. In robotics sorting, data are measured from real world objects in shop floor conditions (i.e., uncontrolled lighting), which involves a large spread in the data (i.e., noisy data). 3. Data dimensions are on different levels of measurements: RGB color features are interval type variables while length features are ratio type variables. To overcome these difficulties, a new weighted K-means algorithm, called Gap Ratio K-means, is introduced. It consists in defining weights which capture information about the relative differences between consecutive data points for each feature dimension. Gap Ratio K-means is evaluated on a real world example consisting in physically clustering Lego bricks of different colors and shapes. It is compared with two other variants of K-means and demonstrates more robust clustering results.

2.1 Problem description and challenges

As mentioned in the previous chapter, the standard approach to robot sorting consists in solving an instance retrieval or supervised classification problem on objects characterized by color and shape features. In this chapter, we propose to adapt such kind of sorting application to the unsupervised case (Figure 2.1).

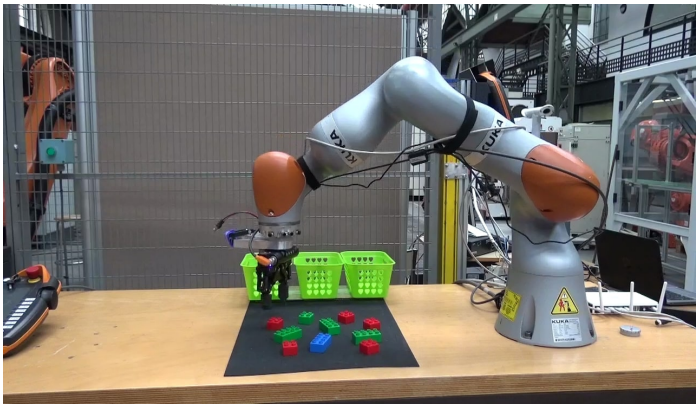


Figure 2.1: KUKA LBR iiwa sorting objects from color and shape features using the Gap-Ratio K-means algorithm.
Video at : `https://www.youtube.com/watch?v=korkcYs1EHM`

Unsupervised sorting of real world objects represented by color and shape features presents various challenges: Firstly, in the general case, length and color *features are of a different order of magnitude*, which implies the need for data normalization. Secondly, because such application is meant for ordinary environments, the clustering algorithm needs to be robust to unmastered light conditions, which is synonymous with *widely spread datasets*. For such noisy data, the loss of information involved by normalization can impact negatively the clustering results. A way to solve this issue is to use weighted K-means algorithms to re-inject the information in the normalized data. Finally, the features chosen are on *different levels of measurement* ([Stevens, 1946]): RGB-color features are interval variables whereas lengths are on a ratio scale. Statistics using ratios or means (e.g., Coefficient of Variation (CV)) cannot be used on interval type data. Hence, all the weighted K-means algorithms using this kind of statistics cannot be used for the proposed problem. These three specificities of the clustering problem motivate the development of a *new weighted K-means algorithm*. This clustering method, called Gap-Ration K-means (GRKM), is the main contribution of this chapter.

2.2 Preliminaries

2.2.1 Clustering

The sorting problem described above is a clustering problem (([Theodoridis and Koutroumbas, 2006b]), ([Duda et al., 2001])), also called unsupervised classification. Given an unlabelled dataset, the goal of clustering is to define groups (called clusters) among the entities. Elements in one cluster should be as similar as possible to each other and as different as possible to other clusters' members. In this chapter, only clustering with predefined number of classes (imposed by the number of bins in which objects can be stored) is studied. There are many possible definitions of similarity between data points. The choice of such definition, together with the choice of the metric to optimize, defines a clustering algorithms. The two surveys ([Xu and Wunsch, 2005]) and ([Berkhin, 2006]), give two slightly different taxonomies of the various clustering algorithms.

For robotics sorting applications, the number of objects to cluster might be small (≈ 10), which makes some deep clustering ([Aljalbout et al., 2018]) and density based ([Ester et al., 1996]) methods irrelevant. After trying several clustering algorithms on simulated color and shape datasets using scikit-learn ([Pedregosa et al., 2011]), K-means ([Theodoridis and Koutroumbas, 2006c]), a partitioning method, appeared efficient for our problem. Therefore, among all clustering methods, this chapter focuses on K-means.

2.2.2 K-means

2.2.2.1 Notations

Throughout this chapter, the following notations are used. Letter i represents indexing on data objects whereas letter j designates the features. Thus, $X = \{x_1, ..., x_M\}$ represents the dataset to cluster, composed of M data points. Each data point is represented by N features and x_{ij} stands for the j^{th} component of the feature vector of object $x_i \in \mathbb{R}^N$. Likewise, the use of letter k represents the different clusters and $C = \{C_1, ..., C_k, ..., C_K\}$ is a set of K clusters. Each cluster C_k, is represented by a cluster center, or centroid, denoted c_k, which is simply a point in the feature space. We also introduce d, the function used to measure dissimilarity between a data object and a centroid. For K-means, such dissimilarity is quantified with Euclidean distance

$$d(x_i, c_k) = \sqrt{\sum_{j=1}^{N} (x_{ij} - c_{kj})^2}. \tag{2.1}$$

2.2.2.2 Derivation

Given a set of cluster centers $c = \{c_1, ..., c_k, ..., c_K\}$, cluster membership is defined by

$$x_i \in C_l \iff d(x_i, c_l) \leq d(x_i, c_k), \text{ for all } k \in \{1, ..., K\}. \tag{2.2}$$

The goal of K-means is to find the set of cluster centers c^* which minimizes the sum of dissimilarities between each data object and its closest cluster center. Introducing the binary variable a_{ik}, which is 1 if x_i belongs to C_k and 0 else, and the membership matrix $A = (a_{ik})_{\substack{i \in \{1, \ ... \ M\} \\ k \in \{1, \ ... \ K\}}}$. K-means can be written as an optimization problem:

$$\begin{aligned} \underset{A, \ c}{\text{Minimize}} \quad & \sum_{i=1}^{M} \sum_{k=1}^{K} a_{ik} \times d(x_i, c_k), \\ \text{subject to} \quad & \sum_{k=1}^{K} a_{ik} = 1, \text{ for all } i \in \{1, ..., M\}, \\ & a_{ik} \in \{0, \ 1\}, \text{ for all } i, k. \end{aligned} \tag{2.3}$$

In practice, (2.3) is optimized by solving iteratively two subproblems, one where the set c is fixed and one where A is fixed. The most widely used algorithm to implement K-means clustering is the Lloyd's algorithm ([Lloyd, 1982]). It is based on the method of Alternating Optimization ([Bezdek and Hathaway, 2002]), also used in the Expectation-Maximization algorithm ([Dempster et al., 1977]). The Expectation step (E-step) consists in associating each data point to it's closest cluster center following (2.2), i.e. computing A. The Maximization step (M-step) consists in computing the centroids minimizing the total dissimilarity within clusters, i.e., computing c. When the L^2 norm is used for dissimilarity, which is the case for K-means, it can be shown that the M-step optimization is equivalent to computing the cluster mean ([Theodoridis and Koutroumbas, 2006c]).

2.2.2.3 Data normalization

In most cases, running K-means on raw data does not work well. This is particularly true when data dimensions are of a different order of magnitude: features with largest scales are given more importance during dissimilarity calculation and clustering results are biased (Figure 2.2). The kind of features studied in this chapter are impacted by this problem, colors are expressed in RGB (between 0 and 255) while lengths depend on the size of the object and the unit chosen to express it. For objects sorted with a standard manipulator, lengths usually measure a few centimeters.

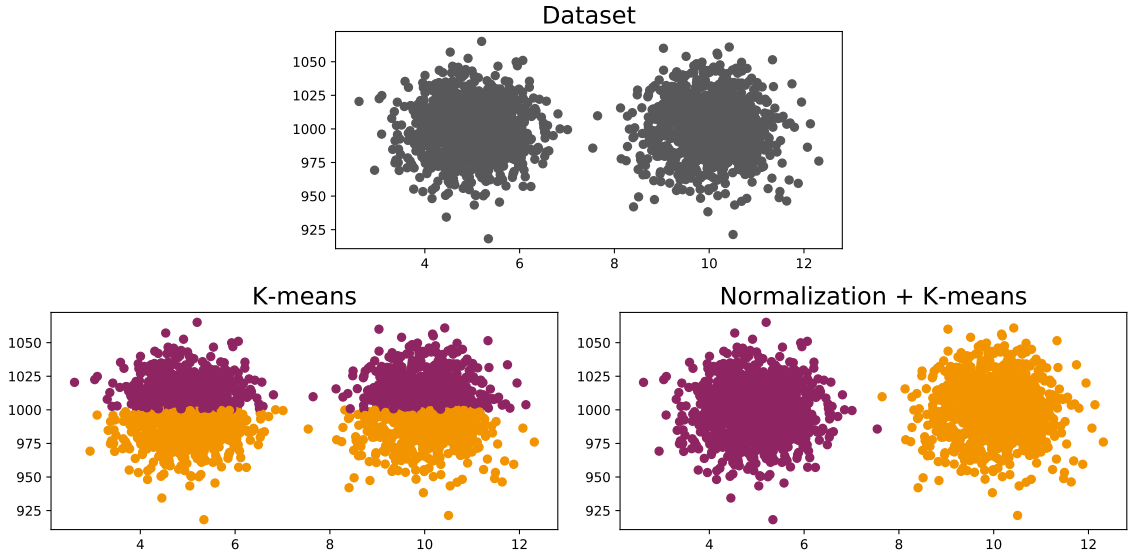


Figure 2.2: Comparison of K-means with and without normalization on a two dimensional made up dataset with dimensions of a different order of magnitude.

To deal with this issue, a common practice is to normalize the data before running the clustering algorithm:

$$\forall i,\ \forall j,\ x_{ij} \leftarrow \frac{x_{ij} - \mu_j}{\sigma_j}, \tag{2.4}$$

where μ_j and σ_j represent respectively the empirical mean and standard deviation over $\{x_{ij},\ \text{for all } i \in \{1, ..., M\}\}$.

2.2.3 Weighted K-means

As explained above, data normalization is often necessary to obtain satisfactory clustering results. However, reducing each feature distribution to a Gaussian of variance 1 can involve a loss of valuable information for clustering. This is particularly true for data with a large spread as the span of each dimension is lost during normalization. For example, Figure 2.3 shows an dataset where three groups are to be found along the same axis. We can see that, by normalizing the data, the important information is lost and K-means fails. Hence, for the color and shape clustering problem, simple normalization cannot be satisfying and other methods need to be employed.

Weighted K-means ([Chen et al., 2009]) is based on the idea that information about the data can be captured before normalization and reinjected in the normalized dataset. In this way, the most relevant features for clustering can be enlarged and the others curtailed. More precisely, in a weighted K-means algorithm, weights are attributed to each feature, giving them different importance. Let us call w_j the weight

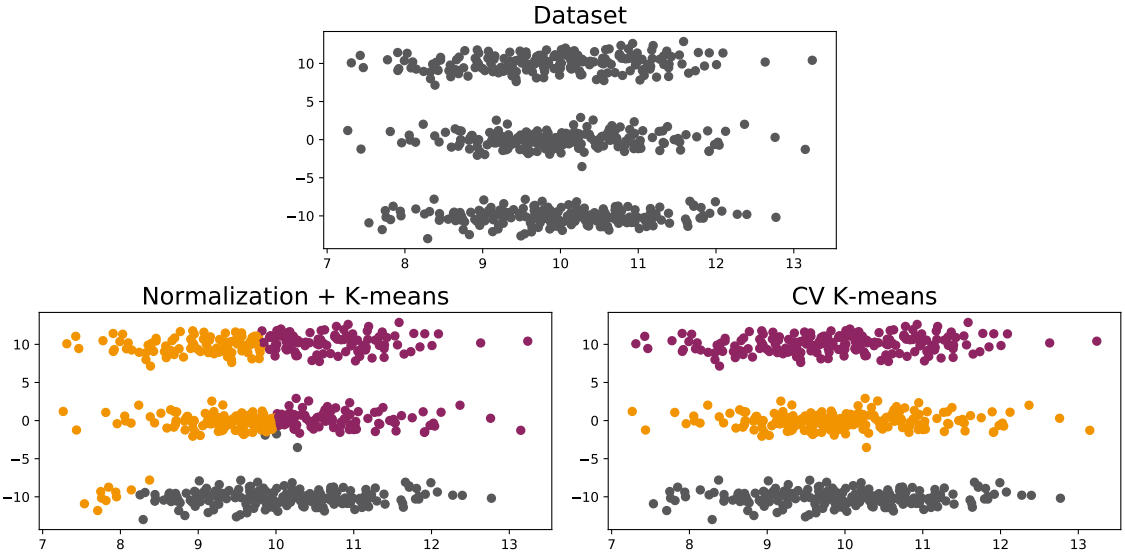


Figure 2.3: Comparison of normalization + K-means against CV K-means on a two dimensional made up dataset where several separations occur along the same axis.

associated with the j^{th} feature. Then, the norm used in the E-step of weighted K-means is

$$d(x_i, c_k) = \sqrt{\sum_{j=1}^{N} w_j (x_{ij} - c_{kj})^2}. \tag{2.5}$$

The distinction between different versions of weighted K-means algorithms lies in the choice of the weights.

A particular example of weighted K-means algorithm is weighted K-means based on coefficient of variation (CV K-means) ([Ren and Fan, 2011]). It relies on the idea that the variance of a feature is an good indicator of its importance for clustering. Hence, the CV weights are derived based on coefficient of variation, also called relative standard deviation:

$$w_j = \frac{\mathrm{cv}_j}{\sum_{j\prime=1}^{N} \mathrm{cv}_{j\prime}}, \text{ where } \mathrm{cv}_j = \frac{\sigma_j}{\mu_j}. \tag{2.6}$$

In the example of Figure 2.3, CV weighted K-means enables to overcome the problem involved by data normalization.

2.2.4 Exponentiated weighted K-means

In this chapter, we also propose an extension of weighted K-means that consists in raising the weights to the power of an integer p in the norm formula:

$$d(x_i, c_k) = \sqrt{\sum_{j=1}^{N} w_j^p (x_{ij} - c_{kj})^2}. \tag{2.7}$$

By doing so, we emphasize even more the importance of features with large weights, which makes sense if the information captured by the weights is relevant. In practice, as the weights are between 0 and 1, p should not be too large to avoid considering only one feature. In the rest of this chapter, CV^p denotes exponentiated CV K-means with exponent p. The influence of p in the clustering results is studied in Section 2.4.3.

2.2.5 Levels of measurements

Using CV weights to solve the unsupervised robotics sorting problem using color and length features cannot be robust. The reason for this statement lies in the concept of levels of measurement ([Stevens, 1946]). More specifically, it comes from the difference between ratio scale and interval scale.

A quantity measured on a ratio scale is one for which ratios are meaningful. For instance, lengths are measured on a ratio scale and one can assert that "10 cm is twice as large as 5 cm". A ratio scale possesses a unique and non-arbitrary zero value, and if a quantity gets further from this zero, the absolute precision to measure it decreases and thus the variance increases proportionally. Hence, coefficient of variation makes sense on a ratio scale.

On the other hand, interval scale is a measurement scale where all possible values are spread between two defined points, for instance the freezing and boiling point for temperatures in Celsius. From a precision perspective, measurement errors are not larger for high values than for low ones. On an interval scale, it is not relevant to use coefficient of variation because when the mean decreases, the variance does not change accordingly. Therefore, at equal variance, features closer to zero have higher coefficients of variation, which biases the clustering process. RGB color features, which are spread between 0 and 255, are interval variables. Hence, CV weights are not relevant for the problem studied in this chapter, which motivates the development of a new weighted K-means algorithm in the next section.

2.3 Gap Ratio K-means

For interval-type variables, ratios are not meaningful and one cannot use moments about the origin (e.g., the mean) as its choice is arbitrary. Hence CV weights are not appropriate for interval variables. However, ratios of differences are meaningful and standardized moments can be defined. Thus, we propose an approach that only *considers relative values between the data points*. Such choice leads to the definition of weights that are relevant for interval type data.

2.3.1 The Gap Ratio K-means algorithm

When doing clustering, we want to distinguish if different feature values between two objects come from noise or from the fact that objects are of different nature. This observation is the main motivation behind the development of Gap Ratio K-means (GR K-means). If we consider that the distribution of a certain feature differs between classes, this feature's values should be more different between objects of different classes than between objects within a class. The goal of GR weights is to capture such information about the features.

To formulate the concept of Gap Ratio mathematically, the different values x_{ij} for each feature are first sorted according to the j^{th} component. For every j, we define the permutation σ_j such that

$$\forall i, i' \in \{1, ..., M\},\ x_{\sigma_j(i),j} \leq x_{\sigma_j(i'),j} \Leftrightarrow \sigma_j(i) \leq \sigma_j(i'). \tag{2.8}$$

Then, we define the i^{th} gap of the j^{th} feature by:

$$g_{ij} = x_{\sigma_j(i+1),j} - x_{\sigma_j(i),j}. \tag{2.9}$$

If there are M data objects, there are $M-1$ gaps for each feature.

After computing all the gaps for feature j, we define I_j, the index corresponding to the biggest gap:

$$I_j = \underset{i \in \{1,...,M-1\}}{\operatorname{argmax}} g_{ij}. \tag{2.10}$$

Then, the biggest gap G_j and the mean gap $\bar{g}_j$ are defined as follows:

$$G_j = g_{I_j,j}, \tag{2.11}$$

$$\bar{g}_j = \frac{1}{M-2} \sum_{\substack{i=1 \\ i \neq I_j}}^{M-1} g_{ij}. \tag{2.12}$$

Finally, we define the Gap Ratio for the j^{th} feature by:

$$\mathrm{gr}_j = \frac{G_j}{\bar{g}_j}. \tag{2.13}$$

In other words, for a given feature, the Gap Ratio is the ratio between the highest gap and the mean of all other gaps. Then the GRs are used to compute scaled weights:

$$w_j = \frac{\mathrm{gr}_j}{\sum_{j\prime=1}^{N} \mathrm{gr}_{j\prime}}. \tag{2.14}$$

The dissimilarity measure for GR K-means is obtained by using weights (2.14) in (2.5). Likewise, we call exponentiated GR K-means the algorithm using dissimilarity measure (2.7) with weights (2.14). Exponentiated GR K-means with exponent p is denoted GR^p K-means.

2.3.2 Intuition behind GR K-means

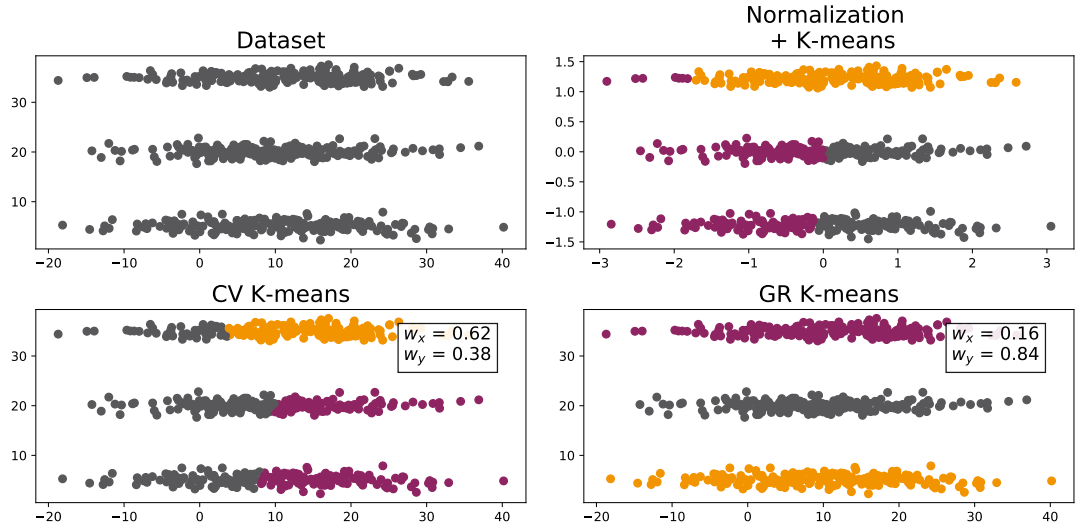
For datasets that are noisy in dimensions that are not important, CV K-means fails because it focuses on these dimensions with high variance. This problem is illustrated on Figure 2.4a, which shows a 2-dimensional toy example where the variance is high in a dimension which is not important. On the other hand, weights and groups obtained with GR K-means indicate that the right information is stored in GR weights for such problem. The biggest gap along the y-axis is much bigger than average gaps whereas these two quantities are similar along the x-axis. In this case, using GR weights is more appropriate than CV weights.

CV weights also fail for interval variables with means close to zero. Indeed, because the variance does not decrease for low values on an interval scale, the coefficient of variation for these dimensions goes to very high values. This problem is illustrated in Figure 2.4b, where we can see that CV K-means completely fails while GR K-means is not affected and still succeeds to cluster the data correctly. This example is particularly relevant for the clustering case studied in this chapter. Indeed, RGB features are on an interval scale and data are noisy, which can lead to such kind of misclassifications.

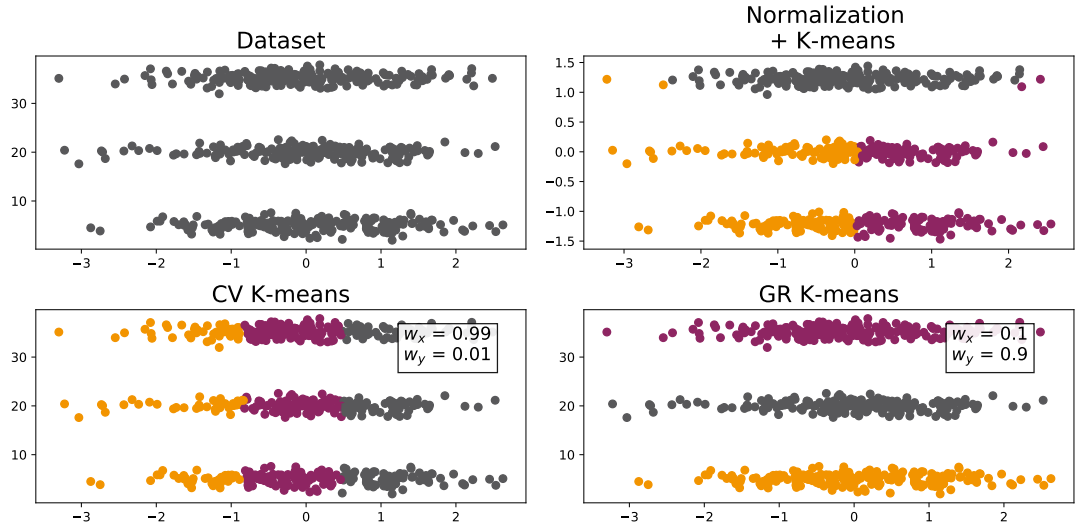
2.4 Experimental validation

2.4.1 Experiment description

In this section, we validate the intuitive reasoning about GR K-means. To do so, we compare the different weighted K-means algorithms (including regular K-means,



(a) Noisy data on a dimension that is not important.



(b) Interval type data with low mean.

Figure 2.4: Made up two dimensional toy examples demonstrating the intuition behind GR K-means.

with weights $w_j = 1,$ for all j, and exponentiated weights) on a first implementation of unsupervised robotic sorting. This implementation consists in sorting a set of Lego bricks of different sizes and colors using a Kuka LBR iiwa collaborative robot equipped with a camera. The application is illustrated in Figure 2.1, which gives a link to a demonstration video.

The dataset used to compare different algorithms is one composed of nine Lego bricks of different sizes and colors, as shown in Figure 2.5. Each Lego brick is represented by its length and width (in cm), as well as its three RGB components (between

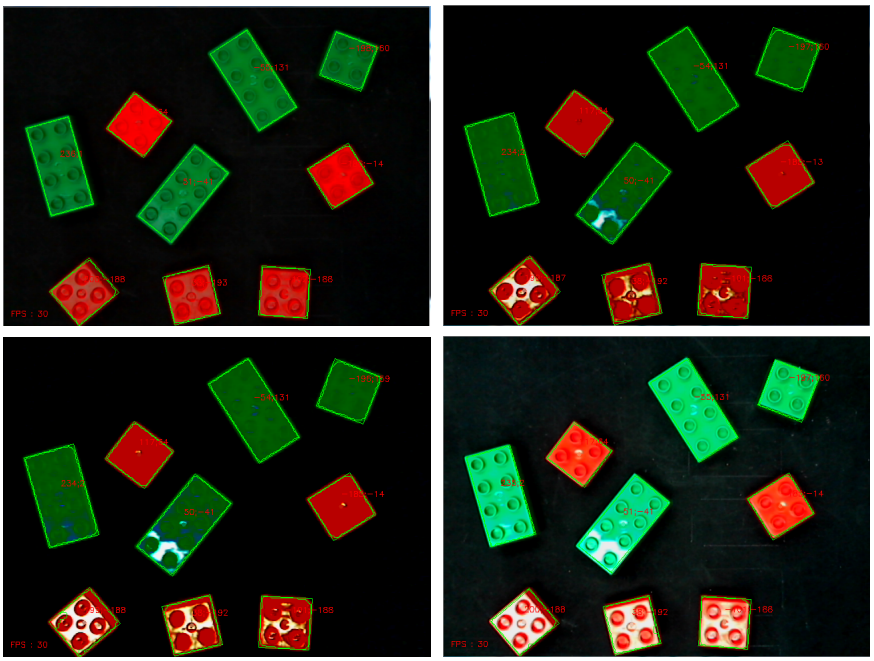


Figure 2.5: Example real-world URS dataset under different light conditions.

0 and 255). The natural classes in this dataset are straightforward to define, the clustering algorithm needs to place the big green, small green and small red bricks in different bins. Hence, the desired number of classes is set to three.

Furthermore, on Figure 2.5, we can see that among the four pictures, lighting varies significantly and that data are very noisy. Color features observed are really different between two runs of the application. The algorithm needs to be robust to poor lighting conditions and to be able to distinguish between red and green even when colors tend to be saturated (see bottom right image). Feature extraction is implemented with open CV ([Itseez, 2015]), using background segmentation to obtain the contours of each object and pixels averaging for color features.

2.4.2 Weighted K-means implementation

To implement both GR K-means and CV K-means, we use the K-means implementation of the open-source library scikit-learn ([Pedregosa et al., 2011]). This way, our results can be checked and further improvements can be tested easily. To implement weighted K-means algorithms, we also use scikit-learn implementation but on a modified dataset. After normalization, the initial data are transformed using the following feature map:

$$\Phi : x_{ij} \to \sqrt{\omega_j} x_{ij}. \tag{2.15}$$

As the dissimilarity computation appears only in the E-step, the dataset modifications are equivalent to changing the norm. Indeed, Equation (2.5) is equivalent to

$$d(x_i, c_k) = \sqrt{\sum_{j=1}^{N} (\sqrt{w_j} x_{ij} - \sqrt{w_j} c_{kj})^2}. \tag{2.16}$$

By doing this, results obtained can be compared more reliably. Differences in the results are less likely to come from poor implementation as the K-means implementation is always the same. Following these steps, implementation is straightforward.

2.4.3 Results

For our experimental comparison of the different algorithms, the experiment is repeated 98 times with different arrangements of the Lego bricks presented on Figure 2.5, and with different lighting conditions. For each trial, if the algorithm misclassifies one or more bricks, it is counted as a failure, else, it is a success. Figure 2.6 presents results obtained on the 98 trials for different weighted K-means algorithms. The "Original dataset" contains the real measured data and the "Slightly modified dataset" is the same dataset with a slight color modification. To test the robustness of the algorithms, we removed 50 to the blue component of the bricks, which corresponds to using bricks of slightly different colors. The notation p^* represents the optimal exponent for the weights for each algorithm (its values can be found in Section 2.4.4). All the results presented in this section are computed on the scaled datasets. Without scaling, success rates are very low (around 5%) because lengths ($\approx 5cm$) and colors (≈ 150) are of a totally different order of magnitude.

The first observation from these results is that regular K-means always results in very poor classification. One possible explanation for such bad behavior is the relatively high spread in the data (due to lighting conditions), which involve high variance in the features values and makes it difficult to differentiate between noise and true difference of nature. For this reason, emphasizing certain features is required, which justifies the use of weighted K-means.

Considering the weighted K-means algorithms, CV K-means performs particularly well on the original dataset. However, after little modifications, it falls into very bad behavior. Such issue with CV K-means comes from the fact that coefficient of variation is not appropriate for interval scale data. In other words, CV K-means can succeed on the original dataset only because the bricks used do not present RGB

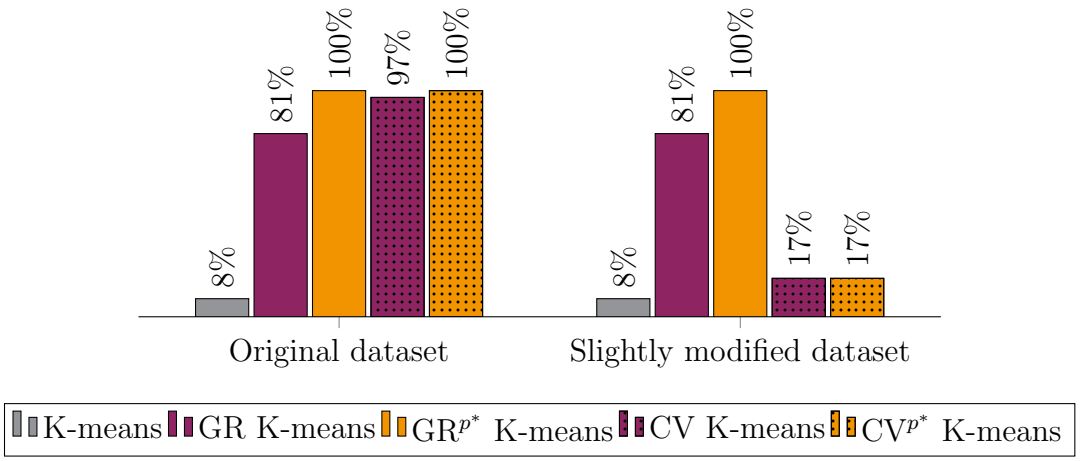


Figure 2.6: Percentage of experiments with zero misclassification. The experiment was run 98 times under different lighting conditions and with different layouts of the bricks. Error rates presented are averaged among these 98 runs.

components too close to zero. On the other hand, GR K-means performs reasonably well ($\approx 20\%$ error rates), and is stable to dataset modifications.

Finally, looking at the optimal versions of the exponentiated weighted K-means algorithms, we see that exponentiated GR K-means can reach perfect classification on both datasets while CV weights cannot produce good classification on the modified dataset. A study of the exponents is proposed in the next section.

2.4.4 Influence of the weights exponents

Another way to determine the relevance of the information stored in the weights is to look at different values of the exponent for exponentiated weighted K-means. If the weights computed by a given algorithm are properly balanced, then increasing their importance by increasing p should improve the clustering result. On the other, if the clustering results decrease when p increases, then the weights are probably not very well balanced and too much importance is given to certain features. In this section, different values of p are tried to evaluate the "stability" of the weights found by both GR and CV K-means. Figure 2.7 represents the evolution of the clustering results (averaged on the 98 runs) when the value of the weights exponent increases, for both the original and the modified datasets.

The first thing to be noticed from these plots is that the two curves for GR weights are superimposable. This shows that GR K-means algorithm is insensitive to average values of the interval scaled features. As for CV K-means, it performs good under

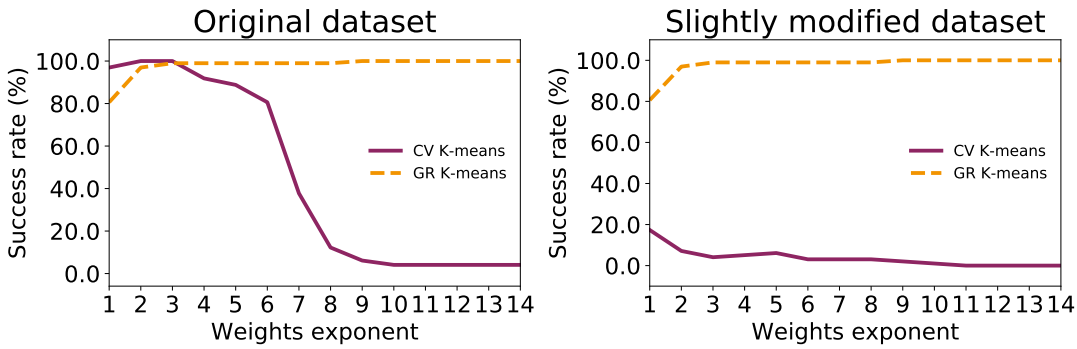


Figure 2.7: Exponent influence for Lego bricks clustering with exponentiated weighted K-means algorithms.

certain conditions (Original dataset) but is not robust to decreasing the mean value of one feature (Modified dataset).

The left plot of Figure 2.7 shows another interesting thing. For low value of p, CV K-means performs better than GR K-means (0% error rate for $p = 3$). Even if the most important thing is to have one exponent for which error is low, it is interesting to note that high exponents generate poor clustering results with CV weights. Such behavior shows that the weights are not so relevant because if they are given too much importance, clustering gets worse. On the other hand, with exponentiated GR K-means error rate tends to decrease when the exponent increases. Information carried by GR weights is good for such clustering problem and should be given more importance. Error rate falls to zero at $p = 9$ and remains stable to exponent increases until relatively high values of p (> 20); the balance between important components is well respected within the weights. For this kind of datasets, characterized by large spread, mixed scales of measurement and relatively independent features, exponentiated GR K-means with relatively high exponent seems to be a good clustering method.

2.4.5 Extension to other data sets

On the one hand, Gap-Ratio weighted K-means was developed with a specific problem (URS from color and shape features) in mind. Hence, it is not surprising that it performs good on such datasets. On the other hand, it is also interesting to test this algorithm on other classification datasets of different nature to see how well it generalizes. Different weighted K-means methods are compared on two famous supervised learning datasets: the Fisher Iris dataset ([Fisher, 1936]) and the Wine dataset, both taken from the UCI Machine Learning Repository ([Lichman, 2013]). Table 2.1 gives some important characteristics of both datasets.

Table 2.1: Statistics of the datasets used to validate GR K-means.

Dataset	Iris	Wine
Number of instances	150	178
Number of attributes	4	13
Number of classes	3	3
Is linearly separable?	No	Yes
Data type	Real	Real and Integers
Scale of measurement	Ratio	Ratio

Supervised dataset are chosen in order to have labels to evaluate the clustering output with external metrics ([Pfitzner et al., 2009]). Hence clustering results are evaluated using both cluster purity (PUR) and Normalized Mutual Information (NMI). These two metrics produce outputs between 0 and 1, with 1 representing perfect match between the cluster assignments and the true labels. Table 2.2 summarizes clustering results for both datasets, using all previously described implementations of different algorithms. For each configuration, the algorithm was run 1000 times with different random initialization. The values reported in Table 2.2 correspond to the average scores over the different runs.

Table 2.2: Results on other data sets. Accuracy and NMI scores averaged over 1000 runs of the algorithms from different centroid initializations. GR^2 and CV^2 denote the exponentiated versions of the algorithms ($p = 2$).

	Iris		Wine	
	PUR	NMI	PUR	NMI
K-means	0.83	0.65	0.97	0.88
GR K-means	0.88	0.72	0.95	0.84
GR^2 K-means	0.96	0.86	0.86	0.63
CV K-means	0.96	0.85	0.93	0.79
CV^2 K-means	0.96	0.87	0.87	0.67

For the Iris dataset, both GR and CV K-means implementations are better than regular K-means. Moreover, increasing the weights exponent improves the quality of the clustering. This means that both gap-ratio and coefficient of variation weights are able to capture the important information for clustering. However, for the Wine dataset, the best option is regular K-means, which might mean that the data dimensions are highly correlated and that one cannot isolate certain dimensions to improve clustering.

2.5 Conclusion

2.5.1 Key findings

In this chapter, we have introduced a first example of unsupervised robotic sorting consisting in clustering real objects from color and shape information. The datasets manipulated for this applications appeared to present various challenges for clustering. Hence, to overcome the fact that the data are noisy and that some dimensions of the feature space are interval variables, we have proposed a new weighted K-means algorithm. This method, called Gap-Ratio K-means, leads to obtain better and more robust clustering results on a physical implementation of URS. Perspectives and future work regarding GR K-means are discussed in Chapter 9.

2.5.2 Limitations of hand-designed features for URS

Although it motivated the development of GR K-means, the data representation used in this chapter is very specific to certain kind of objects. Any classes of objects cannot be clustered properly from simple color, length and width features. For example, there is very little chances that different models of screw drivers would be clustered together using this kind of features (see Figure 1.2). Instead, such generic URS problem requires features with higher level of abstraction, able to understand the semantic nature of the objects at hand.

For this reason, in the next chapters, the URS decision making problem is viewed as an image clustering problem, where each object is represented by a raw picture of the object. The recent successes in transferring knowledge from the ImageNet dataset to other vision tasks motivates the use of pretrained deep Convolutional Neural Networks (CNN) to extract features for the targets image clustering tasks at hand. In the next chapter, we propose to study how to optimize knowledge transfer from ImageNet to a new target unsupervised image classification task.

Part II

Image Clustering with Pretrained CNN Features

Chapter 3

Pretrained CNN Feature Extractors for Image Clustering: a Benchmark Study

Abstract

To implement an Unsupervised Robotic Sorting (URS) application, good image clustering algorithms are necessary. Therefore, this part of the manuscript focuses on image clustering. Recently, a common starting point for solving complex image clustering problems is to use generic features, extracted with deep Convolutional Neural Networks (CNN) pretrained on a large and versatile dataset (ImageNet). However, in most research, the CNN architecture for feature extraction is often arbitrarily chosen without justification. This chapter aims at providing insight on the use of pretrained deep CNN features for unsupervised problems. Different layers from various CNN architectures, combined with standard clustering algorithms, are evaluated on 8 datasets from 4 different subtasks of image clustering: Natural object recognition, Scene recognition, Face Recognition and Fine-grained recognition. Our key findings are that: 1. for all architectures and tasks, the last layer before softmax tends outperform earlier layers; 2. the choice of the CNN architecture can have a huge impact on the clustering results; 3. knowing which architecture to use is a difficult task. The last two statements motivate the use of ensemble methods in the next chapter.

3.1 Introduction

3.1.1 From URS to Image Clustering

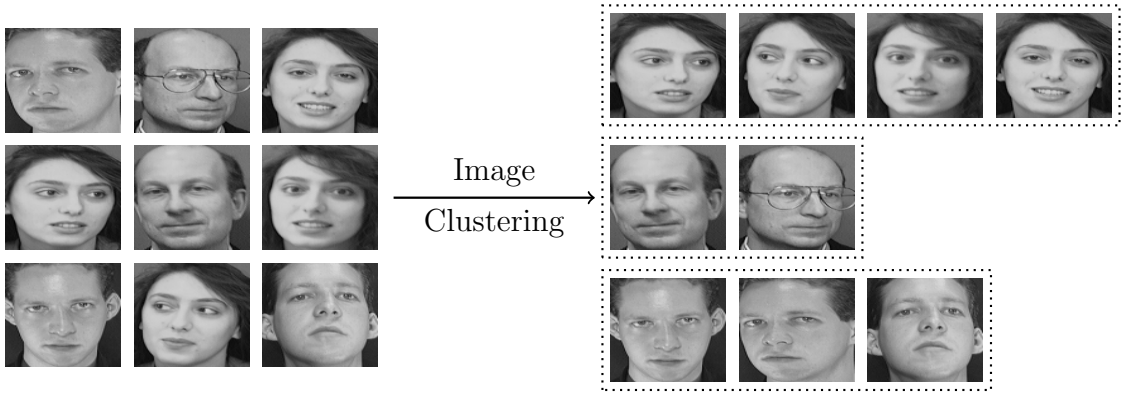
The Unsupervised Robotic Sorting (URS) problem studied in this thesis consists in physically sorting objects in an unsupervised way. The objective is to endow robots with the ability to sort and clean in a way similar to humans. Therefore, the data representation used in such an application should contain enough information to infer semantic content of the different objects and to produce a human-level unsupervised sorting. Stemming from the recent progresses in image classification (([Krizhevsky et al., 2012]), ([Simonyan and Zisserman, 2014]), ([Szegedy et al., 2016]), ([He et al., 2016])), representing the various objects by images seem a smart choice for such clustering application. Then, the decision making module of URS boils down to an Image Clustering (IC) problem. Given a set of unlabeled images, the IC (or image-set clustering) problem consists in finding subsets of images based on their content: two images representing the same object should be clustered together and separated from images representing other objects. This part of the manuscript addresses the human-level image clustering problem, which expected outputs are illustrated in Figure 3.1. The datasets presented in Figure 3.1, as well as all the other datasets studied in this chapter, are originally supervised image datasets. Although in all our experiments the labels are not used at clustering time, having supervision enables to characterize the "human-level" image clustering and gives a way to evaluate our algorithms.
Remark: This problem should not be confused with image segmentation ([Dhanachandra and Chanu, 2017]), which is also sometimes called image clustering.

3.1.2 Previous work

Image clustering has received a lot of attention over the last two decades. It has applications for searching large image databases (([Flickner et al., 1995]), ([Gong et al., 2015]), ([Avrithis et al., 2015])), concept discovery in images ([Lee and Grauman, 2009]), storyline reconstruction ([Kim et al., 2014]), medical images classification ([Wang et al., 2017]), etc. The first successful methods focused on feature selection and used sophisticated algorithms to deal with complex features. For instance, ([Goldberger et al., 2006]) represents images by Gaussian Mixture Models fitted to the pixels and clusters the set using the Information Bottleneck (IB) method ([Tishby et al., 2000]). ([Seldin et al., 2003]) uses features resulting from image joint segmentation and sequential IB for clustering. ([Fukui and Wada, 2014]) uses Bags of Features

(a) VOC2007

(b) MIT



(c) ORL

Figure 3.1: Definition of the image clustering problem. Examples of inputs and expected outputs on three natural images datasets.

with local representations (SIFT, SURF, ...) and defines commonality measures used for agglomerative clustering.

Recently, IC algorithms have shifted towards using features extracted from Convolutional Neural Networks (CNN) pretrained on ImageNet ([Russakovsky et al., 2015]). ([Liu et al., 2016]) uses deep auto-encoders combined with ensemble clustering to

generate feature representations suitable for clustering. ([Wang et al., 2017]) learns jointly the clusters and the representation using alternating optimization ([Bezdek and Hathaway, 2002]). For complex IC problems, the two papers cited above, as well as ([Gong et al., 2015]) and ([Hu et al., 2017]), use pretrained CNN feature extractors to generate a new data representation of the images before clustering.

3.1.3 Limitations

In recent research, using pretrained CNN features has enabled to obtain good clustering results on complex IC problems. However, there exists a variety of publicly available pretrained CNN architectures and, to the best of our knowledge, choosing which one to use has not been studied yet. Indeed, in all the literature mentioned in Section 3.1.2, the choice of the feature extractor architecture is never the same, and never justified. There might be several explanations for such absence of research in this direction. First, because IC is unsupervised, it is not possible to cross-validate design choices for a specific problem, making the CNN selection process very hard. Moreover, using any CNN feature extractor usually results in a huge boost in performance compared to standard computer vision features. These excellent results might feel satisfactory enough and hide the fact that a good architecture choice might improve even more the clustering performance.

3.1.4 Contributions

In this chapter, we propose to study the use of deep pretrained CNN features for unsupervised classification tasks. By carrying out an extensive set of experiments over 8 datasets representing 4 different IC tasks (see Section 3.2.1), we investigate the interrelations between the different: 1. IC datasets; 2. CNN architectures; 3. feature extraction layers; and 4. kinds of clustering algorithms. This study is intended to provide better insight on the use of pretrained CNN features for unsupervised tasks. More precisely, we want to know if

- using different architectures, although pretrained on the same dataset (ImageNet), can change the clustering results,
- some layers in these networks extract features which are better suited for unsupervised classification,
- the feature representations from different CNNs combine better with a certain type of clustering algorithm.

Our experimental results reveal that overall, the last layer before softmax is better than all the other layers for feature extraction, and this is true for every architecture and dataset. The rest of our results can be summarized in one simple sentence: *properly choosing the CNN architecture is important to have good clustering results but it is a hard task*. This conclusion motivates the introduction of a new ensemble method to generate state-of-the-art results at IC in Chapter 4.

3.2 Experiments design

This chapter aims at answering several questions about the use of deep features for image clustering. We want to know if different CNN architectures, although pretrained on the same dataset (ImageNet), behave differently when presented an unsupervised dataset. We also want to know if a CNN architecture should be "cut" in the early or late layers for feature extraction. The ideal would be to come up with generic rules such as: "when facing a particular dataset DS, and to optimize a given metric M, one should choose the architecture NN, extract features from layer L and cluster the new feature set with algorithm C". To do so, we implement the straightforward pipeline presented in Figure 3.2 for several datasets. For each dataset we try multiple combinations of NN-L-C triplets. The results of each combination is evaluated under different metrics.

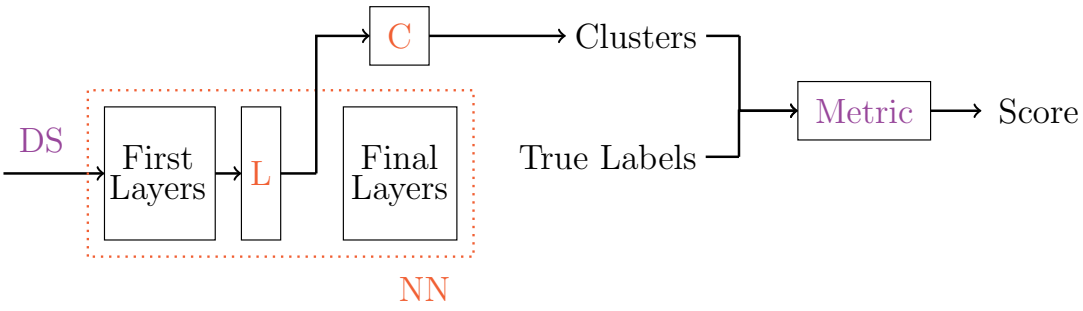


Figure 3.2: General form of the proposed Image Clustering pipeline.

The choices made for studying the different elements in the pipeline in Figure 3.2 are described in details in the coming sections.

3.2.1 Datasets

To obtain results about CNN feature extractors which are generalizable, experiments need to be carried out on many datasets, belonging to different subtasks of IC. Hence,

the proposed pipeline, with different feature extractors and clustering algorithms, is applied to the unsupervised versions of the following tasks:

- **Natural object recognition**: We call natural object recognition the task of classifying images based on a single object it contains. Classes are defined in the most generic way possible (cat, dog, car, etc.). This task is the most similar to ImageNet. Hence, although the precise task (categories) and domain (backgrounds) are different, pretrained deep features are expected to generate good clustering results on this task. Moreover, we also expect that final layers will be better suited for this task because this is the kind of objects they are separating for ImageNet classification.

- **Scene recognition**: This task is different from what the pretrained network was trained to do. Indeed, in scene recognition, a category is defined by the simultaneous presence of multiple objects on the image. For example, a dining room needs to contain chairs and a table on the image (Figure 3.1b). We still expect the last layers to perform better at this task as they are supposed to contain higher level information.

- **Fine-grained recognition**: This task is also very challenging for pretrained deep features because classes are defined within what usually defines a single category for ImageNet. For example, a fine-grained recognition task might consist in recognizing different species of birds. It is difficult because the pretrained network might have learned to produce features which are too generic for this task without additional supervision.

- **Face recognition**: This task is also a fine-grained recognition task, however, it is of such importance today that we study it as its own class of problems.

For each unsupervised classification task listed above, we pick two datasets: one small and one large. The datasets studied, together with their statistics, are listed in Table 3.1.
Hence, we study eight datasets in total. More details on these datasets can be found on the papers in which they were introduced: VOC2007 ([Everingham et al., 2007]), COIL100 ([Nayar et al., 1996]), Archi ([Xu et al., 2014]), MIT ([Quattoni and Torralba, 2009]), Flowers ([Nilsback and Zisserman, 2009]), Birds ([Welinder et al., 2010]), UMist ([Wechsler et al., 2012]), FEI ([Thomaz and Giraldi, 2010]).

Table 3.1: Statistics of the eight datasets used for the image clustering benchmark study.

Dataset	Task	# images	# classes	Images size	Balanced[1]
VOC2007[2]	Natural object	2841	20	Variable	No
COIL100	Natural object	7200	100	128 × 128	Yes
Archi	Scene	4794	25	variable	No
MIT	Scene	15620	67	variable	No
Flowers	Fine-grained	400	17	variable	Yes
Birds	Fine grained	2800	200	variable	No
UMist	Face	564	20	220 × 220	Yes
FEI	Face	6033	200	640 × 480	Yes

[1] *A dataset is balanced if it contains a similar number of instances for each classes.*
[2] *We use a modified version of the VOC2007 test set. All the images presenting two or more labels have been removed in order to be able to evaluate clustering.*

3.2.2 Architectures

To ease and speed up development, we compare the Keras ([Chollet, 2015]) implementations of five popular CNN architectures:

- Two VGG architectures: VGG16 & VGG19 ([Simonyan and Zisserman, 2014]),
- One ResNet architecture: ResNet50 ([He et al., 2016]),
- Two Inception-like architectures: InceptionV3 ([Szegedy et al., 2016]), Xception ([Chollet, 2016]).

We also use the ImageNet pretrained weights provided by Keras.

As of today, ImageNet is the only very large labelled public dataset which has enough variety in its classes to be a good feature extractor for a variety of tasks. Moreover, there are plenty of CNNs pretrained on ImageNet already available online. For these reasons, we use CNNs pretrained on ImageNet. However, the results presented here would probably apply to other databases, when larger and more diverse datasets will be created.

3.2.3 Layers

The IC problem studied in this thesis consists in discovering classes represented by "objects" present on the pictures. Thus, the feature extractor needs to gather semantic level information about the data to make such clustering possible. Such high level information is present in the final layers of the pretrained networks. Thus, to

study the impact of the layer chosen, we pick three layers among the last ones for each network :

- One layer in the end of the convolutional block (L1),
- The second layer before the ImageNet softmax layer (L2),
- The last layer before softmax (L3).

Picking layers from earlier stages of the network is both not very relevant and not practical. Indeed, the closer to the beginning of the network the layer is, the bigger the feature space is and the longer the clustering is. It is probably not relevant because the features are too low level to be informative without additional supervision.

On the one hand, the last one or two layers might provide better results as their goal is to separate the data (at least for the fully-connected layers). On the other hand, the opposite intuition is also relevant as we can imagine that these layers are too specialized to be transferable. The names (as given in the keras implementation) of the layers retained for this study are reported in Table 3.2.

Table 3.2: Names of the feature extraction layers studied.

		VGG16	VGG19	Inception	Xception	Resnet50
L1	name	block5_pool	block5_pool	mixed7	add_12	activation_40
	shape	25,088	25,088	221,952	102,400	200,704
L2	name	fc1	fc1	mixed10	block14_sepconv2_act	activation_47
	shape	4,096	4,096	131,072	204,800	25,088
L3	name	fc2	fc2	avg_pool	avg_pool	avg_pool
	shape	4,096	4,096	2,048	2,048	2,048

3.2.4 Clustering algorithms

Over the last fifty years, many clustering algorithms have been developed. Different surveys propose different classifications of the clustering methods (([Xu and Wunsch, 2005]), ([Berkhin, 2006])). However, a common bipartite classification of the different algorithms seem to emerge. The first group of algorithms are called partitioning methods. Data points are considered independently, as points in the feature space, and the clusters are created by separating the space into different areas. The other type of algorithms are called graph-based methods (or connectivity based methods) and consist in viewing the data as nodes on a graph, connected by a certain distance.

In this chapter, the goal is not to find a good algorithm to solve a specific problem but to study the influence of the chosen CNN feature extractor (architecture + layer) on the IC results. Hence, in this chapter, we only consider standard clustering algorithms in order to isolate the influence of the features. For now, we leave aside the most recent deep methods, which are state-of-the-art at clustering. They will be the purpose of the next chapter. To keep our experiments simple and interpretable, we pick the most widely used algorithms from each of these two main families of algorithms:

- K-means (KM) ([Arthur and Vassilvitskii, 2007]),
- Agglomerative Hierarchical Clustering (Agg) ([Murtagh, 1983]).

For both algorithms, we use the default configuration of the scikit-learn implementations ([Pedregosa et al., 2011]). This avoids fine-tuning our clustering algorithms specifically for deep pretrained features.

There exist a variety of other simple and very popular clustering methods (([Xu and Wunsch, 2005]), ([Berkhin, 2006])). The ones available in scikit-learn have been tried ([Guérin et al., 2018c]) and did not present a major interest with respect to the conclusions drawn from these experiments. However, keeping connectivity based and graph based algorithms enable us to analyze if different architectures represent data differently.

3.2.5 Metrics

Although we do not use the labels, our experiments are carried out on datasets that are inherently supervised. Hence, we can use external validation metrics ([Pfitzner et al., 2009]) to evaluate the quality of the clustering for the different combinations. Thus, we use two popular external clustering metrics in our study.

- **Normalized Mutual Information** (NMI), which is a metric based on information theory and defined as:

$$NMI(Y, C) = \frac{2 \times I(Y, C)}{H(Y) + H(C)}, \tag{3.1}$$

  where Y is the list of ground truth labels, C the cluster assignments, $H(.)$ represents the entropy and $I(Y, C)$ the mutual information between Y and C. NMI ranges between 0 and 1, with 1 representing perfect accuracy.

- **Cluster purity** (PUR) is defined by

$$PUR = \frac{1}{N} \sum_{c \in C} \max_{y \in Y} \mid c \cap y \mid, \tag{3.2}$$

were N is the number of elements in the dataset. Purity measures how much each cluster contains a single class, it also varies between 0 and 1, and a good algorithm has a purity close to 1.

3.3 Results

This chapter aims at studying the behaviour of different deep feature extractors for image clustering. Different architectures, combined with different layers, are tested for feature extraction. Because of the high number of experiments carried out in this benchmark study, the complete results are only presented in the Appendix to improve clarity. The full tables of results for our experiments can be found in Appendix B, they contain NMI and purity scores, as well as clustering time for the eight datasets. We also represent the different NMI scores in the form of histograms in Appendix C to visualize better the influence of the features on the clustering results. The body of the chapter only presents a summary of these results in order to highlight the important information.

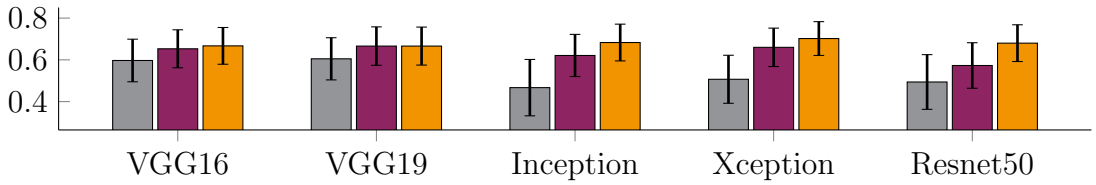
For completeness, Appendix B also includes results using bag of sifts features (BoF) representations ([Fukui and Wada, 2014]). This enables to compare CNN features with standard computer vision features and we can see that although BoF features produce decent purity scores, the NMI scores are much below deep features. We note that BoF results only appear for the smallest dataset of each task because BoFs are expensive to compute and of limited interest for our study.

To evaluate the influence of specific components of the clustering pipeline, we consider our experiments as a 4-dimensional design space which dimensions are: architecture, layer, task, clustering algorithm. The correlation between two factors is then studied by computing the mean and standard deviation (std) over all the results containing them. These two statistics enable us to evaluate both the overall performance and the stability of a combination. Although our experiments are relatively small to draw general conclusion, they enable to give a general trend.

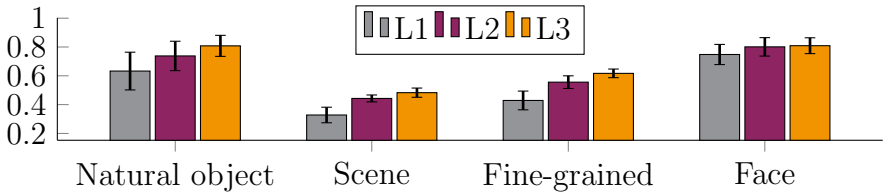
3.3.1 Influence of the layer

This results section begins by studying the impact of the choice of the layer on the clustering results. We want to know how different architectures perform under

different layers. The relation between the clustering task and the position of the layer in the network is also studied. Figure 3.3 presents a summary of our experimental results regarding the impact of the layer chosen.



(a) Layer-architecture interaction
(mean and std across tasks and clustering algorithms).



(b) Layer-task interaction
(mean and std across architectures, datasets and clustering algorithms).

Figure 3.3: Influence of the layers on the clustering results (NMI).

The relation results under the different architecture-layer pairs can be visualized in Figure 3.3a. For all architecture, NMI scores for later layers have higher mean and lower standard deviation. This reveals that, for our experiments, final layers perform better overall and are more consistent. The high standard deviation of earlier layers shows that in some cases, features extracted from early layers can perform well, however, there is more variability and the results can drop to much lower scores in other cases. Such statement can be analyzed in more detailed looking at the plots in Appendix C. For example, for face recognition, some L1 layers present slightly better results than other layers, on the other hand, for fine-grained recognition, choosing L1 can results in NMI scores lower than L3 by about 0.35.

The influence of the layer on different image clustering tasks is represented in Figure 3.3b. Before conducting these experiments, our intuition was that early layers would be better suited for face recognition and fine-grained classification tasks while late layers would be better at natural object and scene recognition. Indeed, high level information about an image are contained in the last layers while early layers represent lower level information (Gabor filters, color blobs, etc.) ([Yosinski et al., 2014]). However, our experimental results show that whatever the task is, later layers

perform better. This effect is damped for faces but it is still true. Moreover, for all tasks, std is higher for early layers.

These results suggest that only the last layer before softmax (L3) should be considered for clustering. Although in some cases other layers slightly outperform L3 (e.g., L2 for Xception on FEI), the profit is small and the risk is high (high std). Such finding implies that some "low-level" information is contained in the last layers of deep CNNs pretrained on ImageNet. This may be explained by the presence of fine-grained recognition classes in the ImageNet dataset (e.g., different breeds of dogs). Hence, only L3 layers are considered in the rest of this results section. Dropping the first layers is also motivated by the fact that their feature spaces are of higher dimensions, which means higher clustering time. For example, on average, clustering L1 layers take about one hour while L3 layers only take one minute. This difference is proportional to the size of the dataset.

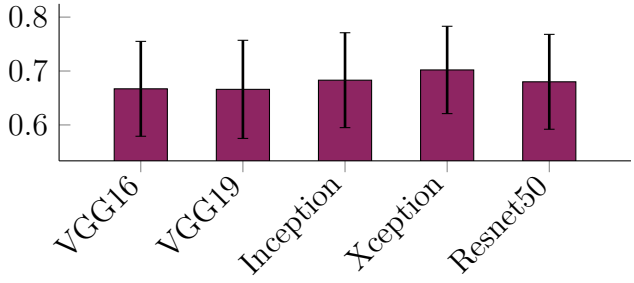
3.3.2 Influence of the architecture

The next analyses focus on the choice of the CNN architecture. These results can be found in Figure 3.4. We want to know if an architecture is better suited for clustering than the others in general (Figure 3.4a) and depending on the task at hand (Figure 3.4b).

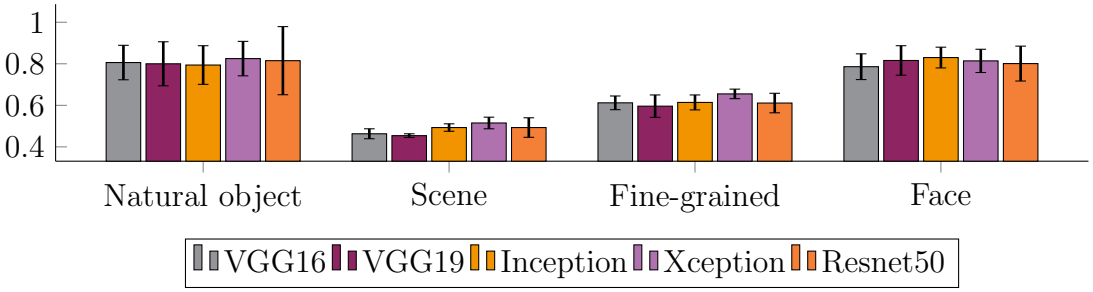
Besides the fact that, in our experiments, Xception presents slightly better results than the other architectures (higher mean and lower std), it is difficult to come up with relevant comments about these histograms. The results for each subtask contain too much variability, which prevents any kind of conclusion like: "for task T, use architecture A". However, we underline that an absence of strong pattern does not mean that any choice is equivalent. Indeed, there can be a huge difference in the results between a good and a bad architecture choice (Figure 3.5). Such absence of trend, together with the criticality of this choice, motivates the development of a new IC algorithm in Chapter 4, which leverages ensemble methods to remove the need for architecture selection.

3.3.3 Influence of the clustering algorithm

For completeness, correlations between the architecture used and the type of clustering algorithm are also checked. This analysis is intended to investigate if the features extracted by different architecture are better suited for partitioning or graph-based

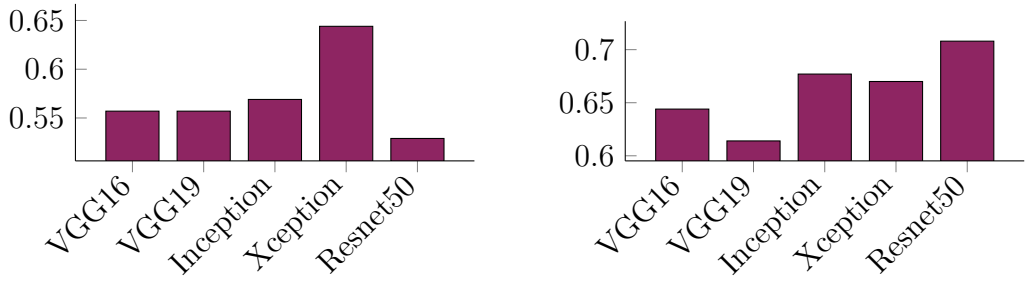


(a) mean and std across tasks and clustering algorithms



(b) Architecture-task interaction (mean and std across datasets and clustering algorithms)

Figure 3.4: Influence of the CNN architectures (L3) on the clustering results (NMI).



(a) Birds - Agglomerative clustering **(b)** Flowers - Agglomerative clustering

Figure 3.5: Examples where the choice of the CNN architecture is crucial for the clustering results.

methods. These results are summarized in Figure 3.6. Our experiments do not suggest any clear conclusions, indeed, the difference in the mean results are way smaller than the standard deviations. In other words, the risk of having poor clustering results is higher than the relative advantage one algorithm can have over the other.

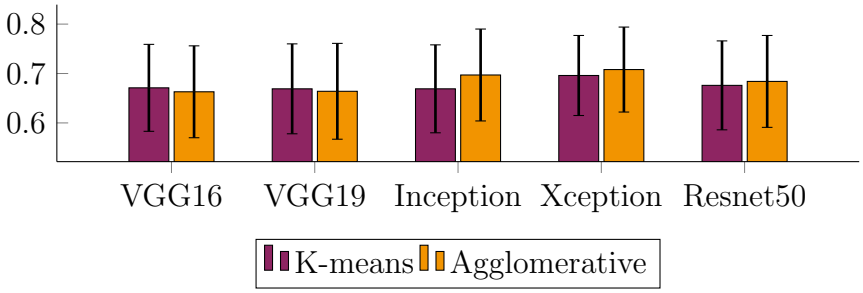


Figure 3.6: Influence of the clustering algorithm for different CNNs.

3.4 Conclusion

3.4.1 Key findings

Using pretrained CNN architecture to extract features is now a common practice to solve image clustering. However, the choice of the architecture and layer for feature extraction is often arbitrary. In this chapter, we conducted extended experiments on 8 standard computer vision datasets from four different IC subtasks to investigate the behavior of these features. Our first key finding is that for all architectures and tasks, the last layer before softmax seems to produce the most discriminative features for clustering. Our experiments also demonstrate that the selected deep feature extractor has a major impact on the results, however, they do not give any insight about how to select it.

3.4.2 Towards ensemble methods for feature extraction

The initial purpose of this benchmark study was to lead to a set of simple rules to help designing better IC algorithms by optimizing feature extraction. For example, we expected that for fine-grained clustering, earlier layers in the network might perform better as they are supposed to contain lower level information. We also expected to find some clear correlations between the tasks to solve and the architecture to use. However, the experiments carried out in this chapter rather demonstrate that such simple rules do not exist. In addition, when facing an IC problem, the feature extractor selection process cannot be cross-validated because of the absence of labels (unsupervised task). The importance of architecture selection, together with the absence of method to solve this problem, motivates the introduction of a new ensemble method to generate state-of-the-art results at IC in Chapter 4.

Chapter 4

Improving Image Clustering using Multiple Pretrained CNN Feature Extractors

Abstract

To improve image clustering results, the current practice consists in replacing raw image data with features extracted by a pretrained convolutional neural network (CNN). However, in the previous chapter, we have shown that the specific features extracted, and, by extension, the selected CNN architecture, can have a major impact on the clustering results. In addition, we have seen that this crucial design choice is hard and is often decided arbitrarily due to the impossibility of using cross-validation with unsupervised learning problems. However, information contained in the different CNN architectures may be complementary, even when pretrained on the same data. In this chapter, we propose to improve clustering performance, by rephrasing the image clustering problem as a multi-view clustering (MVC) problem that considers multiple different pretrained feature extractors as different "views" of the same data. We then propose a multi-input neural network architecture that is trained end-to-end to solve the MVC problem effectively. Our experimental results, conducted on three different natural image datasets, show that: 1. using multiple pretrained CNNs jointly as feature extractors improves image clustering; 2. using an end-to-end approach improves MVC; and 3. combining both produces state-of-the-art results for the problem of image clustering.

4.1 Introduction

4.1.1 Overview

Image Clustering (IC) is a major research topic in machine learning which attempts to partition unlabeled images based on their content. In this chapter, we still consider the IC setting where the number of clusters is a user defined parameter. As explained earlier, research in IC has recently shifted towards using features extracted from Convolutional Neural Networks (CNN) pretrained on ImageNet, which has lead to substantial improvements on complex IC datasets. In the previous chapter, it was shown that choosing a proper architecture is a very hard task when designing an image clustering pipeline. This difficulty, together with the impossibility to cross-validate algorithmic choices for unsupervised tasks, lead to pretrained feature extractor choices which are often arbitrary (([Liu et al., 2016]), ([Wang et al., 2017]), ([Gong et al., 2015]), ([Hu et al., 2017])). This is potentially problematic as we also showed in Chapter 3 that the choice of the architecture used for feature extraction has a major impact on the clustering results.

In this chapter, we aim to remove the need for this design choice. Following the intuition that different pretrained deep networks may contain complementary information (see Section 4.2.2), we propose to use multiple pretrained networks to generate multiple feature representations. Such representations are treated as different "views" of the data, thus casting the initial IC problem into Multi-View Clustering (MVC). The success of ensemble methods for clustering ([Vega-Pons and Ruiz-Shulcloper, 2011]) suggests that such an approach can improve overall clustering results.

Finally, building on the recent success of end-to-end clustering methods ([Aljalbout et al., 2018]), we also propose to leverage JULE ([Yang et al., 2016]), a deep clustering method, to solve the MVC problem. By adapting JULE to optimize the weights of a parallel neural network architecture we demonstrate state-of-the-art IC results on several public image datasets. This approach to MVC also has the advantage of producing a unified representation of the initial dataset which is low-dimensional and compact. An overview of the proposed method to solve IC can be seen in Figure 4.1.

4.1.2 Contributions

We propose to transform the IC problem into MVC by extracting features from several different pretrained CNNs. This removes the crucial design choice of feature extractor selection. We also propose to adapt a deep clustering approach to address MVC. Our

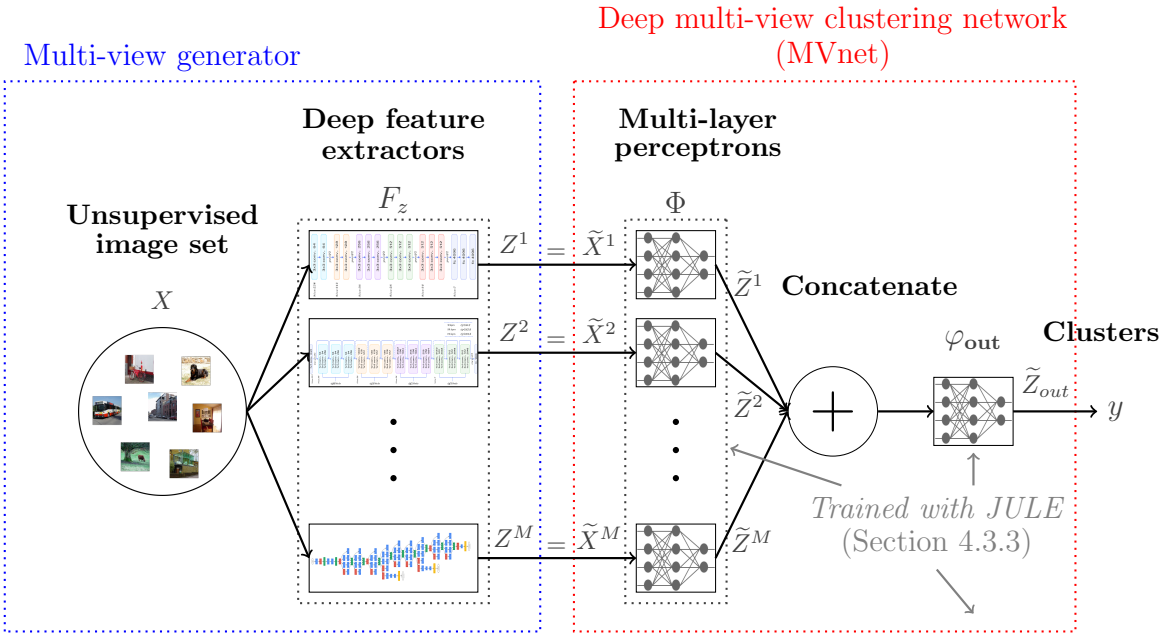


Figure 4.1: Proposed multi-view generation + deep multi-view clustering (DMVC) approach to solve the Image Clustering problem.

experimental results, carried out across 8 standard computer vision datasets, suggest that:

- *Image clustering can be improved by using features extracted from several pre-trained CNN architectures, eliminating the need to select one.*
- *Multi-view clustering can be improved by adopting an end-to-end clustering approach.*
- *These two ideas can be combined to obtain state-of-the-art results at image clustering on several standard IC datasets.*
- *This methodology produces a unified compact low-dimensional representation of the original dataset.*

4.2 From Image Clustering to Multi-View Clustering

4.2.1 Related work

Ensemble clustering (EC) consists in combining different clustering results in order to obtain a unified, final partition of the original data with improved clustering

accuracy ([Vega-Pons and Ruiz-Shulcloper, 2011]). It is composed of two steps: generation, which deals with the creation of a set of partitions, and consensus, where all the partitions are integrated into a better set of clusters. In contrast to EC, Multi-View Clustering (MVC), is concerned with finding a unified partition from multi-view data ([Chao et al., 2017]), which can be obtained by various sensors or represented with different descriptors. Recently, MVC has received a lot of attention: In ([Kumar and Daumé, 2011]), the authors propose different loss functions applied on the concatenated views, in ([Zhao et al., 2017]) and ([Wang et al., 2016]) lower dimensional subspaces are learned before clustering with standard methods.

MVC and EC are closely related and have already been combined in previous work. In ([Tao et al., 2017]), good MVC results are attained by embedding MVC within the EC framework. The authors leveraged the different views to generate independent partitions and then used a co-association-based method to obtain the consensus. In both ([Gao et al., 2015]) and ([Ceci et al., 2015]), generation mechanisms borrowed from EC are used to generate artificial multiple views of the data. In this chapter, we propose to use multiple pretrained CNNs to generate different feature representations of an image dataset. Hence, we generate a MVC problem from an ensemble of pretrained CNN feature extractors.

4.2.2 Why combining CNN architectures may succeed?

Combining CNN architectures that were pretrained on the same dataset might seem counter-intuitive as one can expect that all networks have learned the same information. This section aims at explaining the intuition behind trying this idea. This intuition is then validated experimentally in the rest of this chapter.

Let $\mathcal{I} = \{0, ...255\}^{\nu_1 \times \nu_2 \times 3}$ be the space of ν_1 by ν_2 colored images considered for IC. Then, a classification task T=(L, f^*) is defined by:

- A set of possible labels $\mathrm{L} = \{0, 1, ...K\}$, where 0 represents "none of the defined labels".
- An oracle labelling function $f^* : \mathcal{I} \rightarrow \mathrm{L}$, which associates a label to every image.

For example, for the task of classifying images of cats and dogs, L would be $\{0, 1, 2\}$, and for an image x, the oracle $f^*(x)$ would output 1 if there is a cat on the image, 2 if there is a dog and 0 if there is either none or both. This definition of a classification task is valid for supervised classification or unsupervised classification with known number of classes, which is the case studied in this chapter. Although in practice

we only study datasets composed of images which possess at least one of the labels, adding the "zero" label, allows us to define T on all $\mathcal{I}$.

This definition of a classification task is abstract and f^* is unknown and exists independently of any dataset. In practice, to solve T, one first need to materialize it in the form of a dataset DS $= (X, y^*)$, where $X \subset \mathcal{I}$ is a set of images and $y^* = f^*(X)$ are the corresponding labels in L, which are inferred by human experts. For certain problems, such as medical image annotation, human experts might be scarce, thus making labelling very costly. The classification problem (T, DS) is supervised if y^* is known and unsupervised else. In the supervised setting, solving T for DS means finding a function $f : \mathcal{I} \to \mathrm{L}$ for which there exists a domain on which it is equal to f^*: $\mathcal{D}_f = \{x \in \mathcal{I} \mid f(x) = f^*(x)\}$. We call $\mathcal{D}_f$ the domain of validity of f. Then, we have:

- $\mathcal{D}_f \subset X \Rightarrow f$ does not fit the training set,
- $\mathcal{D}_f = X \Rightarrow f$ overfits the dataset DS,
- $\mathcal{D}_f \supset X \Rightarrow f$ generalizes to some extent.

For many image classification problems, it is very hard to learn f from scratch. Instead, it is more common to use a pretrained CNN feature extractor f_z to project the initial dataset X to a latent feature space of lower dimension d:

$$\begin{array}{rccc} f_z : & \mathcal{I} & \to & \mathbb{R}^d \\ & X & \mapsto & Z. \end{array} \tag{4.1}$$

From now on, X will denote a clustering dataset. Now, let A be a clustering algorithm. Unlike in the supervised case, a clustering algorithm solves an unsupervised classification problem by looking at the whole set at once. In other words, if X contains N data points and A is applied to the outputs of f_z, we have:

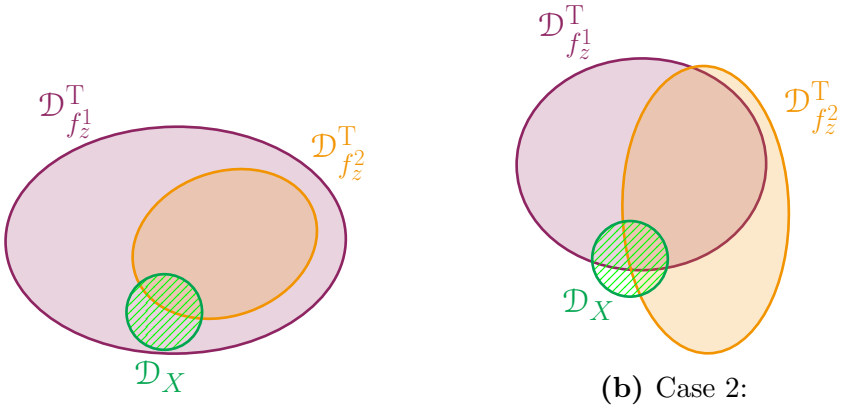
$$\begin{array}{rccc} A : & (\mathbb{R}^d)^N & \to & L^N \\ & Z & \mapsto & y, \end{array} \tag{4.2}$$

where y are the cluster assignments produced by A. From these definitions, we can introduce $\mathcal{D}^{\mathrm{T}}_{f_z,A} \subset \mathcal{I}$, the domain on which applying classification algorithm A to the outputs of f_z enables to solve task T. In the previous chapter, our experimental results suggested that, for a given dataset, the ranking of the different feature extractors is little dependent on the chosen clustering algorithm. Hence, the subscript A is dropped to define $\mathcal{D}^{\mathrm{T}}_{f_z}$, the domain on which f_z produces a "clustering friendly" latent space for task T.

Finally, let f_z^1 and f_z^2 be two pretrained CNN feature extractors, and let T be the task that we aim to solve on the clustering dataset X. In the previous chapter, it was shown that f_z^1 and f_z^2 perform differently on the different datasets. For (T, X), let's assume that, according to some clustering validation metric (e.g. NMI), f_z^1 outperforms f_z^2. Then, we can conclude that either

- $\mathcal{D}^{\mathrm{T}}_{f_z^2} \subsetneq \mathcal{D}^{\mathrm{T}}_{f_z^1}$ (Figure 4.2a) or,
- $\forall j \in \{1, 2\}, \mathcal{D}^{\mathrm{T}}_{f_z^j} \subsetneq (\mathcal{D}^{\mathrm{T}}_{f_z^1} \cup \mathcal{D}^{\mathrm{T}}_{f_z^2})$ (Figure 4.2b),

where the $\subsetneq$ symbol represents strict inclusions.



(a) Case 1: $\mathcal{D}^{\mathrm{T}}_{f_z^2} \subsetneq \mathcal{D}^{\mathrm{T}}_{f_z^1}$

(b) Case 2: $\forall j \in \{1, 2\}, \mathcal{D}^{\mathrm{T}}_{f_z^j} \subsetneq (\mathcal{D}^{\mathrm{T}}_{f_z^1} \cup \mathcal{D}^{\mathrm{T}}_{f_z^2})$

Figure 4.2: Schematic representation of the two possible implications of f_z^1 being better than f_z^2.

In both situation, leveraging both networks can have positive implications on the clustering results:

- **Case 1:** f_z^1 alone contains all the information to cluster X. However, in Chapter 3, we showed that for unsupervised datasets, it is not possible to know which network performs best. Hence, using the two networks allows to make sure that all the information available for clustering are provided to the final clustering algorithm.
- **Case 2:** The combination of the two networks contains more information than each of the networks separately. In other words, even if f_z^2 performs worse than f_z^1, it still contains information that f_z^1 does not. This kind of situation defines a typical setting where ensemble learning would be beneficial.

The domain of X, called $\mathcal{D}_X$, is represented in green on both sketches of Figure 4.2 and illustrates the two potential benefits of combining CNN feature extractors. Obviously, this intuition can be generalized to more than two networks. Although all this development is just an intuition and does not have theoretical evidences, we intend to validate it experimentally in the rest of the chapter.

The potential improvement from using multiple pretrained CNN feature extractors can also be understood through the following contrived example. To recognize a car, one network might learn a wheel detector while another one might detect wing mirrors. Both sets of discriminative features would enable to solve the ImageNet classification task, on which both networks were trained, but would also carry very different information that might be useful in solving a new IC task.

Finally, it is also important to note that the opposite intuition may also be valid. Indeed, introducing redundancy of information might hide the important information and decrease the clustering results. This will be investigated in the rest of the chapter.

4.2.2.1 Visualisation

To visualize this intuition on real data, we leverage the Fowlkes-Mallows Index (FMI) ([Fowlkes and Mallows, 1983]), another external clustering validation metric, which has the advantage of having a local form. For a dataset (X, y^*) and cluster assignments y, which associates a predicted label y_i to every point x_i in X, we note FM_i the local Fowlkes-Mallows score of datapoint x_i. FM_i ranges between 0 and 1 and is high if x_i is well clustered with respect to y^*. The FMI is better explained in Chapter 6 (Section 6.4.1). From this definition of FM_i, we introduce the concept of *FM score per class*:

$$FM_{C_k} = \frac{1}{N_{C_k}} \sum_{p=1}^{N_{C_k}} FM_p, \tag{4.3}$$

where C_k represents true class k and N_{C_k} is the number of elements of C_k in the dataset.

Then, we demonstrate the complementarity of deep feature extractors by carrying out experiments on the UMist dataset (see Section 3.2.1). We apply agglomerative clustering to the best performing network on this dataset (InceptionResnet) as well as the two worst performing networks (VGG16 and Densenet121). These networks are introduced in Section 4.2.4. Then, the NMI, purity, FM and FM_{C_4} scores are computed. As shown in the table in Figure 4.3, InceptionResnet is performing way better than its two competitors with respect to all global metrics. However, looking at class 4, we can see that the two other networks present a significant improvement

	NMI	PUR	FM	FM_{C_4}
InceptionResnet	**0.775**	**0.642**	**0.537**	0.442
VGG16	0.689	0.550	0.372	**0.653**
Densenet121	0.684	0.553	0.384	**0.700**

(a) InceptionResnet **(b)** VGG16 **(c)** Densenet121

Figure 4.3: 2d t-SNE visualization of features extracted by three pretrained CNNs for the UMist dataset. These features form different *complementary* views of the data.

over InceptionResnet. The 2d t-SNE ([Maaten and Hinton, 2008]) representations of the features extracted with the different CNNs are also represented in Figure 4.3. Members of class 4 are in orange and the other classes in purple. For VGG16 and Densenet121, the feature representations of class 4 are more compact and isolated, which explains why they perform better on this class. This experiment demonstrates the complementarity of the different networks on one example and justifies the proposed multiview clustering approach.

4.2.3 IC problem reformulation

Let $X = \{x_1, ..., x_N\} \subset \mathcal{I}$ be an unlabeled set of N natural images, and let $F_z = \{f_z^1, ...f_z^M\}$ be a set of M feature extractors. In theory, F_z can be composed of any function mapping raw pixel representations to lower-dimensional vectors, but in practice, we use pretrained deep CNNs. The first step in our approach is to generate a set of feature vectors from each element of F_z. For all $i \in [1, ...M]$, we denote Z^j the matrix of features representing X such that, its row Z_i^j is the feature vector representing x_i and extracted by f_z^j:

$$Z_i^j = f_z^j(x_i). \tag{4.4}$$

In other words, $Z = \{Z^1, ..., Z^M\}$ can be interpreted as a set of views representing X. Thus, Z is a multiview dataset representing X and the problem of clustering Z is a MVC problem, which can be solved using any MVC algorithm ([Chao et al., 2017]).

A visual representation of the multiview generation mechanism can be seen in the blue frame of Figure 4.1, which is repeated in Figure 4.4 to ease readability.

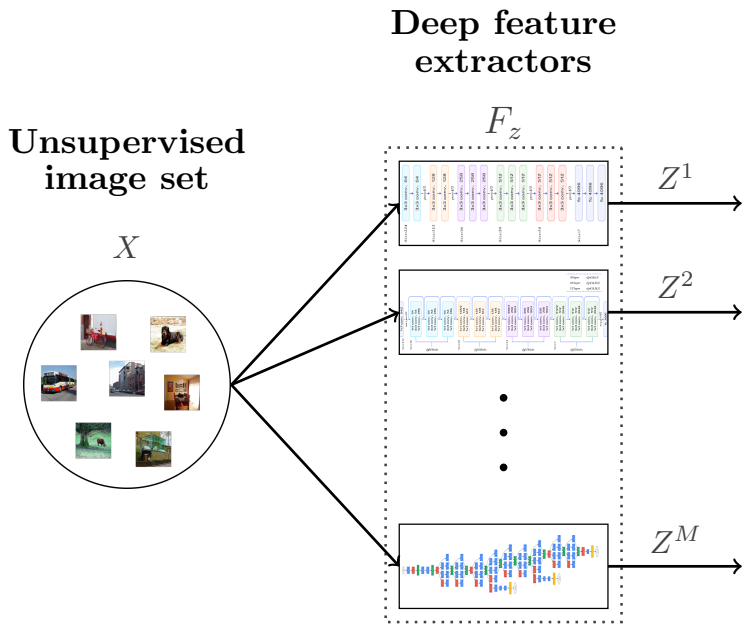


Figure 4.4: Proposed approach to generate multiple views from a unique unsupervised set images.

Remark: All along this chapter, we use letter i for indexing across data samples, letter j for indexing across feature extractors and letter k for indexing across clusters. Similarly, N, M and K respectively stand for the number of data samples, feature extractors and clusters.

4.2.4 Experimental evidences

To conclude this section on artificial multiview data generation, we give first experimental evidences and study the optimal number of networks to use. To do so, experiments are carried out on 4 of the standard datasets introduced in Chapter 3: VOC2007, Archi, Flowers and UMist. In these experiments, as well as in the rest of the chapter, we use 10 pretrained architectures: VGG16, VGG19, Inception, Xception, Resnet50, Densenet121, Densenet169, Densenet201 ([Huang et al., 2017]), Nasnet ([Zoph et al., 2017]) and InceptionResnet ([Szegedy et al., 2017]). In this section, MVC problems are solved using the Multi-View Ensemble Clustering method ([Tao et al., 2017]) with agglomerative clustering (MVEC_{agg}). This method consists in clustering each view separately using agglomerative clustering and generating a consensus partition using an ensemble clustering method based on co-association matrix.

This method is better detailed in Chapter 5. We choose agglomerative clustering as our base algorithm because its simplicity enables us to study straightforwardly the clusterability of the different feature spaces. Agglomerative clustering is also preferred over K-means because it does not depend on random initialization and thus avoids having random effects in our results.

Then, to study the clustering quality of multiview data from m (≤ 10) CNNs, we generate all the multiview problems from all the possible combinations among the different architectures. Each of these problems is then solved with MVEC_{agg} and the NMI scores are computed. In Figure 4.5, for all $m \in \{1, ..., 10\}$ we report the mean and standard deviation across the C_m^{10} NMI scores. To generate Figure 4.5, for each dataset, we need to solve $\sum_{m=1}^{10} C_m^{10} = 1023$ MVC problems. These results suggest that combining more networks both increases the clustering accuracy on average and decreases the variability, which can be seen as the risk to obtain poor results. These results are in line with the intuitions from Section 4.2.2. From these simple experiments, we decide to use all ten networks in the rest of the chapter, which is likely to give the most robust clustering results. It is also interesting to note that the two networks case performs worst than the single network case for all four datasets. This probably comes from the absence of a "majority" to distinguish which information is relevant.

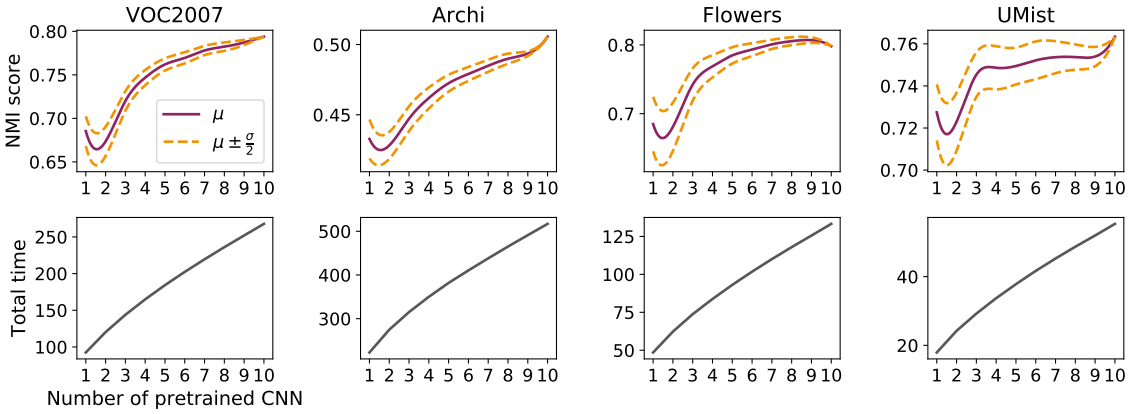


Figure 4.5: Evolution of the NMI score and total time (in sec) for different numbers of pretrained CNN feature extractors.

The evolution of the clustering time with the number of feature extractors is also reported in Figure 4.5. Obviously, when more networks are added to the pipeline, the total clustering time increases. However, when running MVEC_{agg} with m networks,

if we have m GPUs available, each feature extraction and agglomerative clustering can be run in parallel on a dedicated GPU. Thus, for a given dataset, if we note

- t_1^j the time for feature extraction with f_z^j,
- t_2^j the time for running agglomerative clustering on Z^j and
- t_3 the time for merging the different partitions into a consensus partition,

the total clustering time is $\max_j (t_1^j + t_2^j) + t_3$. For this reason, the time for running MVEC_{agg} does not increase linearly with the number of feature extractors. Using 10 networks instead of one only increases the clustering time by a factor of around 2.5, As illustrated on Figure 4.5. The increase in the total time comes from the fact that the slowest networks are selected more often when more networks are used and that the partitions grouping becomes slower. In practice the slowest step is feature extraction. If only $m\prime(< m)$ GPUs are available, then the total time increases by a factor of $\sim \lceil \frac{m}{m\prime} \rceil$, where $\lceil . \rceil$ denotes the ceiling function. Making the method scalable to a large number of CNNs when few GPUs are available is a potential area of improvement for the method.

4.3 Deep Multi-View Clustering

4.3.1 Preliminaries: Deep end-to-end clustering

In this chapter, we also propose to leverage recent end-to-end deep clustering methods to solve the MVC problem. End-to-end clustering methods based on neural networks have produced excellent results over the past two years. A complete literature review of the topic is outside the scope of this paper, for that we refer the reader to the following recent survey ([Aljalbout et al., 2018]). However, to better understand how deep clustering works we describe the first successful method, Deep Embedding for Clustering analysis (DEC) ([Xie et al., 2016]), which is a centroid-based clustering method. This approach begins by pretraining a multi-layer perceptron (MLP) using an auto-encoder input reconstruction loss function. Then, the MLP is fine-tuned to output a set of cluster centers which define cluster assignments. The joint optimization of the network and the centroids are based on the minimization of the Kullback-Leibler (KL) divergence between the current distribution of the features and an auxiliary target distribution, derived from high-confidence predictions. Improved Deep Embedding Clustering (IDEC) is an improved version of DEC introduced in ([Guo et al., 2017]). It modifies DEC by replacing the loss function by a combined

loss, taking into account the auto-encoder reconstruction loss during fine tuning. Such an approach preserves the local structure of the data and appears to improve the DEC clustering results.

4.3.2 Deep multi-view clustering (DMVC)

In this section, we define our approach for solving MVC. Let $\mathcal{C}_{\text{ee}}$ be any deep end-to-end clustering framework. $\mathcal{C}_{\text{ee}}$ is defined by a loss function $\mathcal{L}$ and a procedure $\mathcal{P}$ to optimize the loss function. Multiple approaches have already been adopted to define the clustering-oriented loss $\mathcal{L}$ and the optimization procedure $\mathcal{P}$. Two examples, IDEC and JULE, are discussed respectively in Section 4.3.1 and 4.3.3.1. To apply $\mathcal{C}_{\text{ee}}$ to an unsupervised dataset $\widetilde{X}$, one first needs to specify a neural network architecture φ_θ, parameterized by θ, which projects $\widetilde{X}$ into a lower dimensional feature space: $\widetilde{Z}_\theta = \varphi_\theta(\widetilde{X})$. Then, $\mathcal{P}$ is applied to minimize $\mathcal{L}(\theta, \widetilde{X})$, producing both a good representation $\widetilde{Z}_{\theta_{\text{final}}}$ and a set of cluster assignments y_{final}.
Remark: The proposed solution to Image clustering (IC) is in two steps: feature extraction and deep clustering. The "tilde" notation is introduced to avoid confusion between the different input spaces and latent spaces. Hence, X and Z refer respectively to the input and latent spaces of the deep feature extractor f_z, which weights are fixed, whereas $\widetilde{X}$ and $\widetilde{Z}$ refer to the deep clustering network φ_θ, parameterized by θ. Obviously, if features from a single feature extractor are used to train a single deep clustering network, we have $Z = \widetilde{X}$.

The choice of the architecture of φ_θ usually depends on the kind of dataset to solve. For example, when dealing with large images, φ_θ can be a CNN and when $\widetilde{X}$ is composed of smaller vectors, φ_θ can be a multi-layer perceptron (MLP). In the case of MVC, each element of $\widetilde{X}$ is a collection of vectors. For example, the i^{th} element of $\widetilde{X}$ is written as $\widetilde{X}_i = \{Z_i^j; \forall j \in [1, ...M]\}$. For this reason, to embed MVC into a deep clustering framework, we need to define a different neural network architecture for φ_θ, which we call MVnet. MVnet consists of a set of M independent MLPs, denoted $\Phi = \{\varphi_{\theta_1}, ...\varphi_{\theta_M}\}$, such that, $\forall j \in [1, ...M]$, the dimension of the input layer of φ_{θ_j} is equal to the dimension of the output layer of the associated f_z^j. We also define $\varphi_{\theta_{out}}$, another MLP with input layer dimension equal to the sum of the dimensions of the output layers over the elements of Φ. Thus, an MVnet is composed of three layers: a parallel layer containing all the elements of Φ, followed by a concatenating layer which feeds into $\varphi_{\theta_{out}}$. A visual representation of the MVnet architecture can be seen in Figure 4.6, which is a duplicate of the red box in Figure 4.1. We note that all the elements of Φ are independent and do not share any weights.

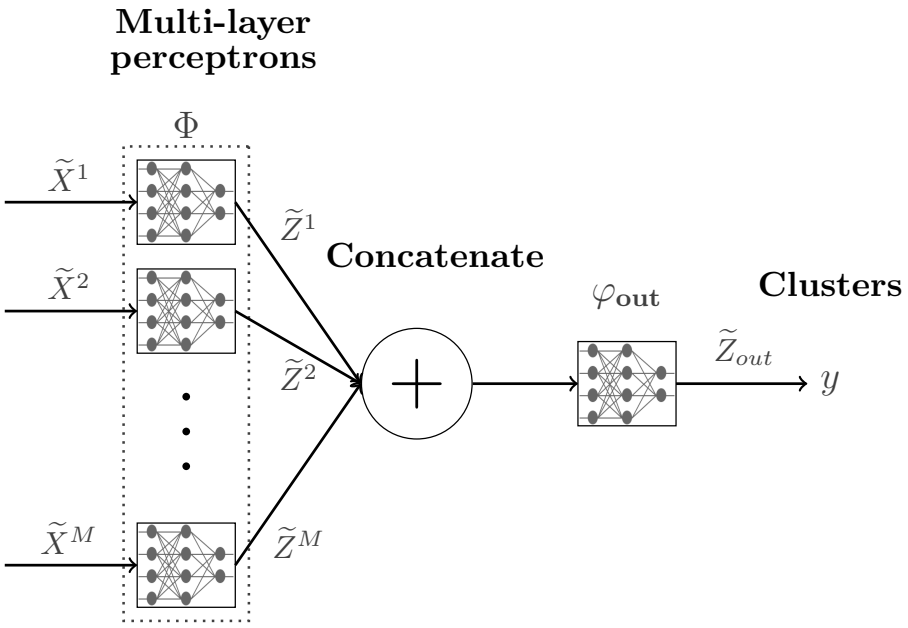


Figure 4.6: Proposed deep multi-view clustering approach to solve multi-view clustering.

DMVC is a generic framework and MVnet can be optimized using most deep end-to-end clustering approaches. In practice, we have tried to implement DMVC within both the IDEC and the JULE ([Yang et al., 2016]) frameworks. After carrying out some simple experiments on standard datasets, we noticed that JULE performs significantly better. In addition, implementing IDEC on a new dataset requires additional parameters tuning to pretrain the autoencoders, which is time consuming, potentially error-prone and less generic. For these two reasons, in the rest of this chapter, we adopt JULE to solve the DMVC problem. The proposed JULE-DMVC implementation is explained in more details in the next section.

4.3.3 DMVC with JULE

Joint Unsupervised Learning of Deep Representations and Image Clusters, or JULE, is an iterative end-to-end clustering process that has demonstrated excellent experimental results on several natural image datasets. In this section, we propose to leverage JULE to train an MVnet to solve the MVC problem. We start by explaining the standard JULE framework and then propose an extention to adapt it to MVnet. We note that Section 4.3.3.1 is a summary of JULE, for a more complete description, we refer the reader to the original paper ([Yang et al., 2016]).

4.3.3.1 Joint Unsupervised Learning of Deep Representations and Image Clusters

Notations and Definitions

Let $\widetilde{X}$ be an unsupervised dataset containing N samples that we aim to cluster into K^* groups, where the target number of clusters $K^* \in \mathbb{N}$, is a user defined parameter. Let φ_θ denote a neural network parameterized by θ, which produces a lower dimensional representation of the initial dataset $\widetilde{Z}_\theta = \varphi_\theta(\widetilde{X})$. JULE is an iterative process, hence we introduce $\theta[t]$ and $y[t]$ to talk about the values of the weights and cluster assignments at iteration t. At step t, the cluster assignment $y[t]$ defines a set of $K[t] \leq K^*$ clusters: $\{C_k[t],\ k \in \{1, ..., K[t]\}\}$. Likewise, $\theta[t]$ defines a latent space $\widetilde{Z}_{\theta[t]} = \{\widetilde{z}_i[t] = \varphi_{\theta[t]}(\widetilde{x}_i),\ \widetilde{x}_i \in \widetilde{X}\}$.

Then, we define $\mathcal{N}_i^\kappa \subset \widetilde{Z}_\theta$, the set of the κ nearest neighbors of $\widetilde{z}_i$. This definition is used to introduce the matrix W, which defines the similarity between data samples:

$$W(i_1, i_2) = \begin{cases} \exp(-\frac{||\widetilde{z}_{i_1} - \widetilde{z}_{i_2}||_2^2}{\sigma^2}), & \text{if } \widetilde{z}_{i_2} \in \mathcal{N}_{i_1}^\kappa \\ 0, & else. \end{cases} \tag{4.5}$$

In this equation, $\sigma^2 = \frac{1}{N \times \kappa} \sum_{\widetilde{z}_{i_1} \in \widetilde{Z}_\theta} \sum_{\widetilde{z}_{i_2} \in \mathcal{N}_{i_1}^\kappa} ||\widetilde{z}_{i_1} - \widetilde{z}_{i_2}||_2^2$ is used to normalize W across all the data and κ is a user defined parameter. In practice, we follow the recommendations of the authors of the original paper ([Yang et al., 2016]) and use $\kappa = 20$. The definition of similarity in Equation 4.5, which can be found in ([Zhang et al., 2012]), considers that a sample is similar to another sample only if it belongs to its neighborhood. If this condition is verified, the similarity varies inversely to the distance between these points. We note that W is not symmetric.

Finally, W is used to define the notion of clusters affinity which represents the similarity between two clusters C_{k_1} and C_{k_2}:

$$\begin{aligned} \mathcal{A}(C_{k_1}, C_{k_2}) = &\frac{1}{|C_{k_1}|^2} \mathbf{1}_{|C_{k_1}|}^T W_{C_{k_1}, C_{k_2}} W_{C_{k_2}, C_{k_1}} \mathbf{1}_{|C_{k_2}|}^T \\ &+ \frac{1}{|C_{k_2}|^2} \mathbf{1}_{|C_{k_2}|}^T W_{C_{k_2}, C_{k_1}} W_{C_{k_1}, C_{k_2}} \mathbf{1}_{|C_{k_1}|}^T. \end{aligned} \tag{4.6}$$

In Equation 4.6, $|C_k|$ is the number of samples in cluster C_k, $\mathbf{1}_{|C_k|}$ is a vector of length $|C_k|$ composed only of ones, and $W_{C_{k_1}, C_{k_2}}$ is the submatrix of W which points from samples in C_{k_1} to samples in C_{k_2}. This definition of cluster affinity also come from ([Zhang et al., 2012]).

Overview

JULE is an iterative optimization process which leverages alternating optimization to obtain both good cluster assignments $y[t_f]$ and a good new representation of the initial data $\widetilde{Z}_{\theta[t_f]}$. Indeed, going from iteration $t : (\theta[t], y[t])$ to iteration $t+1$ consists in solving two subproblems:

- Representation learning: The initial network $\varphi_{\theta[t]}$ is trained on the dataset $(\widetilde{X}, y[t])$ to generate a set of updated weights $\theta[t+1]$.
- Clusters merging: A new set of cluster assignments $y[t+1]$ are generated from similarities computed in $\widetilde{Z}_{\theta[t+1]}$.

These two steps are explained in more details in the rest of this section. Because clusters are being merged, the total number of clusters decreases when we progress through the optimization: $t < t\prime \Rightarrow K[t] < K[t\prime]$. JULE stops at iteration t_f such that $K[t_f] = K^*$. As a final step, the network $\varphi_{\theta[t_f]}$ can optionally be trained on $y[t_f]$ to fine-tune the representation. All the steps composing JULE can be visualized on a two dimensional toy example in Appendix D.

Initialization

The initial set of clusters $y[t_0]$ is computed using the initialization method proposed in ([Zhang et al., 2013]). At first, a cluster is created for each sample, containing the sample and its nearest neighbor in the input space $\widetilde{X}$, thus creating N clusters. Then, the number of clusters is reduced by merging clusters which contains duplicated samples. This heuristic process usually lead to clusters which contain between 3 to 5 samples. The weights of the neural network $\theta[t_0]$ are initialized using Xavier initialization ([Glorot and Bengio, 2010]), which consists in drawing them from a normal distribution with zero mean and a variance which is inversely proportional to the number of neurons.

Representation learning

To go from $\theta[t]$ to $\theta[t+1]$, JULE creates triplets of samples (i_a, i_p, i_n), where

- i_a is the anchor sample,
- (i_a, i_p) is a "positive pair", i.e. $y[t]_{i_a} = y[t]_{i_p}$,
- (i_a, i_n) is a "negative pair", i.e. $y[t]_{i_a} \neq y[t]_{i_n}$.

Then φ_θ, which has initial weights $\theta[t]$, is trained to minimize a triplet loss ([Schroff et al., 2015]):

$$\mathcal{L}(\theta, (i_a, i_p, i_n)) = \gamma W_t(i_a, i_p) - W_t(i_a, i_n) + \alpha, \tag{4.7}$$

where γ and α are user defined parameters. This kind of loss has become very common in representation learning ([Sermanet et al., 2018]). It consists in bringing closer points with the same labels while increasing their distance to other points. The parameter γ weights the importance between the positive and negative pairs. A large γ fosters φ_θ to produce compact clusters while a small one emphasizes well separated clusters. The parameter α is usually called the margin, it defines a minimum distance between clusters. In ([Yang et al., 2016]), the authors suggest to use $\gamma = 2$ and $\alpha = 0.2$.

Clusters merging

The cluster merging process is driven by a single parameter η, which is called the unfolding rate. To go from $y[t]$ to $y[t+1]$, we first need to compute $W[t+1]$ from Equation 4.5 in latent space $\widetilde{Z}_{\theta[t+1]}$. Then $\mathcal{A}$ is computed with Equation 4.6 and the two most similar clusters (C_a, C_b) are merged:

$$(C_a, C_b) = \underset{k_1 \neq k_2}{\operatorname{argmax}} \mathcal{A}(C_{k_1}, C_{k_2}). \tag{4.8}$$

When merging two clusters, the total number of clusters goes from $K[t]$ to $K[t] - 1$. During the "cluster merging" step of iteration t, this operation is repeated until the total number of clusters reaches $K[t+1] = \max\{\lfloor (1-\eta)K[t] \rfloor, K^*\}$. Recommendations for the unfolding rate are to use $\eta = 0.2$ for face recognition datasets and $\eta = 0.9$ for all other cases.

Remark: *After each merging, the set of clusters is modified and $\mathcal{A}$ must be recomputed. A few tricks to fasten this operation are introduced in the original paper.*

4.3.3.2 JULE with multiview data

In this section, we propose to use JULE to train an MVnet to solve the MVC problem. The initialization step for JULE requires to merge the first clusters based on distances in the initial feature space of the data. To avoid having to define a distance in a multiview space, a different approach is adopted. First, each φ_{θ_j} is pretrained separately on Z^j. After training all the φ_{θ_j}'s, $\varphi_{\theta_{out}}$ is trained on the concatenation of the $\widetilde{Z}^j = \varphi_{\theta_j}(Z^j)$. Once MVnet has been properly initialized, it is used to produce

a meaningful initial unified latent representation of the multiview data. This representation serves as the initial space in which the first cluster labels are assigned. Once the first clusters are initialized, JULE can be carried out normally on the MV data.

Another straightforward way to use JULE to solve MVC is to concatenate the different views and apply JULE to the concatenated features. This method is taken as a baseline for comparison in order to evaluate the MVnet approach and is referred to as the Concatenate and Cluster approach (CC).

4.4 Experimental validation

4.4.1 Experimental setup

Our experiments are conducted on the same 8 datasets presented in Section 3.2.1 from Chapter 3. For multiview generation, we use the Keras ([Chollet, 2015]) implementations and pretrained weights of the ten CNN architectures introduced in Section 4.2.4. For each network, the chosen layer is the last before softmax, as suggested by the experimental results of Chapter 3.

To solve the generated MVC problem, we compare the two proposed DMVC methods (CC and MVnet) against MVEC. These three methods are implemented using JULE, which has been implemented in Keras to ease integration within the full IC pipeline (Figure 4.1). Other versions of these algorithms, using IDEC instead of JULE, have been tested in ([Guérin and Boots, 2018]) but do not present a major interest. For MVnet, we also report the results without fine tuning ($\text{MVnet}_{\text{fix}}$), i.e. just after the initialization of each MLP. The results are also compared to MVEC with agglomerative clustering (MVEC_{agg}) so as to have a standard baseline for comparison.

DMVC is a framework for *unsupervised* classification, hence, hyperparameter tuning should be avoided. In all of our experiments, we use default parameters for every sub-algorithm used. For agglomerative clustering, we use the default configuration of the scikit-learn implementation ([Pedregosa et al., 2011]). For MVEC, the co-association matrix is clustered with agglomerative clustering with average linkage. Finally, for JULE, we use the hyperparameters recommended in the original paper ([Yang et al., 2016]), which are given in Section 4.3.3.1. We also use the same kind of neural network architecture used in the original paper:

- all the MLPs constituting the MVnet architecture used in our experiments have dimensions $d - 160 - 160$, where d is the input dimension,
- the activation functions for the hidden layer are rectified linear unit,

- and we also use $l2$-regularization during training.

.

The clustering results are evaluated using both normalized mutual information (NMI) and purity (PUR), which are commonly used in unsupervised classification. They both range between 0 and 1, with 1 representing perfect accuracy.

4.4.2 Experimental results

All the results of our experiments can be found in Appendix E. They report NMI and PUR scores for every pretrained CNN independently as well as for the different MVC methods applied to the MVC problems generated with the ten feature extractors. It is difficult to draw conclusions from the large number of results reported in Appendix E. For this reason, this section presents a condensed version of these results.

First of all, an algorithm is good if it produces cluster assignments with both high NMI and high purity. To simplify the analysis of the results, we introduce a mixed clustering evaluation metric to analyze jointly NMI and PUR results:

$$\text{MIX}_\beta = \beta \text{ NMI} + (1 - \beta) \text{ PUR}, \tag{4.9}$$

where β is a coefficient between 0 and 1 which weights the relative importance between NMI and PUR. Figure 4.7 represents the $\text{MIX}_{0.5}$ scores for the different methods and datasets.

To evaluate the interest of the proposed multi-view generation approach, both DMVC approches and MVEC need to be compared with each feature extractor taken independently. Comparing results with each of the ten networks is both cumbersome and difficult to analyze. Instead, we prefer to report results for the Best network (BNet) and the Worst one (WNet). BNet (respectively WNet) represents the network which demonstrates the best (respectively worst) results on the precise dataset where it appears. In practice, BNet is impossible to choose in advance because on a true unsupervised dataset, external evaluation metrics (NMI, PUR, etc.) cannot be computed. The only possible strategy for selecting a feature extractor $f_z^{j^*}$ among $F_z = \{f_z^j, j \in \{1, ..., M\}\}$ is one we call the Leading Network (LNet) strategy. Given a clustering problem X and a clustering algorithm A, the LNet strategy consists in using a set of P supervised datasets $\{(X_1, y_1^*), ..., (X_P, y_P^*)\}$ and choose $f_z^{j^*}$ such that

$$j^* = \underset{j \in \{1,...,M\}}{\text{argmax}} \left(\frac{1}{P} \sum_{p=1}^{P} \text{MIX}_{0.5}(A(f_z^j(X_p)), y_p^*) \right). \tag{4.10}$$

In other words, the leading network is the one which presents the best clustering results on average across the P supervised datasets, with respect to algorithm A. Using the online optimization formalism ([Hazan et al., 2016]), different feature extractors can be considered as experts and the obtained $\text{MIX}_{0.5}$ scores as rewards from previous trials. Then, the LNet strategy simply becomes a Follow-the-Leader strategy.

In our case, we use 8 datasets to evaluate the proposed image clustering methods. Hence, for each dataset, $f_z^{j^*}$ is computed by applying the LNet strategy on the $P = 7$ other datasets. The results obtained by $f_z^{j^*}$ on each dataset are reported under the name LNet. In practice, it appears that the optimal feature extractor $f_z^{j^*}$ is Densenet169 for all eight datasets[1].

The condensed version of the results defined above can be found in Figure 4.7. We now propose to analyze these experimental results to explain our three key findings.

4.4.3 Results interpretations

4.4.3.1 IC can benefit from the use of several CNN feature extractors

When using multiple pretrained CNNs instead of one, the ideal scenario is when the MV approach outperforms every independent network, i.e. it outperforms BNet. When this occurs, we can conclude that the different feature extractors contain complementary information which should be leveraged when possible (see Figure 4.2b and Section 4.2.2). In Figure 4.7, we can see that for all datasets but Birds and COIL100, the best of all methods is a MV method. In the specific case of COIL100, the tie between BNet and all the MV methods might mean that the domain of validity of BNet includes the domains of all the other networks (Figure 4.2a).

We remind that, in practice, it is impossible to predict which feature extractor will be the BNet. When facing an unsupervised dataset, without additional knowledge, the only way to obtain the BNet results is to choose a CNN at random and to get lucky. The risk of using random selection is to fall in the worst case scenario (WNet). This risk can be measured by the margin separating the MV methods from WNet. Likewise, the potential benefit of random selection is measured by the difference between MV and BNet results. In Figure 4.7, we see that random selection is not worth considering because the risk is much higher than the potential benefit, which most of the time is even negative.

[1] ⚠ The fact that Densenet169 is the LNet for all the datasets means that it is consistently good across datasets. It does not mean that it is the BNet for each of the datasets. Our results actually show that Densenet169 is the BNet only for Flowers.

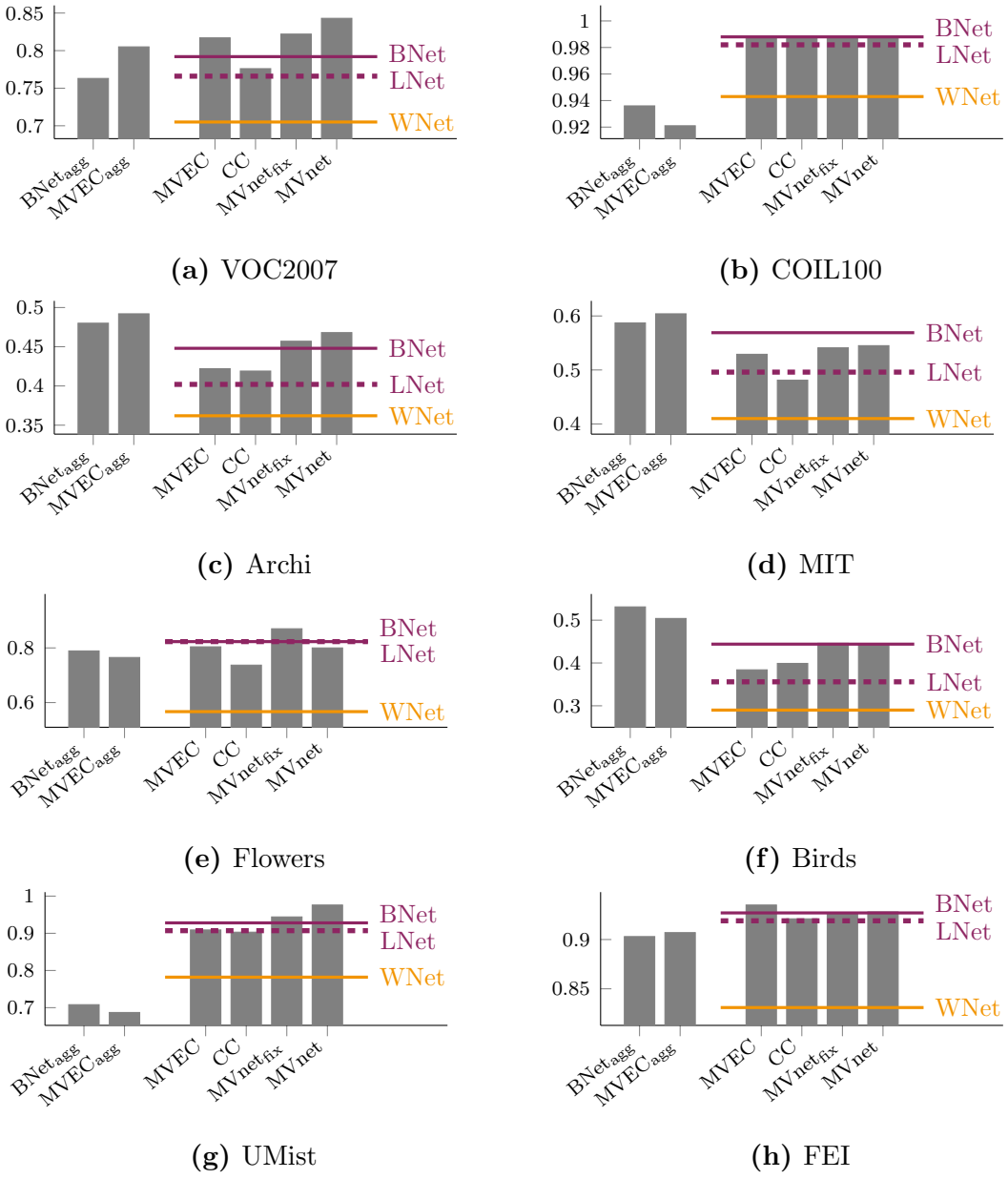


Figure 4.7: Multi-view clustering results
$MIX_{0.5}$ score values for the different MVC methods and datasets.

The second possible benefit of leveraging our MV generation approach is to improve results from LNet. To the best of our knowledge, the LNet strategy introduced above is the only feature extractor selection method which is better than random. In Figure 4.7, we can see that both MVnet and MVEC are above LNet for a large majority of cases. We also point out that for all 8 datasets, there is at least one ensemble method that outperforms LNet. One possible way to improve the LNet strategy would be to increase the number of trials, i.e. the number of dataset on

which LNet is computed. However, we doubt that the results would vary much with a larger P. These experimental results suggest that when facing an unsupervised dataset, using multiple feature extractors should be preferred over selecting a single one.

4.4.3.2 MVC can be improved by adopting an end-to-end approach

We first note that methods implemented with JULE outperform agglomerative clustering for most of the datasets. The three exceptions are the two scene recognition datasets (Archi and MIT) and Birds. One possible reason for the failure of JULE in these cases might be that the hyperparameters are not appropriate. Indeed, on the one hand, in the original paper, authors give hyperparameters recommendations for natural recognition and face recognition datasets. For these two tasks, we note that the results with JULE are very good. On the other hand, for other IC problems, such as scene and fine-grained recognition, different parameters may work better. The scalability of JULE to different kind of IC datasets would be worth investigating. We also believe that other deep clustering methods might work better on these datasets. For exemple, the results reported in ([Wang et al., 2017]) appear to be good for unsupervised scene recognition tasks. It might be worth trying to adapt their method to multiview data and thus improve results on Archi and MIT.

To evaluate the interest of using MVnet independently from other considerations, such as the problems involved by JULE on certain datasets, we now only look at the 4 bars on the right of each diagram. The first thing we note is that the MVnet architecture is better suited for multiview data than the CC approach. We also underline that in most cases, CC presents limited interest compared to LNet. These results suggests that, for a new IC dataset, data extracted from multiple CNN feature extractors should be preprocessed independently before being considered jointly. We now compare the MVnet approach against MVEC. Overall, MVnet seems to perform better and in cases where it does not, results are similar. Finally, fine-tuning $\text{MVnet}_{\text{fix}}$ end-to-end seem to be a good idea. Indeed, except for Flowers, it always seem to perform either better or similarly.

4.4.3.3 Combining multi-view generation from multiple architectures and DMVC produces state-of-the-art results at IC

We conclude this results interpretation section by stating that, to the best of our knowledge, the results reported in this thesis for VOC2007, COIL100, Flowers, UMist and FEI are the new state-of-the-art for classifying these datasets without labels.

4.4.4 Learned representations

The quality of a deep clustering algorithm can also be assessed by studying the new feature representation it generates.

4.4.4.1 Evaluation with K-means

The features extracted with MVnet are first evaluated by reclustering them using K-means. K-means is a simple clustering algorithm which performs best on representations presenting compact clusters, which are distant from each others. We choose Densenet169, which is the LNet, to represent the fixed CNN feature representation methods. Results are reported in Figure 4.8 and show that for most datasets, better features are generated as we progress in the training of MVnet. For Birds, MIT and Archi, the results remain similar, which is likely to come from the fact that JULE does not perform well on these datasets.

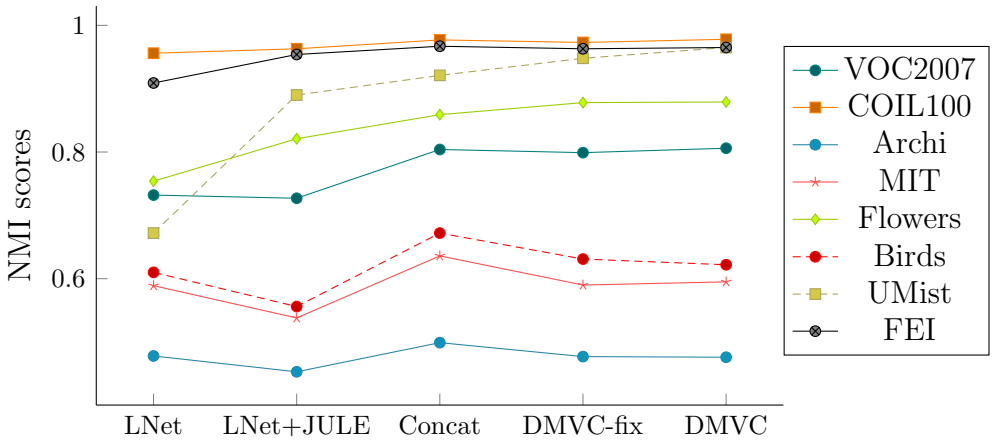
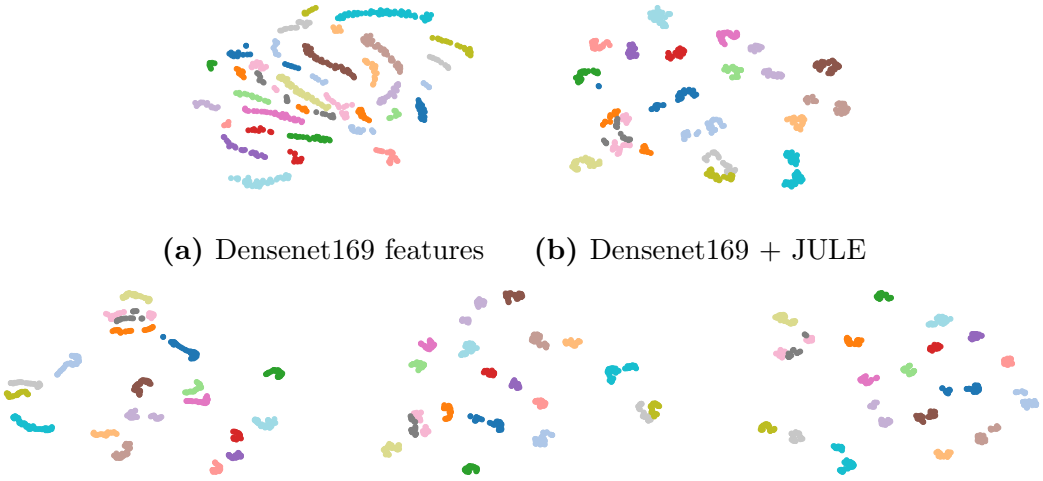


Figure 4.8: NMI scores for K-means applied to feature representations from different stages of the DMVC pipeline.

4.4.4.2 Visualization

In Figure 4.9, we also propose to visualize the evolution of the 2d t-SNE representation of features at different stages of the MVnet training for the UMist dataset. It also shows that this way of training MVnet produces representations that generate more compact clusters, which are distant from each others.



(a) Densenet169 features **(b)** Densenet169 + JULE

(c) Concat **(d)** $\text{MVnet}_{\text{fix}}$ **(e)** MVnet

Figure 4.9: 2d t-SNE visualization of the features extracted from the UMist dataset at different stages of the DMVC framework.

4.5 Conclusion

4.5.1 Key results

In this chapter, we propose a two-step approach to solving the image clustering problem. First, we generate multiple representations of each image using pretrained CNN feature extractors, and reformulate the problem as a multi-view clustering problem. Second, we define a multi-input neural network architecture, MVnet, which is used to solve MVC in an end-to-end manner. In theory, any deep clustering framework can be adapted to train an MVnet on unsupervised data. In practice we propose to implement this approach within the JULE framework and demonstrate state-of-the-art results for image clustering on several natural images datasets. This approach also has the advantage of removing the design choice of selecting a single feature extractor. Perspectives and future work regarding Image Clustering from multiple pretrained CNN feature extractors are discussed in Chapter 9.

4.5.2 Back to unsupervised robotic sorting

This part of the manuscript focuses on image clustering, which is a necessary skill for a robot to sort objects in an unsupervised way. The proposed methods for solving image clustering sets a new benchmark for this problem. However, one of the objectives of this thesis is to propose methods for unsupervised robotic sorting, which is a broader problem than image clustering. Indeed, in the IC setting studied in this part, images

are given as inputs to our algorithms whereas in a typical robotics setting, the robot needs to collect its own data, for example by moving a hand-mounted camera. In this setting, the points of view under which the objects are observed can have a significant impact on the final results. For this reason, the next part of this manuscript focuses on view selection for semantic content maximization.

Part III

Image Acquisition in Unsupervised Robotic Sorting

Chapter 5

Unsupervised Robotic Sorting from Fixed Camera Poses

Abstract

To provide a robot with the ability to sort objects based on their high level semantic nature, it is relevant to represent the objects by images. In the previous chapter, we studied Image Clustering (IC), which is a required skill for such an implementation of Unsupervised Robotic Sorting (URS). However, IC alone is not sufficient and the quality of the sorting also depends on the image acquisition process. This chapter presents an implementation of URS using fixed, vertical camera poses to gather the images, which is a common approach in robotic sorting. This methodology is first shown to work well on an industrial use case, consisting in unsupervised sorting of tools in a shopfloor environment. Then, a challenging dataset is built to further evaluate the robustness of this image acquisition approach to lighting conditions, background changes and objects poses. By using different poses of each objects jointly in a multi-view clustering framework, we show that the objects poses are a very important factor regarding the success of URS from fixed camera poses. Although the poses of the objects cannot be changed in a practical robotic sorting situation, these experiments show that the views under which the objects are observed are paramount for URS and motivate the development of a feasible optimal view selection method in Chapter 6.

5.1 Introduction

5.1.1 Real world classification system

The standard Machine Learning (ML) formulation of a classification problem is concerned with building pipelines that map numerical data to predictions. For example, in the case of image classification, the goal is to build models mapping tensors of pixels (images) to categories. According to ([Theodoridis and Koutroumbas, 2006a]), to design such a pipeline, one must solve four major subproblems: feature generation, feature selection, classifier design and system evaluation. Some classification methods perform some of these steps jointly, for example, deep learning models perform feature generation, feature selection and classifier design simultaneously. This general definition of a classification pipeline applies to both supervised and unsupervised tasks. However, when implementing a *real world classification system*, the inputs to the classification pipeline do not exist a priori and first need to be *measured from physical objects*. A schematic representation of a real world implementation of a classification system can be seen in Figure 5.1.

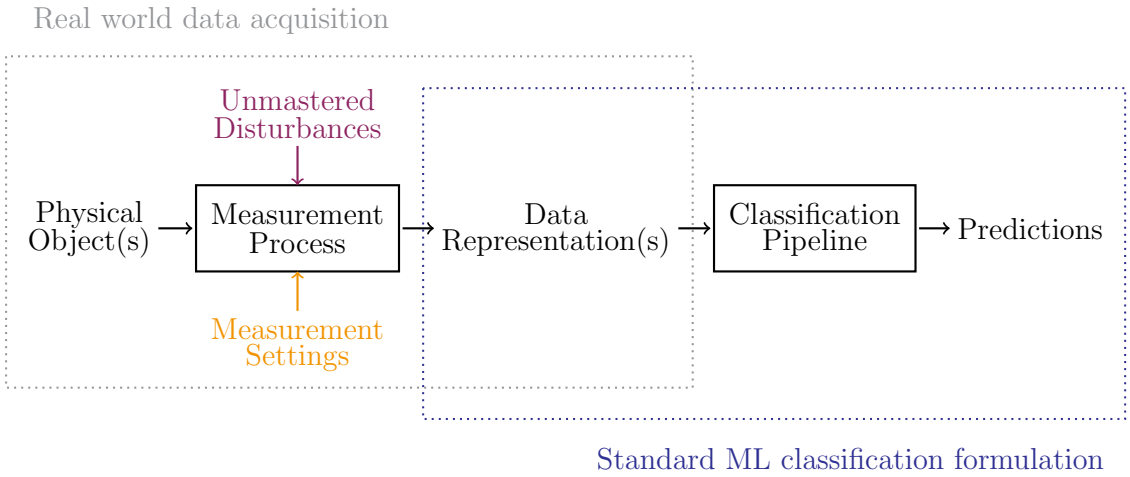


Figure 5.1: Schematic representation of a real world classification system.

To illustrate this problem, the very common toy example of a classification system to distinguish between apples and oranges can be considered. To be able to assign a category to a fruit, i.e. a physical object, one first needs to represent it by a mathematical object, i.e. extract a data representation. For example, a fruit can be represented by three real numbers: its weight, its diameter and its rugosity. Then, the

measurement process consists in weighting the fruit, as well as measuring its diameter and its rugosity. After choosing what quantities are measured, the values obtained for a given fruit can still vary from two classes of factors:

- Measurement settings: These are the design choices that are made for measuring the objects, e.g. the tools used, the protocol, etc.
- Unmastered disturbances: These are all the variations that cannot be controlled but still influence the values of the measures, e.g. non-circularity of the fruits, thermal expansion, etc.

Then, building a good real world classification system consists in choosing measurement settings and a classification pipeline that are robust to the variations of the unmastered disturbances.

5.1.2 Real world URS implementation

The Unsupervised Robotic Sorting (URS) problem studied in this thesis consists in physically sorting objects based on their semantic nature. With the recent advances in deep learning, many hard image understanding problems have now been solved. For this reason, the feature representations chosen to represent the objects to be sorted are images. Therefore, the final predictions are obtained by solving an Image Clustering (IC) problem. The important problem of IC was studied in Part II of this thesis, in which it was proposed to leverage multiple pretrained feature extractors to improve state-of-the-art results. However, to implement URS on a real robot, the data acquisition process also needs to be defined and can be of utmost importance. Indeed, depending on the scene in which the robot evolves, the objects poses, the camera poses, and the lighting conditions, the images passed as input to the IC pipeline vary a lot. A schematic representation of the decision making module of a URS implementation is proposed in Figure 5.2. The parameters in purple change among the different runs of the application and cannot be controlled, while the ones in orange need to be defined when setting up the application. Then, the objective of a URS application is to find an IC pipeline and to define rules for camera poses such that they jointly produce unsupervised object classifications which are consistently good for many kind of objects and in the range of variation of the disturbances.

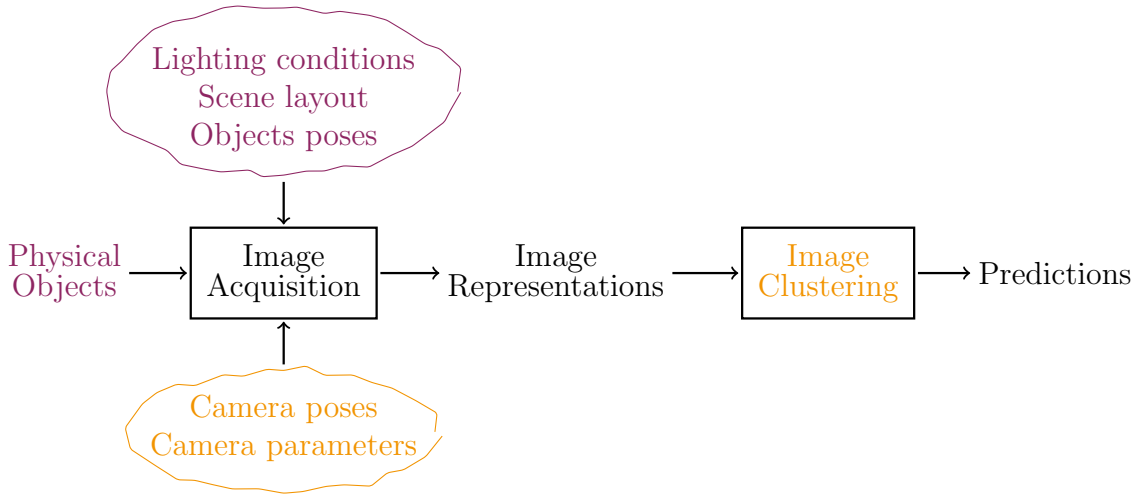


Figure 5.2: Schematic representation of a decision making module of a real world Unsupervised Robotic Sorting application.

5.1.3 Chapter organization

This chapter begins by describing a real world implementation of URS. It consists in observing objects restricted to a given region from a fixed camera pose, which is fairly common in robotics sorting (([Zhihong et al., 2017]), ([Eitel et al., 2015b])). Then, a challenging dataset is created to test the robustness to background and lighting conditions of such an approach for image acquisition. Finally, some experiments using multiple poses of each objects jointly are carried out, and appear to have a very positive impact on the robustness to unmastered disturbances. These results suggest that the view under which the objects are observed is paramount for the clustering results and motivate the development of a semantic view selection model in Chapter 6.

5.2 URS implementation with fixed camera poses

A first implementation of semantic URS was proposed for the demonstration sessions of the "TechDay Robotique Arts et Métiers[1]" in Lille, France. This demonstrator is implemented with a KUKA LBR iiwa robot with both a camera and a parallel gripper mounted on the end-effector. The environment, i.e. the workspace in which the objects are placed, is an empty wooden table that is split into disjoint areas.

[1] `https://artsetmetiers.fr/fr/techday-robotique`

Then, the URS application goes as follows:

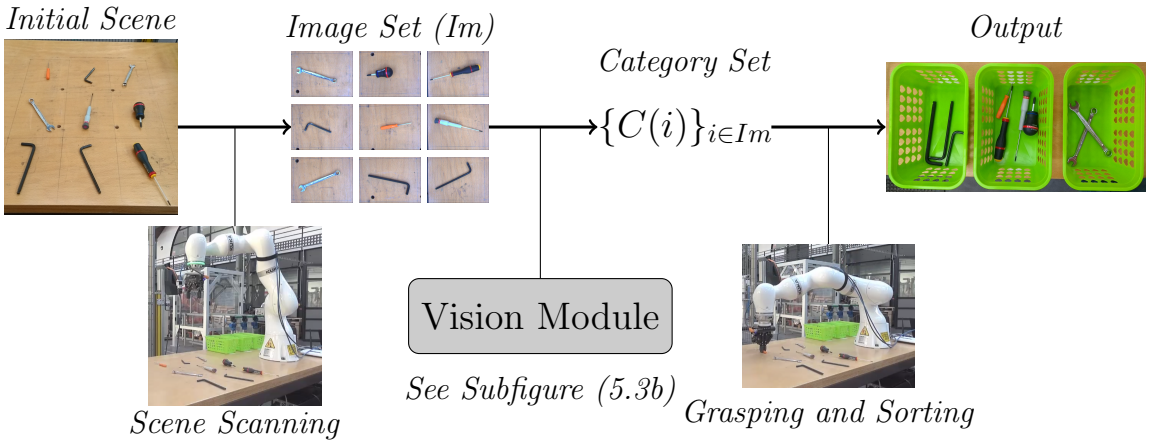
- **Setting up the scene:** Different previously unseen objects are placed in the environment such that there is no more than one object per area.
- **Scene scanning:** For each of the environment subdivisions, the robot moves the camera to the perpendicular top pose where it can see the whole area and takes a picture.
- **Clustering:** Once an image dataset of all the objects to sort has been gathered, the IC pipeline with one feature extractor presented in Chapter 3 is applied and returns a bin number for each object. The number of bins available determines the number of clusters that the IC pipeline produces.
- **Grasping and sorting:** Finally, the robot physically sorts the objects according to the clustering results.

Figure 5.3 shows a schematic view of the system, and a video of the implementation can be found at `https://youtu.be/NpZIwY3H-gE`.

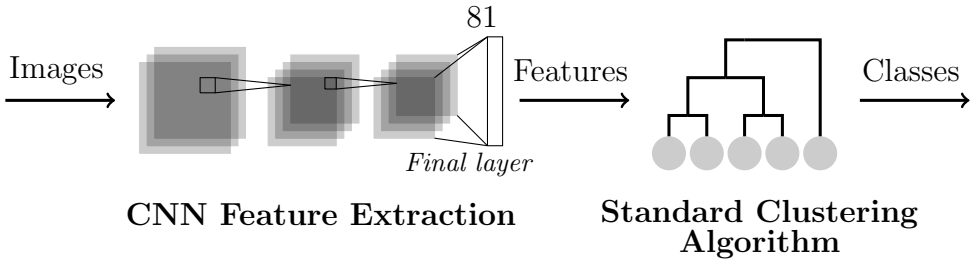
The demonstrations were carried out using sets of different tools (screw drivers, allen keys, flat keys, clamps, etc.) and the goal was to correctly group together the different models of the same tool. The demonstration sessions happened in a shopfloor with unmastered lighting conditions (glass roof), and in about 100 runs, only one misclassification was recorded, which was due to very high brightness (white image). In practice, the vision module was composed of an Xception feature extractor combined with agglomerative clustering, which is the best performing pipeline on the robustness evaluation dataset (see Section 5.3).

5.3 Robustness to background/lighting conditions

The proposed implementation, with fixed top down camera poses, demonstrates the feasibility of a URS decision making module in a real world application. In the perspective of a real application in industrial workstations, it is essential to further investigate its robustness to different environments and lighting conditions. In this section, we propose to build a dataset to evaluate such robustness.



(a) Unsupervised robotic sorting pipeline.



(b) Vision module description.

Figure 5.3: Unsupervised Robotic Sorting with fixed camera poses.

5.3.1 Dataset

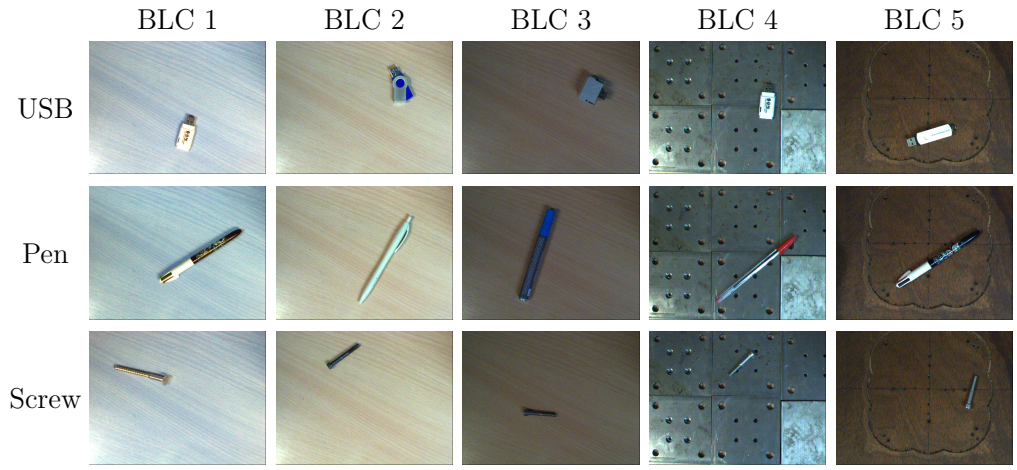
To test the robustness to unmastered disturbances of the URS application, a challenging dataset for image clustering is created, it is composed of pictures of objects which can be found in industrial workstations. All pictures constituting the dataset are taken with a fixed camera, which looks perpendicularly at a planar surface. To create the dataset, we vary both the background on which the objects are placed and the location and intensity of the light source. Such a pair defines a background/lighting condition (BLC). Objects are chosen from seven classes, and pictures of each object are taken under five different BLC. For each object and each BLC, the dataset contains four pictures under different position/orientation within the field of view of the top down camera. Using multiple poses of each object enables to reduce the influence of the object pose, and thus isolate the impact of background and lighting conditions. The different poses are also used in Section 5.4 to study how the point of view under which the objects are observed influences the sorting results. This dataset, which is illustrated in Figure 5.4, appears to be challenging for image clustering because of the

BLC variations but also because some classes have low intra-cluster similarity (e.g. USB drives) and extra-cluster similarity between some classes is relatively high (e.g. pens and screws).

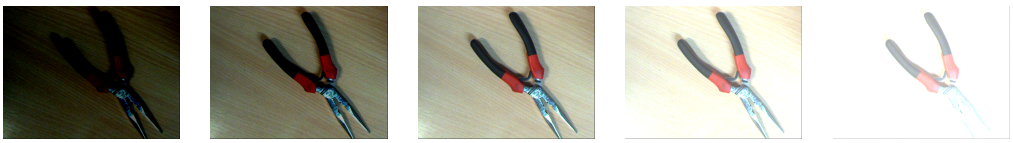
Table 5.1: Statistics of the proposed tool clustering dataset *(BLC stands for Background/Lighting Conditions)*.

#Images	Images size	#Classes	#Images per class	#BLC
560	640 x 480	7	12 to 24	5

The dataset statistics are summarized in Table 5.1 and sample images, illustrating the different objects, poses and BLC, can be seen on Figure 5.4a. The dataset, together with its description, can be downloaded at: `https://github.com/jorisguerin/toolClustering_dataset`. For further evaluation of the robustness to lighting conditions, we also modify computationally the brightness of the pictures under conditions 2 only. This allows to isolate the influence of brightness in the clustering results. Example images with the applied brightness filters can be visualized on Figure 5.4b.



(a) Different poses, backgrounds and lighting conditions.



(b) Artificial brightness modifications on BLC 2.

Figure 5.4: Example images used to evaluate robustness to unmastered disturbances.

5.3.2 Results

The objective of this section is to *evaluate the robustness of the image acquisition method to background and lighting conditions*. However, in Chapter 3, it was shown that a bad choice of CNN feature extractor and/or clustering algorithm can lead to poor clustering results for certain datasets. Hence, to isolate the influence of the image acquisition scheme as much as possible, a good clustering pipeline is needed. Therefore, the experiments are conducted on 10 clustering pipelines (5 CNN architectures and 2 clustering algorithms), and the results are used to choose the one presenting the strongest results. For a given BLC, a clustering problem is sampled by randomly taking one image (i.e. one position/orientation) for each object. The final experiments consist in sampling 1000 clustering problems for each of the BLC and cluster them with each of the pipelines. Results from these experiments are reported in both Tables 5.2 (physical BLC variation) and 5.3 (artificially modified brightness on BLC 2). For each BLC, the results reported are the means over the 1000 runs. Averaging over all the positions enables us to reduce the dependence of the results on the objects poses, and thus better study the robustness to BLC.

Table 5.2: Clustering results for different CNN architectures and clustering algorithms on the tool clustering dataset.

		BLC1		BLC2		BLC3		BLC4		BLC5	
		NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*
Inception V3	Agg	0.82	*0.81*	0.82	*0.81*	0.80	*0.80*	0.65	*0.65*	0.79	*0.76*
	KMeans	0.80	*0.79*	0.79	*0.78*	0.76	*0.76*	0.63	*0.64*	0.75	*0.73*
Resnet50	Agg	0.81	*0.81*	0.74	*0.74*	0.74	*0.75*	0.62	*0.59*	0.72	*0.71*
	KMeans	0.77	*0.78*	0.71	*0.72*	0.71	*0.72*	0.58	*0.58*	0.70	*0.70*
VGG16	Agg	0.76	*0.75*	0.74	*0.73*	0.73	*0.72*	0.61	*0.60*	0.70	*0.69*
	KMeans	0.72	*0.72*	0.70	*0.70*	0.71	*0.70*	0.58	*0.57*	0.67	*0.67*
VGG19	Agg	0.76	*0.76*	0.77	*0.76*	0.71	*0.72*	0.59	*0.58*	0.71	*0.70*
	KMeans	0.73	*0.73*	0.73	*0.73*	0.69	*0.70*	0.56	*0.56*	0.67	*0.67*
Xception	Agg	0.86	*0.85*	0.90	*0.90*	0.84	*0.85*	0.69	*0.69*	0.83	*0.81*
	KMeans	0.84	*0.83*	0.87	*0.86*	0.82	*0.82*	0.66	*0.66*	0.80	*0.80*

Table 5.3: Clustering results for different CNN architectures and clustering algorithms for different artificially modified lighting conditions on the BLC2 subset.

		Very dark		Dark		Normal		Bright		Very bright	
		NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*
Inception V3	Agg	0.74	*0.72*	0.81	*0.79*	0.82	*0.81*	0.80	*0.80*	0.71	*0.70*
	KMeans	0.70	*0.70*	0.77	*0.75*	0.79	*0.78*	0.77	*0.77*	0.66	*0.67*
Resnet50	Agg	0.67	*0.67*	0.73	*0.73*	0.74	*0.74*	0.69	*0.68*	0.61	*0.61*
	KMeans	0.65	*0.66*	0.70	*0.71*	0.71	*0.72*	0.66	*0.66*	0.58	*0.59*
VGG16	Agg	0.66	*0.66*	0.73	*0.72*	0.74	*0.73*	0.68	*0.68*	0.61	*0.61*
	KMeans	0.62	*0.63*	0.69	*0.69*	0.70	*0.70*	0.65	*0.66*	0.57	*0.58*
VGG19	Agg	0.67	*0.67*	0.76	*0.75*	0.77	*0.76*	0.74	*0.72*	0.64	*0.65*
	KMeans	0.64	*0.65*	0.73	*0.73*	0.73	*0.73*	0.71	*0.70*	0.59	*0.62*
Xception	Agg	0.77	*0.77*	0.88	*0.89*	0.90	*0.90*	0.84	*0.84*	0.73	*0.74*
	KMeans	0.74	*0.74*	0.85	*0.86*	0.87	*0.86*	0.82	*0.82*	0.70	*0.71*

From these experimental results, three important conclusions are drawn:

- In Tables 5.2 and 5.3, we can see that agglomerative clustering always present better results than KMeans. Likewise, Inception-like architectures tend to outperform their competitors, particularly Xception, which outperforms all the other CNN architectures for every clustering algorithm and every BLC. Based on these results, from now on, we only study the Xception + Agg vision pipeline.

- Table 5.3 shows that the chosen pipeline seems to be robust to reasonable changes in lighting conditions (Dark and Bright) but its performances start to decrease when the brightness is really high or low. This makes sens as it also becomes difficult for a human to identify these objects.

- Background changes seem to have a more important impact on the clustering results. Indeed, in Table 5.2, BLC 4 shows much lower results than other BLC. This might come from the fact that the background in BLC4 contains geometric shapes (lines, circles) which are distractors for the network.

5.4 Robustness to objects poses

5.4.1 Methodology

In Section 5.1, it was shown that given a set of objects to sort, the clustering results depend on the lighting conditions, the scene layout, the objects poses and the camera poses to observe each object. The dependence on background and lighting conditions was studied experimentally in Section 5.3, and we now want to evaluate the influence of the objects poses on the results of the proposed implementation. To do so, we

propose to vary the poses of the objects under a fixed camera view, which boils down to leveraging the different poses in the dataset introduced in Section 5.3.1.

In the unsupervised image classification setting, the clustering results depend on the entire set of images jointly, thus making it hard to evaluate the contribution of an individual image. For this reason, a Multi-View Clustering (MVC) approach is adopted to evaluate the influence of the object poses. Indeed, if using multiple poses jointly enables to improve the clustering results substantially, we can conclude that either some views are better than the others, or that the views are complementary (see Section 4.2.2). In both cases, this would motivate the development of better view selection methods, which is the purpose of the next chapter. The MVC problems are solved using the Multi View Ensemble Clustering (MVEC) method that was used for comparison in Chapter 4, and which is described in more depth in Section 5.4.2.

Remark: The Deep Multi-View Clustering method proposed in Chapter 4 was not used because our implementation of URS consists in sorting a small number of objects (< 50), which leads to IC problems containing too few instances. Deep end-to-end clustering methods are usually bad at solving small scale tasks because, with too few data, the desired number of clusters is reached too fast (after the initialization step) and the representation learning networks cannot converge. However, such method is not to be excluded for larger scale industrial URS problems leveraging multiple views.

5.4.2 Multi-view clustering using Ensemble-clustering

Let $\Omega = \{\Omega_1, ...\Omega_M\}$ be the set of M objects to cluster. For each object Ω_i, let $\omega_i = \{\omega_{i,1}, ...\omega_{i,m_i}\}$ be the set of m_i views representing it. The clustering pipeline chosen from the experiments in Section 5.3.2 (Xception + Agg) is denoted C. The proposed MVEC method, inspired by ([Tao et al., 2017]), goes as follows. For each object Ω_i, one image $\omega_{i,j}$ is selected randomly. Then, the new image set is clustered using C, thus producing a partition (cluster assignment for each object) denoted P_k. This procedure is repeated N times and the set of N partitions generated is noted $P = \{P_1, ..., P_N\}$.

Once the N partitions have been generated by randomly sampling images in each ω_i, MVEC must find a consensus partition P^*, maximizing the agreement between the different partitions. In this thesis, an approach based on objects co-occurrence is adopted to compute P^*. Since this work is about unsupervised classification, there is no correspondence between the class assignments of the different partitions of P. To avoid this problem, we use intermediate representation of P, called the co-association

matrix. This matrix, denoted $\mathcal{A}$, is a $M \times M$ symetric matrix which entries are defined by

$$\mathcal{A}_{pq} = \frac{1}{N} \sum_{t=1}^{N} \delta(P_t(\Omega_p), P_t(\Omega_q)), \tag{5.1}$$

where $P_t(\Omega_i)$ is the label associated with object Ω_i and generated by partition P_t, and $\delta(a, b)$ is the Kronecker symbol, which is 1 if $a = b$ and 0 otherwise. Entry (i, j) of CA measures how many times objects Ω_i and Ω_j have been classified together by the different partitions.

Finally, let C^* be any connectivity-based clustering algorithm. For instance, C^* can be variants of agglomerative clustering or spectral clustering. The consensus partition P^* is obtained by applying C^* using $\mathcal{A}$ as the precomputed similarity measure between objects:

$$P^* = C^*(\mathcal{A}). \tag{5.2}$$

A schematic representation for the MVEC pipeline can be found in Figure 5.5. In our implementation, image sampling for partition generation is uniform, C^* is agglomerative clustering and the number of partitions generated is $N = 1000$.

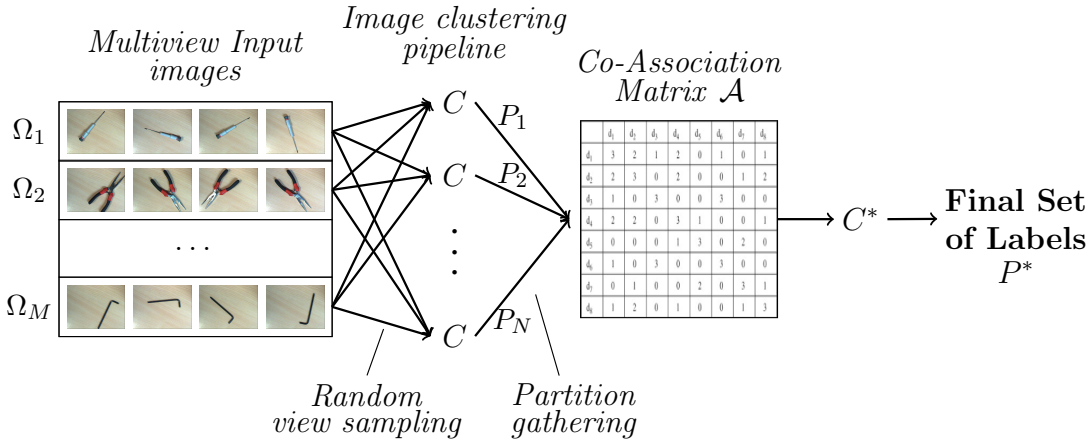


Figure 5.5: Proposed MVEC approach to use multiple views of each object.

5.4.3 Results

The results using MVEC with the clustering pipeline chosen above are reported in Table 5.4. MVEC consistently performs significantly better than the corresponding single view approach, which are written in parenthesis in Table 5.4 as a reminder. It is also more robust to poor background (BLC4) and lighting (Very bright) conditions.

Table 5.4: Clustering results of MVEC for different BLC. For comparison, the corresponding single view results are reminded in parenthesis.

		NMI	Purity
	BLC1	0.95 *(0.86)*	0.96 *(0.85)*
BLC2	Very dark	0.91 *(0.77)*	0.93 *(0.77)*
	Dark	1.00 *(0.88)*	1.00 *(0.89)*
	Normal	1.00 *(0.90)*	1.00 *(0.90)*
	Bright	0.96 *(0.84)*	0.96 *(0.84)*
	Very bright	0.84 *(0.73)*	0.86 *(0.74)*
	BLC3	0.95 *(0.84)*	0.96 *(0.85)*
	BLC4	0.84 *(0.69)*	0.82 *(0.69)*
	BLC5	0.95 *(0.83)*	0.96 *(0.81)*

Regarding computation time, although MVEC takes longer, this time can be drastically reduced by parallelizing both the partition generation process and the co-association matrix computation.

The excellent results obtained by MVEC suggest that the proposed image acquisition method that consists in using fixed top down camera views is very sensitive to objects poses. Indeed, the fact that the robustness can be increased significantly by duplicating the objects poses suggests that in the single view approach, some of the poses were very poor for clustering, thus decreasing the average results presented in Tables 5.2 and 5.3. Another possible cause of having poor results with a single view approach is that the individual images may not be compatible to use jointly. For example, it may not be easy to group together two mugs if one is seen from the top and the other from the side. Hence, by increasing the number of views, the event of having similar images for similar objects becomes more likely. Robustness to poor lighting conditions also makes sense as light comes from a certain direction and there are always better angles to observe the objects.

5.5 Conclusion

5.5.1 Key results

Unsupervised Robotic Sorting consists in physically grouping together previously unseen objects in a way that makes sense at a human level. In this chapter, a first working implementation is presented. In this application, objects are placed in constrained regions of the environment and the image representations are acquired from predefined fixed camera poses. The unsupervised image classification module is com-

posed of a pretrained Xception feature extractor and agglomerative clustering. This approach is shown to work for a practical use case, in a real-world industrial environment. To further test the robustness of this image acquisition approach to lighting conditions, background changes and objects poses, a robustness testing dataset was specifically created. This dataset, which is challenging for image clustering, is made publicly available to help other researchers to test their image clustering algorithms. On the one hand, the proposed URS pipeline appears to be fairly robust to reasonable lighting changes and to different uniform backgrounds. On the other, the results decrease drastically when the background presents strong patterns and for very bright/dark images. The influence of the poses of the objects is evaluated by carrying out multi-view clustering experiments. We show that sorting results can be improved by using jointly images from several objects poses, which suggests that under a fixed camera pose, the poses of the observed objects are of major importance. Perspectives and future work regarding practical implementations of URS are discussed in Chapter 9.

5.5.2 Towards adaptive camera poses

The first approach implemented in this chapter to solve URS demonstrates rather good results, which is very encouraging. However, it was also shown that using fixed camera poses for images collection is sensitive to objects poses and would benefit from using several of these poses for each object. This makes the fixed camera approach problematic because, in practical situations, the pose of an object in the environment is an unmastered input to URS and cannot be changed. Hence, the fixed camera approach is risky as it cannot account for all possible variations of the objects poses.

On another note, the goodness of a pose of an object is not intrinsic to the pose itself but rather relative to the camera pose. What really matters is the relative pose between the object and the camera. In addition, the camera pose is a user defined setting that can be modified. Consequently, a different image acquisition module, where the camera poses are adapted to the unmastered objects poses, could help improving URS. This statement motivates the development of an autonomous optimal view selection pipeline in Chapter 6.

Chapter 6

Semantically Meaningful View Selection

Abstract

An understanding of the nature of objects could help robots to solve both high-level abstract tasks and improve performance at lower-level concrete tasks. Although deep learning has facilitated progress in image understanding, a robot's performance in problems like object recognition often depends on the angle from which the object is observed. Traditionally, robot sorting tasks rely on fixed top-down views of the objects. By changing its viewing angle, a robot can select a more semantically informative view leading to better performance for object recognition. In this chapter, we introduce the problem of semantic view selection, which consists in finding good camera poses to gain semantic knowledge about observed objects. We propose a conceptual generic formulation of the problem, together with a relaxation based on clustering, to make it solvable. We then present a new image dataset consisting of around 10k images representing various views of 144 objects under different poses. Finally we use this dataset to propose a first solution to the problem by training a neural network to predict a custom "semantic score" from a top view image and camera pose. The views predicted to have higher scores are then showed to provide better clustering results than fixed top-down views.

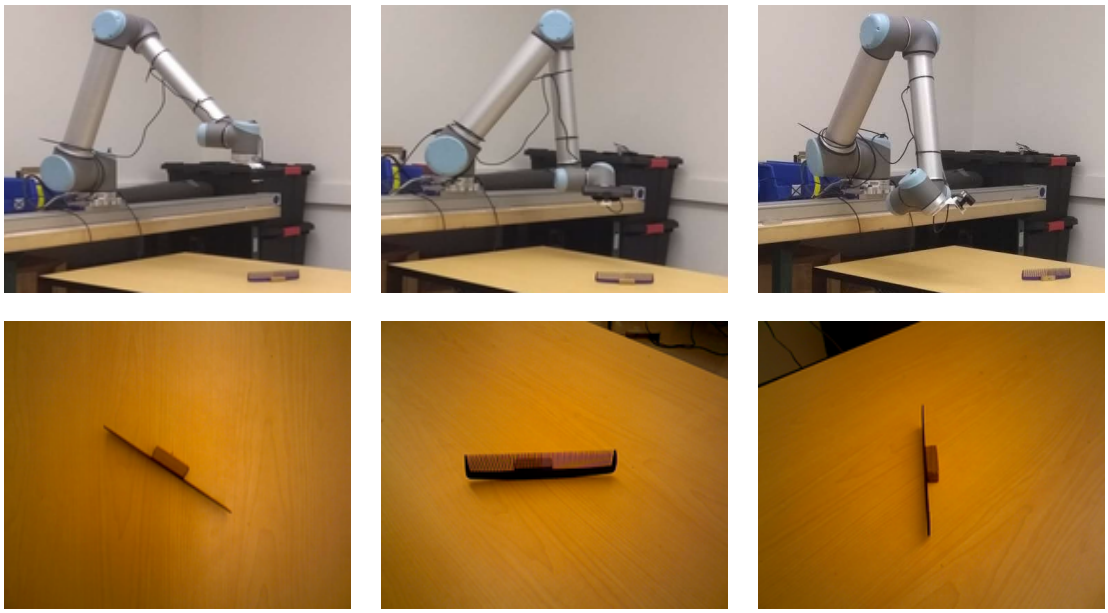
6.1 Introduction

6.1.1 From URS to Semantic View Selection

A robust Unsupervised Robotic Sorting (URS) application requires both a good strategy for image acquisition and a good Image Clustering (IC) pipeline. In the previous chapter, a first straightforward pipeline for image acquisition is presented. It consists in gathering images of the different objects from fixed, top down camera poses. Although this approach, combined with a good IC pipeline, has demonstrated good results on a real-world industrial implementation, we have also shown that URS may benefit from an adaptive view selection scheme. In this chapter, we propose to study the problem of *optimal view selection for clustering*. This problem is embedded into the broader problem of *optimal semantic view selection*. As explained in Section 6.2, we believe that choosing views which are semantically meaningful and views which are good for clustering are two problems which are highly positively correlated. In the meantime, viewing the problem as one of finding "semantic views" is more generic and has more potential direct and indirect applications. Moreover, formally defining the generic semantic view selection problem leaves the door open to different approaches for future research. Hence, this chapter deals with the problem of Semantically Meaningful View Selection (SVS) independently of any URS considerations. The link back to URS is done through the experimental results section, where we demonstrate that our approach for adapting camera poses enables to improve unsupervised sorting. The following section proposes a richer introduction about SVS, thus extending the context of URS.

6.1.2 Introduction and context for Semantic View Selection

Recent advances in machine learning have increased robot autonomy, allowing them to better understand their own state and environment, and to perform more complex tasks. An important research direction is to improve semantic understanding of objects, which can aid in tasks such as manipulation. For example, better semantic understanding can be directly used to solve tasks such as supervised (([Eitel et al., 2015b]), ([Zhihong et al., 2017])) and unsupervised (Chapter 5) sorting. It can also impact indirectly other important tasks, such as robotic grasping (([Bohg et al., 2014]), ([Lenz et al., 2015])). Indeed, ([Chua et al., 2017]) showed that the way people grasp objects not only depends on their forms and shapes, but also on our semantic understanding of the object. Knowledge of manipulated objects is also important for human robot collaboration ([Tsarouchi et al., 2017]), where significant



(a) Top view: $\theta = 90°$; $\varphi = 90°$

(b) Good view: $\theta = 45°$; $\varphi = 45°$

(c) Bad view: $\theta = 135°$; $\varphi = 45°$

Figure 6.1: Illustration of the Semantic View Selection Problem. The angular parameterization is defined in Section 6.3.

efforts have been made to make robot behavior safe and adaptive to humans ([Munzer et al., 2017]). For example, a robot should not hand a human a knife by the blade.

Vision-based methods are a natural choice to acquire knowledge about manipulated objects, as supported by the recent advances in deep learning for both supervised (([Chollet, 2016]), ([He et al., 2016])) and unsupervised (([Yang et al., 2016]), ([Aljalbout et al., 2018])) image classification. However, a robot can act in the real world to change the view under which the object is observed. This can have a huge impact on understanding what the object is. For example, in Figure 6.1, only the middle image enables the robot to understand that it is looking at a comb. The robot's ability to act has not been fully exploited in previous research. For example, prior work on robotic sorting of objects relies on a fixed, perpendicular top pose for the robot camera (see Chapter 5). Some previous work for best view selection ([Dutagaci et al., 2010]) has focused on producing representative views of 3d mesh models. Although this is a promising approach, it is not applicable for many robotics tasks, especially when complete 3d models are not available for all of the manipulated objects.

A possible reason why view selection was not studied before may be that previous work mostly deals with instance retrieval or supervised classification. In these two applications, the models to recognize the object are trained to be robust to view

changes by overfitting the problem to solve. Indeed, instance retrieval deals with recognizing a precise object from a bank of objects. In this context, the view is not crucial as each face of an object is likely to contain characteristics which are specific enough to distinguish this object from the rest of the set. In other words, an instance retrieval model is overfitting on the set of objects. Likewise, supervised learning can be viewed as a form of overfitting on the given classes. As a matter of fact, when training a supervised model, we just need to be able to separate the object from classes which are present in the supervised problem. These characteristics do not have to be very specific to the semantic class of the object. For example, detecting a wheel might be sufficient to distinguish a car from a horse, but it is not generic enough to know that we are not looking at a bus.

In this chapter, we aim to find a method to optimize the poses of a robot with a hand-mounted camera, to maximize the semantic content of the images and understand the nature of objects being observed (Figure 6.1). In Section 6.2, we first propose a generic conceptual formulation of this problem, which we call the Semantic View Selection (SVS) problem. We then relax the problem by reducing it to the optimization of a clustering-based objective. To solve this problem, we introduce a new image dataset containing 144 objects, from 29 categories, under different poses and observed under various views. Both the data collection process and the dataset content are described in Section 6.3. A first approach using the clustering SVS problem formulation on the new dataset is then detailed in Section 6.4. It consists in training a multi-input deep convolutional neural network to map a top view and proposed camera pose to a semantic score. Our experimental results, in Section 6.5, demonstrate that the proposed network can predict camera poses which outperform fixed poses at unsupervised sorting tasks.

6.2 The Semantic View Selection (SVS) problem

In this section, we formally introduce the problem of selecting optimal views for semantic understanding and introduce notations.

6.2.1 Generic formulation: the semantic function

We begin this section by introducing some definitions. A scene is defined by a set of objects and their relative poses. For example, the scene in Figure 6.2 is composed of a table, a mug, a bottle and other objects around. In this chapter, the scenes are considered static, which means that in Figure 6.2, changing the pose of the mug would

(a) View 1 **(b)** View 2

Figure 6.2: Example of a scene containing two objects under different viewpoints.

define a different scene. However, the same logic could apply to dynamic scenes by making all variables time dependant. Let ζ denote such a scene.

Now, let c be a camera, mounted at the end-effector of a robot manipulator, with which the scene is observed. We also define $\mathcal{D}_\zeta \subset SE(3)$, the domain of valid camera poses for the scene ζ, which is defined by all the poses of c that are in the reachable workspace of the robot and for which there is no collision between the robot, the camera, and the different elements composing the scene. Then, we can define the *view function* associated with ζ, which associates an image to any valid camera pose, by

$$\begin{array}{rccc} v_\zeta : & \mathcal{D}_\zeta & \rightarrow & \mathcal{I} \\ & p_c & \mapsto & I. \end{array} \tag{6.1}$$

In Equation (6.1), $\mathcal{I} = \{0, ..., 255\}^{\nu_1 \times \nu_2 \times 3}$ is the space of ν_1 by ν_2 colored images (resolution of c) and p_c denotes a camera pose expressed in a frame that is fixed with respect to ζ. A view function is defined with respect to a given scene ζ, indeed, an identical camera pose in a different scene can produce very different images.

Then, we consider a given object ω in ζ and we define the conceptual *semantic function* $S_\omega(.)$, representing the semantic information about ω contained in an image. $S_\omega(I)$ is high if I is highly informative about the nature of ω (second column of Figure 6.1) and low if it's not (third column of Figure 6.1). A natural choice for the output space of S_ω is $[0, 1]$. A semantic function is defined relatively to the object considered, which can be understood by considering the example in Figure 6.2. Indeed, for the image in Figure 6.2b, S_{bottle} should be high while S_{mug} should be low. We also note that in practice, the semantic function is defined with respect to a subjective label definition. In this thesis, we use the most generic and simple possible label definition (e.g. spoon, mug, toothbrush, etc.), without adding any

specific description (e.g. silver spoon, blue mug, etc.). More concretely, semantic meaningfulness can be viewed as the information contained in the output of a high level feature extractor, e.g. last layer of a pretrained deep CNN, which can be used to infer the general category of the object represented in the image.

For a given scene ζ and a given object ω in this scene, the Semantic View Selection (SVS) problem is defined as follows:

$$\underset{p_c \in \mathcal{D}_\zeta}{\text{Maximize}} \quad S_\omega \circ v_\zeta(p_c). \tag{6.2}$$

In other words, we aim to find p_c^* such that the view $v_\zeta(p_c)$ maximizes S_ω. The function $S_\omega \circ v_\zeta(.)$ depends on the object being study ω, its pose, the other objects in the scene that can act as distractors, and the pose of the camera.

6.2.2 First relaxation: clusterability functions

The semantic function defines the general form of the SVS problem, but in practice, it cannot be evaluated. Therefore, to approximate S, we introduce a new family of *clusterability functions*: $\{S^{a,m}, a \in \mathcal{A}, m \in \mathcal{M}\}$, where $\mathcal{A}$ represents the space of all possible image clustering algorithms and $\mathcal{M}$ the space of all clustering evaluation metrics. In other words, an element of $\mathcal{A}$ is a function mapping any set of images to a corresponding set of labels:

$$a \in A \iff a : \mathcal{I}^n \to \mathrm{L}^n, \tag{6.3}$$

where $\mathrm{L} = \{1, ..., K\}$ is the set of possible labels, n is the number of images in the dataset and K is the expected number of clusters. In practice, K does not have to be fixed but in this thesis we only deal with clustering algorithms for which K is known. We also assume that for each image in $\mathcal{I}$, there exists an associated ground-truth label, which can be seen as an underlying subjective optimal classification (see Section 6.2.1). For example, in the context of unsupervised robotic sorting, although no information is known by the robot a priori, we can suppose that a qualified worker should be able to judge the quality of the sorting afterwards. Following these notations, an elements of $\mathcal{M}$ is a function which takes two sets of integers as inputs, the predicted labels and the ground truth labels, and outputs a real valued score, usually in $[0, 1]$:

$$m \in \mathcal{M} \iff m : \mathrm{L}^n \times \mathrm{L}^n \to [0, 1]. \tag{6.4}$$

Now, let Ω_∞ be the conceptual infinite set of all possible objects. Similarly, let $\mathcal{Z}_\infty^\omega$ be the conceptual infinite set of all possible scenes containing object ω. A natural image clustering problem with n images is defined by

$$\text{cp} = \{v_{\zeta_{\omega[i]}}(p_c[i]) \mid \text{for all } i \in \{1, ..., n\} : \ \omega[i] \in \Omega_\infty, \ \zeta_{\omega[i]} \in \mathcal{Z}_\infty^{\omega[i]}, \ p_c[i] \in \mathcal{D}_{\zeta_{\omega[i]}}\}.$$

In other words, any set of natural images containing an underlying label (defined by the $\omega[i]$'s) can be viewed as a clustering problem. Let $\mathcal{L}_{\text{cp}}$ be the ground truth labels associated with cp and $a(\text{cp})$ be the cluster assignments (predictions) for cp with algorithm a. Finally, for a given natural image I, generated with the hand-mounted camera c, we define CP_∞^I, the infinite set of all possible natural image clustering problems containing I. Then, $S^{a,m}$ is defined by

$$S^{a,m}(I) = \mathop{\mathbb{E}}_{\text{cp} \in \text{CP}_\infty^I} [m(a(\text{cp}), \mathcal{L}_{\text{cp}})], \tag{6.5}$$

which is the average score under metric m of all possible clustering problems containing I. $S^{a,m}(v_{\zeta_\omega}(p_c))$ is high if the image generated with camera c in the pose p_c, observing object ω in scene ζ_ω, is good for clustering ω with algorithm a and low if not, where good means having a high score under m.

We assume that a and m are respectively a good image clustering routine and a good clustering metric, i.e. they have been shown to work well in practice. In the rest of the chapter, we also assume that $S^{a,m}$ and S are highly positively correlated. This assumption is based on the intuition that a semantically meaningful image should be properly clustered with similar objects by a good clustering pipeline. Indeed, as detailed later, the common clustering pipeline used in this paper consists in extracting features from a deep feature extractor and clustering the new set of features using a standard clustering algorithm. This choice for a is in line with the definition of semantic meaningfulness proposed in Section 6.2.1, as the final representation of a view, passed to the clustering algorithm, is a vector a features extracted from a pretrained CNN. Another motivation for choosing a clustering-based estimate for the semantic function is that supervised classification or object detection methods might not be adapted. Indeed, to compute the Monte-Carlo estimate of such function (see Section 6.2.3), the selected algorithm needs to be run many times on relatively small datasets. Doing this in a supervised way has high chances to result in overfitting, in which case all views would have high semantic scores.

6.2.3 Second relaxation: clusterability on a finite dataset

As it is not feasible to consider all possible scenes containing all possible objects, we further relax the above definitions to consider a finite dataset. Let Ω_N be a finite set of objects containing N elements. For a given element ω of Ω_N, we also define $\mathcal{Z}^{\omega}_{N_\omega}$, a set of N_ω scenes containing ω, and $\mathrm{P}^{\zeta_\omega}_{N_{\zeta_\omega}}$ a set of N_{ζ_ω} camera poses observing object ω in scene ζ_ω. In other words, the set

$$X = \{v_{\zeta_\omega}(p_c) \,|\, \omega \in \Omega_N, \zeta_\omega \in \mathcal{Z}^{\omega}_{N_\omega}, p_c \in \mathrm{P}^{\zeta_\omega}_{N_{\zeta_\omega}}\}$$

is an natural image dataset containing $\sum_{\omega\in\Omega_N} \sum_{\zeta_\omega \in \mathcal{Z}^{\omega}_{N_\omega}} (N_{\zeta_\omega})$ images. For a given image $I \in X$, if the dataset X is large and diverse enough, an estimate of $S^{a,m}(I)$ can be computed by

$$S^{a,m}_X(I) = \mathop{\mathbb{E}}_{\mathrm{cp}\in \mathrm{CP}^I_X} [m(a(\mathrm{cp}), \mathcal{L}_{\mathrm{cp}})], \tag{6.6}$$

where CP^I_X is defined like CP^I_∞ with cp's sampled from X.

For large datasets, it might be computationally intractable to compute $S^{a,m}_X(I)$ as the number of possible combinations of images grows exponentially with the number of views. Thus, we propose to compute the Monte-Carlo estimate

$$\hat{S}^{a,m}_X(I) = \mathop{\mathbb{E}}_{\mathrm{cp}\in \mathrm{CP}^I_{X,MC}} [m(a(\mathrm{cp}), \mathcal{L}_{\mathrm{cp}})], \tag{6.7}$$

where $\mathrm{CP}^I_{X,MC}$ is a subset of N_{MC} elements of CP^I_X, and N_{MC} is a large natural integer ($N_{MC} \geq 2\times10^5$ in our experiments). The method used for sampling clustering problems from X and thus creating $\mathrm{CP}^I_{X,MC}$ is explained in Section 6.4.2.

6.2.4 Partially-observable Semantic View Selection

Given an object ω in a scene ζ, the relaxed SVS problem, aims to find a camera pose p_c such that $\hat{S}^{a,m}_X(v_\zeta(p_c))$ is high. In a generic robotic pipeline, the exact setup of a scene is generally unknown and needs to be estimated from partial observations. Let ψ_ζ be the observation from which we want to estimate the elements composing ζ and their relative poses. For example, ψ_ζ can be a top-view image, taken from an initial predefined camera pose. Our approximation of the clusterability function score is then dependent on ψ_ζ as a surrogate for the exact scene setup. More concretely, we want to optimize the parameters α of a function $f_\alpha : \{\psi_\zeta, p_c\} \to s \in \mathrm{D}_m$, where D_m is the output domain of the metric m (usually $[0, 1]$), such that s is an estimate of $\hat{S}^{a,m}_X(v_\zeta(p_c))$. A typical practical choice for f_α would be a convolutional neural network, where α represents its trainable parameters.

6.3 Dataset Construction

To tackle the proposed relaxed SVS problem, we have built an image dataset representing various everyday objects under different poses, and observed under multiple views with a camera mounted on the end-effector of a UR10 robot manipulator (see Figure 6.1). The dataset can be downloaded at `https://github.com/jorisguerin/SemanticViewSelection_dataset` and its statistics can be found in Table 6.1. In this dataset, for a given object, the scenes are composed of the object alone on a wooden table, under different poses. For example, Figure 6.1 images taken from different camera poses for one of such scenes for a comb.

Table 6.1: Statistics of the proposed multi-objects/multi-pose/multi-view image dataset.

# Classes	# Object/class (*total*)	# Poses/object (*total*)	# Images/pose (*total*)
29	4-6 (*144*)	3 (*432*)	17-22 (*9112*)

6.3.1 Estimating object location and size

The dataset was collected using an Asus Xtion RGBD sensor, hand-mounted on a UR10 robot manipulator. For a given object ω in a given pose, we gather images corresponding to several camera poses, with ω centered in the image. The first step is to estimate the location of the Geometrical Center of the object (GC_o). To do so, we place the robot in an initial pose such that the camera can see the entire workspace in which objects can be placed. We store a background image of this pose, corresponding to what the camera sees when there is no object. Then, using RGB background subtraction, the xy-contour of the object is obtained, the z axis being vertical. From this contour, we estimate the x and y components of GC_o, the width and the length of ω. Finally, we compare the minimum values of the point cloud inside and outside the xy-contour to estimate both the z component of GC_o and the height.

6.3.2 Parameterization of camera poses

To parameterize camera poses, we define a reference frame at GC_o. We then compute $d = \sqrt{length^2 + width^2 + height^2}$, the diagonal of the object's bounding box, and define the radius R such that d takes 70% of the smallest dimension of the image if the optical center of the camera (OC_{cam}) is at a distance R of GC_o and z_{cam} is

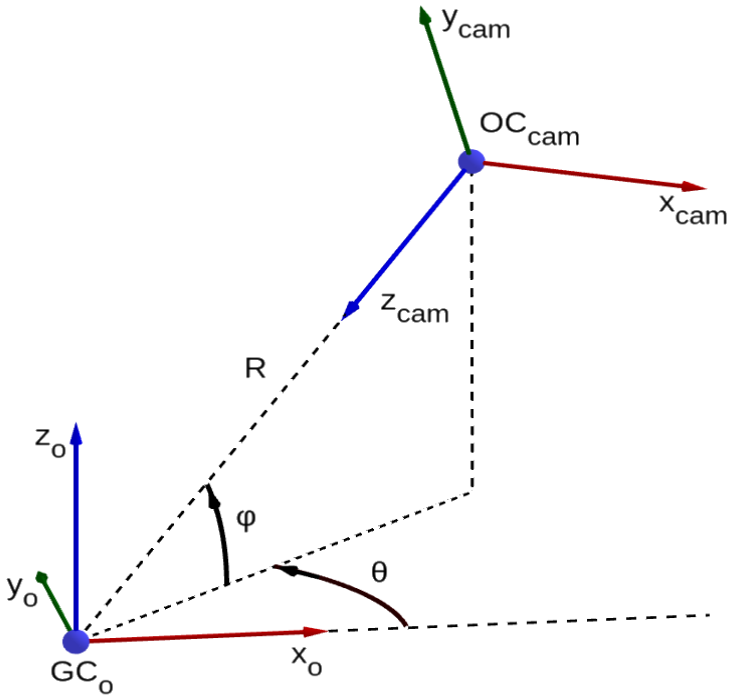


Figure 6.3: Definition of the parameters used to sample camera poses (R, θ and φ).

pointing towards GC_o. The camera poses are sampled on the half-sphere of radius R, centered at GC_o, such that z_o is positive. For each position of OC_{cam} on the sphere, the camera is positioned such that z_{cam} is pointing towards GC_o, x_{cam} is in the xy_o plane and y_{cam} is pointing "upwards".

6.3.3 View sampling and data collection

On the sphere, the location of OC_{cam} is localized by two angles, θ and φ, which are defined as in Figure 6.3. Hence, a camera pose is simply represented by a (θ, φ) pair. In our implementation, θ is sampled every 45° between 0° and 315° but excluding 270°, φ is sampled every 15° between 45° and 75°. The views for $\theta = 270°$ correspond to configurations where the camera is oriented towards the robot base. They were not collected in the dataset to avoid seeing the robot on the images. The other missing values come from unreachability of the camera poses with the robot manipulator, which occurs when the RRT connect ([Kuffner and LaValle, 2000]) planner fails to generate a valid plan. Furthermore, while it could be interesting to sample angles lower than $\varphi = 45°$, these configurations are often unreachable because the robot would collide the table. A subset of the views gathered for one object in a particular pose can be seen in Figure 6.4.

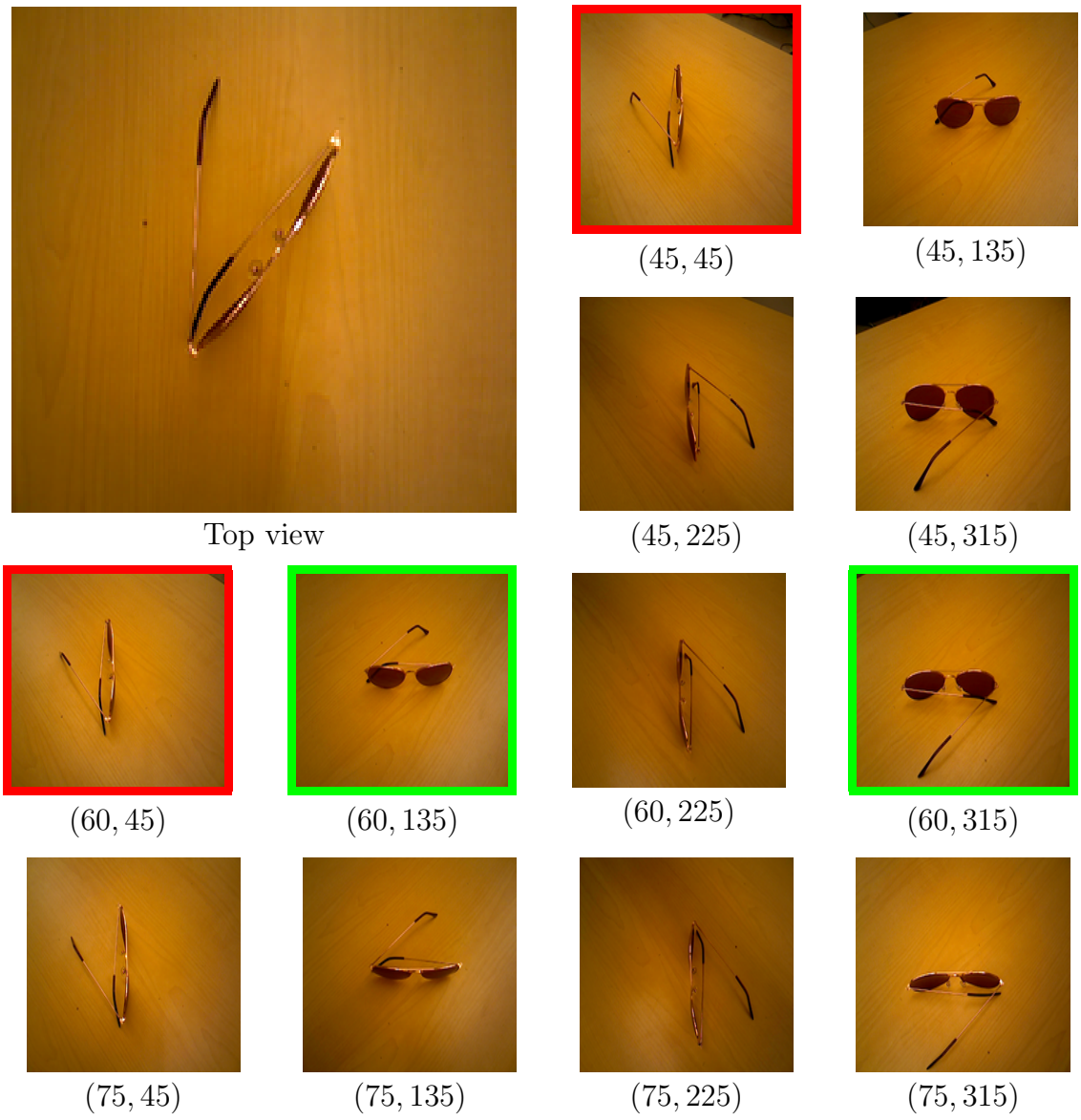


Figure 6.4: *Best viewed in color.* Subset of views for one instance of the sun glasses class in a particular scene. The two images with highest (resp. lowest) FM individual indexes are framed in green (resp. red). *See Sections 6.4 and 6.5 for more details.*

6.4 Proposed approach

6.4.1 Clustering pipeline and metric

Given an image I, the clusterability function $\hat{S}_X^{a,m}(I)$, used to represent the semantic function, is defined by both a good clustering pipeline a and a clustering evaluation metric m.

In this work, we use the image clustering pipeline described in Chapter 3, which consists in getting a new representation of each image from the last layer of a deep CNN feature extractor, pretrained on ImageNet ([Russakovsky et al., 2015]), and clustering the new set of features using a standard clustering algorithm. Although some variants of this algorithm are tested in Section 6.5, the standard pipeline in this paper uses Xception ([Chollet, 2016]) to extract features, and agglomerative clustering ([Murtagh, 1983]) to cluster the deep features set. We use the implementation and weights of Xception proposed by the Keras library ([Chollet, 2015]).

The clustering metric chosen to represent the clusterability is the Fowlkes-Mallows (FM) index ([Fowlkes and Mallows, 1983]), which was already used in Chapter 4 and is defined by

$$FMI_{\mathrm{cp},a} = \frac{TP}{\sqrt{(TP+FP)(TP+FN)}}, \tag{6.8}$$

where TP, FP and FN respectively represent the number of true positive, false positive and false negative pairs after clustering cp using a. The FM index ranges between 0 and 1. We choose this index because it can be converted straightforwardly to a local form

$$FMI^i_{\mathrm{cp},a} = \frac{TP_i}{\sqrt{(TP_i+FP_i)(TP_i+FN_i)}}, \tag{6.9}$$

where FMI^i represents the individual FM score of image $I_i \in \mathrm{cp}$, TP_i, FP_i and FN_i respectively represent the number of true positive, false positive and true negative pairs containing I_i. This individual form is used in the next section to reduce the sample complexity for computing $\{\hat{S}^{a,m}_X(I), I \in X\}$. The clustering pipeline and metric being chosen, we drop the a and m superscipts in the coming sections.

6.4.2 Training Set

From the 29 categories composing the dataset, five are chosen at random to constitute a validation set which is neither used for fitting the clusterability scores nor for training the view selection network. In this section, X refers to all the views composing the 24 remaining categories. From X, a clustering problem is created by sampling randomly the number of categories, the selected categories, the number of objects per category, the selected objects, the pose for each object (scene), and one view (camera pose) for each scene. As every estimated quantity is computed on the training set from now on, we drop the subscript X. To build the training set for

the SVS problem, we start by generating a large set of N_{cp} random clustering problems $\text{CP}_{MC} = \{\text{cp}_i, i \in N_{\text{cp}}\}$. Then, for each (cp, I) pair, we define the following intermediate score

$$\tilde{s}_{\text{cp}}(I) = \begin{cases} FMI^I_{\text{cp}} & \text{if } I \in \text{cp}, \\ 0 & \text{otherwise.} \end{cases} \tag{6.10}$$

The *individual semantic view score* of image I is then defined by

$$\hat{s}(I) = \sum_{\text{cp} \in \text{CP}_{MC}} \tilde{s}_{\text{cp}}(I)/N^I_{\text{cp}}, \tag{6.11}$$

where N^I_{cp} is the number of elements in CP_{MC} containing I. Likewise, for a given I, we can define a *global semantic view score* $\hat{S}(I)$ by replacing FMI^I_{cp} by FMI_{cp} in (6.10).

We build CP_{MC} such that $\min(\{N^I_{\text{cp}},\ I \in X\})$ is at least 2×10^5. In practice, this requires to solve $N_{\text{cp}} \approx 3 \times 10^7$ clustering problems. Because of the high computational expenses of this process, we cannot get a much higher number of samples for the Monte Carlo estimate. Hence, $\hat{s}(I)$ seems more appropriate than $\hat{S}(I)$ to estimate the semantic content of I because it evaluates the individual contribution of each view to the global clustering results.

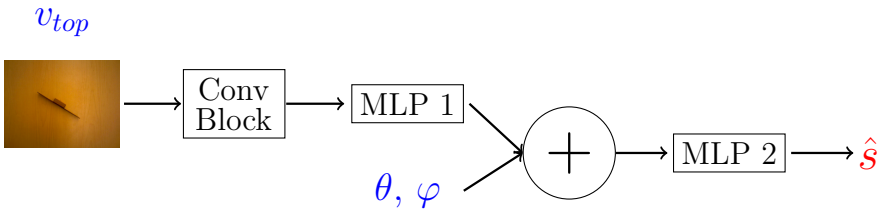
Hence, our training set for the next section is composed of $\{\{I^I_{\text{top}}, \varphi^I, \theta^I\}, \hat{s}(I)\}$ input/output pairs, where I^I_{top} represents the top view image associated with I, φ^I and θ^I are the angles parameterizing I. We note that, for each pose, the scores among all the views are scaled to the $[0, 1]$ interval to help training, as some objects are harder to cluster than others. In such case the best views of a difficult object might have lower $\hat{s}$ values than the worst views of an easy object. This makes the view selection problem harder because predicting the intrinsic "clusterability" of an object from a poor view can be very challenging.

6.4.3 Learn to predict semantic scores

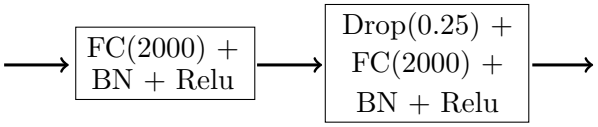
After computing semantic view scores for each view in our training set, we aim to solve the SVS problem introduced in Section 6.2.4. To do so, we train a multi-input neural network architecture to predict $\hat{s}^I$ from a triplet $(I^I_{top}, \theta^I, \varphi^I)$.

The top image is first passed through a VGG convolutional block ([Simonyan and Zisserman, 2014]) with 19 layers initialized with weights pretrained on ImageNet. Then, the outputs of the convolution block are fed into a first multi-layer perceptron (MLP). The outputs of this first MLP are concatenated with the angular inputs and fed into a second MLP, which outputs the semantic estimate. The architecture is

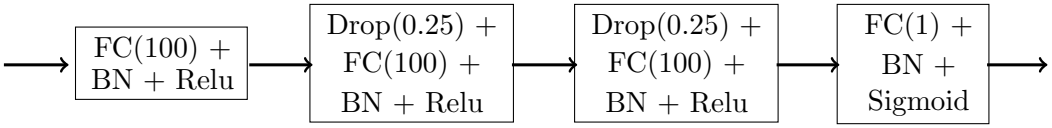
summarized in Figure 6.5, where BN denotes a batch normalization layer, Drop(x) a dropout layer with x% drops and FC(k) a fully connected layer with k neurons. ReLu and Sigmoid are standard activation layers.



(a) Overall architecture



(b) MLP 1



(c) MLP 2

Figure 6.5: Proposed SV-net architecture. *Inputs are in blue and outputs in red. The Conv Block is the convolutional part of VGG19 (until "block4_pool" layer).*

To train this network, we use the Adam optimizer ([Kingma and Ba, 2014]), with an initial learning rate of 10^{-3}. The choice of the architecture was cross-validated by removing randomly two categories from the training set. At test time, all the dropout rates are set to 0.

The choice of treating SVS problem by predicting a score from proposed camera poses instead of regressing directly on these poses has two main motivations. First, when dealing with robots, it might be impossible to plan a trajectory to some views, hence, it seems more relevant to only consider reachable views. Then, the SVS problem may contain many possible solutions, in which case there is no unique mapping between a top view and a good camera pose. Indeed, a network would likely converge to the average of all good poses, which might not be a good pose.

6.5 Experiments

6.5.1 Baseline for comparison

Given an object clustering problem, we define a *view selector* as any process to select the camera pose to observe the objects. To evaluate the quality of a view selector on a clustering problem, we compare its results under a certain (a, m) pair against two baseline view selectors. The first method is usually the one implemented when dealing with autonomous robot sorting, it consists in observing the object from the top view. This view selector will be noted TOP in our experiments. Another baseline view selector, denoted RAND, consists in choosing the views uniformly at random among the possible views.

We consider a view selector successful if it can outperform these two baselines. Indeed, borrowing from the reinforcement learning literature, comparing against the best fixed view can be assimilated to some form of regret analysis. Computing the best fixed view for our problem would be too computationally expensive. Instead, we compare against one arbitrary fixed view (TOP) and also against randomly selected views to ensure that the chosen fixed view was not particularly weak.

6.5.2 Evaluation of the individual semantic view score

The individual semantic view scores are fit using a particular (a, m) pair. Therefore, to evaluate it, we must test it on additional clustering pipelines and metrics. We vary pipelines by changing both the deep feature extractor and the clustering algorithm. The three pipelines tested are denoted XCE_AGG, VGG_AGG and XCE_KM, where XCE stands for Xception, VGG for VGG19, AGG for agglomerative clustering and KM for KMeans. As for the clustering metrics, we use the Fowlkes-Mallows index (FM), normalized mutual information (NMI) and cluster purity (PUR), which are three commonly used metrics to evaluate clustering algorithms when the ground truth is known.

We compare two view selectors against RAND and TOP. The first one, denoted $\mathrm{OPT}_{\mathrm{ind}}$, consists of choosing the image with the highest individual semantic view score. Following the notations from Section 6.2.3, for a given object ω in a given scene ζ_ω, the camera poses are sampled from $\mathrm{P}^{\zeta_\omega}_{N_{\zeta_\omega}}$. The $\mathrm{OPT}_{\mathrm{ind}}$ view selector chooses the pose

$$p_c^{\mathrm{ind}} = \underset{p_c \in \mathrm{P}^{\zeta_\omega}_{N_{\zeta_\omega}}}{\mathrm{argmax}}(\hat{s}(v_{\zeta_\omega}(p_c))). \tag{6.12}$$

Similarly, the second view selector, denoted OPT_{glob}, chooses the pose that produces the image with the highest global semantic view score:

$$p_c^{\text{glob}} = \underset{p_c \in \text{P}_{N_{\zeta_\omega}}^{\zeta_\omega}}{\text{argmax}}(\hat{S}(v_{\zeta_\omega}(p_c))). \tag{6.13}$$

The results are averaged over 10^4 clustering problems and reported in Table 6.2.

Table 6.2: Semantic view scores validation. Comparison of clustering results among different view selectors on the training set. *for each* (c, m) *pair, the best view selector is in bold.*

		FM	NMI	PUR
XCE_AGG	TOP	0.48	0.78	0.73
	RAND	0.50	0.78	0.74
	OPT_{glob}	0.85	0.94	0.93
	OPT_{ind}	**0.87**	**0.95**	**0.94**
XCE_KM	TOP	0.44	0.75	0.71
	RAND	0.46	0.76	0.72
	OPT_{glob}	0.81	**0.93**	0.91
	OPT_{ind}	**0.83**	**0.93**	**0.92**
VGG_AGG	TOP	0.39	0.72	0.67
	RAND	0.38	0.72	0.66
	OPT_{glob}	0.49	0.78	0.73
	OPT_{ind}	**0.52**	**0.79**	**0.74**

The first thing to note is that both semantic estimators, although fit with $a =$ XCE_AGG and $m =$ FM, seem to pick views which are much better than TOP and RAND. This is not surprising as these results were computed on the dataset used to compute the estimators. However, it strengthens the belief that $\hat{s}$ and $\hat{S}$ are good semantic function estimators as they generalize to other feature extractors, clustering algorithms, and metrics. Surprisingly, we also note that $\hat{S}$ and $\hat{s}$ performances are very similar. This might mean that the number of samples in the MC computation is sufficient for $\hat{S}$ to be a good estimator of S. However, in our experiments, we also acknowledge that for a given object ω in a given scene ζ, the values of the set $\{\hat{S}(v_\zeta(p_c)) \mid p_c \in \text{P}_{N_{\zeta_\omega}}^{\zeta_\omega}\}$ are much closer to each other, which reveals that information about individual data is lost when considering the global estimator. The slightly better results of the $\hat{s}$ estimator, as well as its better separability, justifies its use for training SV-net.

Finally, we refer the reader back to Figure 6.4, where images with both high and low $\hat{s}(I)$ values have been outlined. This gives a qualitative validation of the index

relevance for estimating the semantic function. Indeed, it is easier to tell that the robot is looking at sun glasses from the green-outlined images than from the red-outlined ones.

6.5.3 Evaluation of the learned semantic view selector

To evaluate our semantic view selection network (SV-net), we adopt a similar approach to the one in the previous section. The SV-net view selector is compared against RAND and TOP under various configurations on the validation set, which was neither seen for $\hat{s}(I)$ computation, nor for training SV-net. The complete list of the 29 categories can be found in the dataset folder (`https://github.com/jorisguerin/SemanticViewSelection_dataset`). Results are averaged over 10^4 clustering problems randomly sampled from the validation set and are reported in Table 6.3. In the results presented, the five randomly chosen categories composing the validation set were: comb, hammer, knife, toothbrush and wrench. We note that SV-net was able to predict views which are better than TOP and RAND, which is a more remarkable result than in the previous section as these kind of objects where never seen by the network before. SV-net is able to extract sufficient information from a single top view image to predict if a camera pose will provide good high-level features about the object being observed.

Table 6.3: SV-net validation. Comparison of clustering results between different view selectors on the test set. *for each* (c, m) *pair, the best view selector is in bold.*

		FM	NMI	PUR
XCE_AGG	TOP	0.44	0.51	0.70
	RAND	0.48	0.56	0.74
	SV-net	**0.55**	**0.63**	**0.78**
XCE_KM	TOP	0.44	0.51	0.70
	RAND	0.48	0.55	0.73
	SV-net	**0.55**	**0.62**	**0.78**
VGG_AGG	TOP	0.46	0.53	0.71
	RAND	0.44	0.51	0.70
	SV-net	**0.48**	**0.55**	**0.73**

As a qualitative validation, four samples of predicted images can be seen on Figure 6.6. We also underline that the absolute values of the clustering scores cannot be compared between tables. Indeed, the object considered are different and there is no guarantees that there exist views able to reach similar clustering accuracy when different classes of objects are considered.

(a) Example top views

(b) Associated SV-net selections

Figure 6.6: Examples of views predicted by SV-net.

6.6 Conclusion

6.6.1 Key results

In this chapter, we have introduced a new problem called semantic view selection. The SVS problem consists of finding a good camera pose to improve semantic knowledge about an object from a partial observation of the object. We created an image dataset and proposed an approach based on deep learning to solve a relaxed version of SVS problem.

By fitting an index based on averaged view clustering quality and training a neural network to predict this index from a top view image, we show that it is possible to infer which view results in good semantic features. This has many practical applications including autonomous robot sorting, which is generally solved from top view images only. Indeed, one can use the SV-net to enhance any sorting robot with the ability to select better views to reduce sorting errors. Results reported in Section 6.5.3 demonstrate that SV-net predictions outperform the approach proposed in Chapter 5. The views obtained with SVS are better than fixed views and have the potential to improve URS. Perspectives and future work regarding semantic view selection are discussed in Chapter 9.

6.6.2 Towards fully autonomous unsupervised robotic sorting

Parts II and III of the thesis focus on the decision making module of Unsupervised Robotic Sorting (URS). It was shown that by properly choosing the views under which to observe the objects and with a good image clustering pipeline, the results for objects classification can be improved. However, a complete sorting pipeline is composed of many other steps than decision making. To reach complete autonomy and robustness to different kinds of objects and environments, a sorting robot must have various other skills such as scene segmentation, objects localization, grasping, trajectory generation, control, etc. Hence, the next part focuses on two of such skills: trajectory learning and object localization, and thus constitutes a step towards context independent robotic sorting.

Part IV

Further developments

Chapter 7

Model independent trajectory optimization

Abstract

The methods for object understanding proposed in the previous chapters are important skills to increase the operating range of many robotic applications. Similarly, this part of the thesis, presents contributions in two other tasks that can increase autonomy, flexibility and robustness of various industrial applications, including robotic sorting. Chapter 8 deals with 3D object localization and this chapter focuses on precise trajectory learning. Recent reinforcement learning methods have enabled to accomplish difficult high dimensional robotic tasks under unknown dynamics, using iterative Linear Quadratic Gaussian (iLQG) control theory. These algorithms are based on building a local time-varying linear model of the dynamics from data gathered through interaction with the environment. These techniques often require an accurate model of the manipulated system to converge, which can be impractical for tasks where such a model is unknown or changing frequently. This chapter proposes a model independent version of iLQG by regressing the quadratic cost function directly from the data. This way, any sensor information can be used to design the cost function, thus making the trajectory learning easier to define and reprogram. The proposed approach is validated against another model independent method for a Cartesian positioning task, with various industrial robot models, in a simulated environment. Simulation is also leveraged to tune the hyperparameters of the method. These parameters are then transferred to a real industrial robot for both standard Cartesian positioning and a target reaching task with a laser pointer, for which a model cannot be computed in the general case.

7.1 Introduction

Chapters 2 to 6 focus on the unsupervised robotic sorting application. This problem is closely related to unsupervised object understanding, an important skill for many autonomous robotic tasks. However, to implement industrial robotic applications that work under a broad range of conditions, many other skills need to be mastered. In this part of the manuscript, we study two of such skills: trajectory learning and 3D object localization. For example, in the robotic sorting context, these two skills are essential to build a functional implementation that works in various environments, and for any kind of objects. Indeed, object localization is a required skill to be able to observe the different objects (view selection) and to grasp them. Likewise, to navigate the robot between different configurations, the robotic system may benefit from smart trajectory optimization methods, which is the focus of this chapter. We propose to adapt a recent method based on optimal control, to accept any kind of loss function based on sensor signals. The remainder of this section gives a specific introduction about trajectory learning.

7.1.1 Literature overview

Optimal feedback control theory provides an efficient framework for robot manipulators movement generation as it enables to compute both an optimal open-loop trajectory and a feedback controller at once. To carry out such tasks, classical optimal control uses a model of the dynamics and selects the solution that minimizes a properly designed cost function. Under linear dynamics and quadratic cost, an optimal solution can be found analytically thanks to the widely studied theory of Linear-Quadratic-Regulators (LQR) ([Lewis et al., 2012]). When the dynamics is a more complex, non-linear function, the problem becomes more difficult. However, the iterative Linear-Quadratic-Regulator algorithm (iLQR) ([Li and Todorov, 2004]) still enables to converge towards a locally optimal solution by iteratively fitting local linear approximations to the dynamics.

On the other hand, autonomous learning of manipulation skills have received much interest over the past decades. Thanks to different Reinforcement Learning (RL) techniques ([Kober and Peters, 2012]), various problems involving robot manipulators have been explored and successfully applied (([Lin, 2009]), ([Deisenroth et al., 2011]), ([Park et al., 2007])). Among RL techniques, policy search methods seem to be the most appropriate for high dimensional robotic control problems ([Deisenroth et al., 2013]). Recently, several researchers have proposed methods to link these two fields by

using iterative Linear-Quadratic-Gaussian control (iLQG) (([Todorov and Li, 2005]), ([Tassa et al., 2012])) to compute control laws under unknown dynamics (([Mitrovic et al., 2010]), ([Levine and Abbeel, 2014])). This approach is interesting as it removes the need to model the dynamics, which can be difficult for complex systems and environments.

7.1.2 Limitations

Besides the dynamics, an optimal control problem is also defined by its cost function. In order to find a good trajectory it is required to design a proper cost function in relation to the task to be carried out. Nevertheless, in most applications, we acknowledge that the cost function is quadratic and expressed explicitly in terms of the state and control variables. Such form of the cost function makes it easy to run the iLQG algorithm and appears to be well suited for many applications. However, in some cases, it can be useful to formulate the cost function in terms of variables that are not explicitly the state and control vectors. For example, ([Levine et al., 2015b]) propose a cost function defined in the Cartesian space whereas the robot is controlled in joint space. In such cases, a model of the system (i.e. a direct Denavit-Hartenberg (DH) model ([Dombre and Khalil, 2013]) of the manipulator) is required to express analytically a quadratic expansion of the cost function, which is necessary to run iLQG. Such dependence on a robot model can be problematic since calibration is a complex and time consuming process (([Majarena et al., 2010]), ([Elatta et al., 2004])) and because the method is very specific to a certain system (([Nubiola and Bonev, 2013]), ([Jubien et al., 2014])). It is impossible to derive a model taking into account every dynamic phenomena. For example, if the payload on the robot is changed, the model calibration might not be valid anymore as strains might appear. In an autonomous learning framework, it does not seem appropriate to have to redesign a model after any change on the system.

7.1.3 Contributions and chapter organization

In this chapter, we propose a method to overcome the model dependence issue by regressing the quadratic approximation of the cost function from samples, in the same way as for the dynamics. In this way, the cost function can be defined from any measurable quantity, which allows to have a more appropriate cost in some cases. Moreover, the cost function can be defined in a more intuitive way and robot programming can become more accessible to non-specialists.

The task chosen to validate the method is a Cartesian positioning task. The robot is trained to learn how to reach a Cartesian point with its end-effector using only position sensors information and without any DH model. The cost function is the distance measured between the end effector and the target Cartesian position while the robot is controlled in joint space. To emphasize the interest of the method, we also propose to learn trajectories for target reaching with a laser pointer. For this task, the DH model cannot be computed in general because the transform to the end effector is unknown.

Two different methods are proposed to update the controller. The first one, which is a first order optimization method, is less elaborated but has less parameters to tune and is faster to implement. Therefore, the cost function learning scheme described above is validated using first-order controller updates. Using the V-REP software ([E. Rohmer, 2013]), robots with different specifications (number of joints, length of links, ...) learn the same positioning task, which illustrates that the method is independent of the robot model. We compare our results with another method where the Cartesian distance is included in the state vector and the cost is thus a function of the dynamics. Our method is shown to be more stable and to converge faster to high precision positioning. Once the method is shown to work, a second order optimization technique is implemented and its parameters are tuned using V-REP. Finally, the parameters found are tested on a real Kuka LBR iiwa robot on both the standard Cartesian positioning problem and a laser pointer target reaching task. This work received an IEEE IES Student Travel Paper Award ([Guérin et al., 2016]). A video that summarizes the key findings from this chapter is available at: `https://www.youtube.com/watch?v=M9RK2XznHEE&index=4&list=PLO5umi31c_83SMDdkk26shJZbOnbqbSpl`.

The chapter is organized as follows. The derivation of the iLQG algorithm is written in Section 7.2, where both first order and second order methods are proposed to update the controller. In Section 7.3, the regression method for cost function estimation is validated through simulation. We tune the parameters for second order controller updates with V-REP simulation in Section 7.4 and in Section 7.5 we validate them experimentally on a real robot.

7.2 Trajectory optimization using ILQG

7.2.1 Overview of Reinforcement Learning and Trajectory Optimization

Reinforcement Learning (RL) is a subfield of Machine Learning that studies the behavior of an *agent* taking *actions* in an *environment*. The environment responds to each action by rewarding (or penalizing) the agent. The goal of the agent is to maximize such reward (or minimize the cost). In this chapter, we consider the episodic setting. For a better overview of the field of RL, the reader can refer to the following good survey ([Kober and Peters, 2012]). To go further, we also suggest the following textbook ([Sutton and Barto, 1998]).

At a given time t, the RL framework is composed of three main elements:

- x_t, the *state* of the system, is a vector that contains all relevant information about the configuration of the system (or agent). For example, in our case, the agent is the serial robot and its state vector is composed of the angular joint positions.

- u_t, the *action* taken by the agent. This control vector represents commands sent to change the state of the system. For us, u_t corresponds to joint target positions.

- l_t, the *cost* resulting from sending command u_t when in state x_t. In a robotic Cartesian positioning task, the cost at time t is proportional to the distance to the target point.

With these elementary notations, we can introduce the following concepts:

- The *cost* and *dynamics* functions at time t are defined as follows:

$$l_t = L_t(x_t, u_t), \tag{7.1}$$

$$x_{t+1} = F_t(x_t, u_t). \tag{7.2}$$

  L_t outputs the cost and F_t the next state, both with respect to current state and action. We note that *environment* and *dynamics* refer to the same thing.

- The *controller* is the function we want to optimize. For a given state, it needs to output the action with smallest cost that follows the dynamics. In our case, at time step t, it is denoted by Π_t and has the special form of a time-varying linear controller:

$$u_t = \Pi_t(x_t) = K_t x_t + k_t. \tag{7.3}$$

Then, a *trajectory* τ, of fixed length T, is defined by the repetition T times of the pattern shown in Figure 7.1. Mathematically, it can be denoted by

$$\tau = \{x_0, u_0, x_1, u_1, ..., u_{T-1}, x_T\}, \tag{7.4}$$

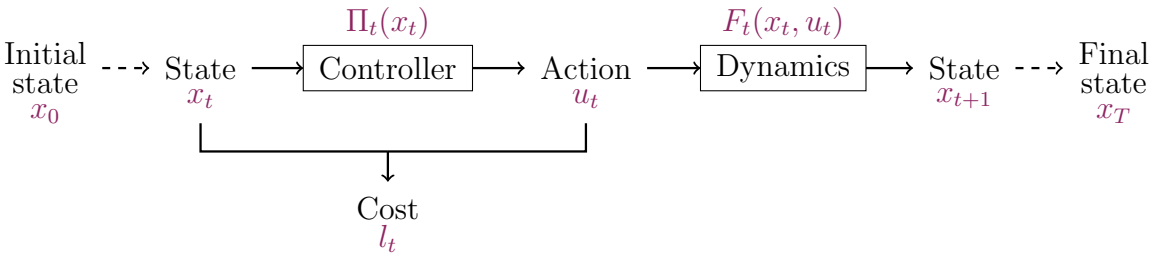


Figure 7.1: Block diagram to define a trajectory and summarize the notations.

The guiding principle of iLQG is the following:

- From a nominal trajectory, denoted $\bar{\tau} = \{\bar{x}_0, \bar{u}_0, \ldots, \bar{x}_T\}$, a new improved controller is derived by:
  - approximating the Taylor expansions of the cost and dynamics (Section 7.2.2),
  - updating the controller according to this approximations (Section 7.2.3).
- From this new controller, a new nominal trajectory is computed.

This process is repeated until a good enough controller is reached. In most application, this iterative process turns out to converge rather rapidly towards a locally optimal trajectory.

7.2.2 Local approximations of cost and dynamics

As explained earlier, from a given nominal trajectory $\bar{\tau}$, the goal is to update the controller such that the cost is minimized in its neighborhood. In this process, the first step is to compute local approximations of the cost function and dynamics around the nominal trajectory:

$$F_t(\bar{x}_t + \delta x_t, \bar{u}_t + \delta u_t) \approx \bar{x}_{t+1} + [F_{x_t}, F_{u_t}] \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}, \tag{7.5}$$

$$L_t(\bar{x}_t + \delta x_t, \bar{u}_t + \delta u_t) \approx \bar{l}_t + [L_{x_t}, L_{u_t}] \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta x_t^T, \delta u_t^T \end{bmatrix} \begin{bmatrix} L_{x,x_t} & L_{x,u_t} \\ L_{u,x_t} & L_{u,u_t} \end{bmatrix} \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}, \tag{7.6}$$

where δx_t and δu_t represent variations from the nominal trajectory and capital letters indexed by one (respectively two) letters represent sub-vectors (respectively submatrices) of the appropriate Jacobian (respectively Hessian) matrices.

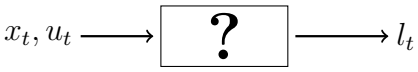
7.2.2.1 Different approaches

In the original paper for iLQG ([Todorov and Li, 2005]), equations (7.1) and (7.2) are considered known and both Taylor expansions are computed analytically. Later on, it was proposed to compute the linear dynamics through regression on observed values (([Levine and Abbeel, 2014]), ([Mitrovic et al., 2010])). This way, trajectories can be found without a model of the environment, which is difficult to have in most situations. However, the cost function still needs to be expressed directly in terms of the state and action variables. If one wants to design a cost function from sensor measurements, it might not be straightforward to obtain the partial derivatives with respect to the state and control vectors. A good example is the task of robot positioning, where the cost function is a Cartesian distance and the variables controlled are joint positions (Section 7.3). In this case, different methods, illustrated in Figure 7.2, can be used to compute the second order Taylor expansion of the cost.

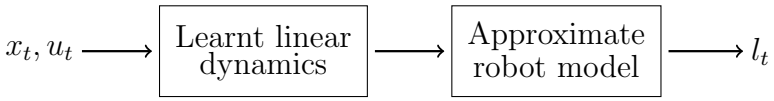
The first method (Figure 7.2b) consists in using a model of the robot to convert the angular state variables into Cartesian end-effector positions. This robot model is then used to derive the cost function with respect to the angular positions. In some practical situations, such model can be hard to obtain with enough precision, e.g. humanoid robots with a lot of degrees of freedom. Indeed, coming up with a precise model of a robot manipulator is a hard and tedious task ([Majarena et al., 2010]). Moreover, the need for a model of the robot reduces the generality of the algorithm and its precision is dependent on the model, which varies from one robot to another and even with the same robot loaded differently.

As stated in the introduction, the framework for this chapter requires a model-free method. One possibility to avoid the model is to increase the state vector with a measurement directly proportional to the cost function (Figure 7.2c). For example, for the positioning task in Section 7.3, the Cartesian distance can be considered to be part of the state. By doing this, the cost function can be expressed directly from the state and the dynamics and the derivation can be done. This trick enables to define the cost function from any measurable quantity. However, as shown in Figure 7.2c, the cost function approximation is not really quadratic as it is computed from a first order approximation of the distance.

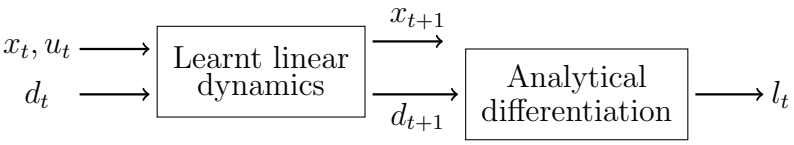
In this chapter, we propose a different model independent approach, illustrated in Figure 7.2d, in which the cost function is directly approximated quadratically. This method is developed in the next section.



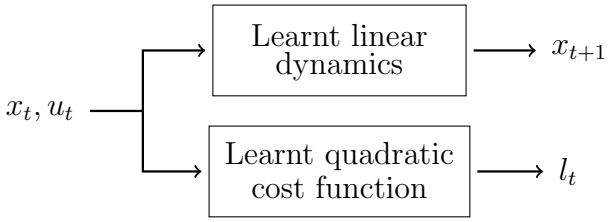
(a) Function to approximate.



(b) Using a model of the robot.



(c) Including the distance d_t in the state representation.



(d) Learning the quadratic approximation of the cost.

Figure 7.2: Different ways to compute the second order Taylor expansion of a Cartesian distance cost function from angular state and control vectors.

7.2.2.2 Quadratic cost approximation through exploration and regression

The proposed method consists in computing both approximations following an exploration and regression scheme. The first stage generates a certain number N of random trajectories around the nominal. These trajectories are normally distributed around $\bar{\tau}$ with a certain time-varying covariance, denoted Σ_t for time step t. Therefore, during sample generation, the controller is stochastic and follows:

$$\Pi_t(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, \Sigma_t), \ \ \forall t \in \{0, ..., T-1\}, \tag{7.7}$$

where $\mathcal{N}(\mu, \Sigma)$ is the normal distribution of mean μ and standard deviation Σ. From these samples, we can make two regressions, a linear one ([Freedman, 2009]) to get the dynamics and a quadratic one ([De Brabanter et al., 2013]) to approximate the cost function.

A quadratic regression consists in transforming the data with a second order polynomial kernel and applying a linear regression on the resulting higher dimensional data. In the general case, a quadratic expansion of the cost w.r.t. δx_t and δu_t

(respectively n and m entries) is obtained by solving a linear regression with input data in the form:

$$[1, \underbrace{\delta x_1, ..., \delta x_n}_{n}, \underbrace{\delta x_1^2, \delta x_1 \delta x_2, ..., \delta x_n^2}_{\frac{n(n+1)}{2}}, \underbrace{\delta u_1, ..., \delta u_m}_{m}, \underbrace{\delta u_1^2, ..., \delta u_m^2}_{\frac{m(m+1)}{2}}, \underbrace{\delta x_1 \delta u_1, ..., \delta x_n \delta u_m}_{n \times m}]^T.$$

This corresponds to a regression with $(1 + m + n + \frac{n(n+1)}{2} + \frac{m(m+1)}{2} + m \times n)$ learnable parameters, which is more than the other approaches where only the linear dynamics is learned ($(1 + m + n)$ learnable parameters).

The higher dimensionality of the cost regression requires a higher number of samples for equal approximation precision, which can be an issue when data generation is expansive. However, all the samples found during the dynamics exploration phase can also be used for the quadratic regression if the cost is recorded at the same time than the new state observed. The efficiency of the methods in Figures 7.2c and 7.2d are compared in Section 7.3.

7.2.3 Update the controller

7.2.3.1 Compute the cost-to-go

Once the Taylor expansions of the cost and dynamics functions have been estimated, the next step is to update the controller to get lower cost over the whole trajectory. To do so, we need to leverage the state-value (V) and action-value (Q) functions, which are defined as follows:

$$Q_t^{\Pi}(x_t, u_t) = \mathbb{E}_{\Pi}\left[\sum_{j=t}^{T} l_j\right] = L_t(x_t, u_t) + \mathbb{E}_{\Pi}\left[\sum_{j=t+1}^{T} l_j\right], \tag{7.8}$$

$$V_t^{\Pi}(x_t) = \mathbb{E}_{\Pi}\left[\sum_{j=t}^{T} l_j\right] = \mathbb{E}_{\Pi}\left[L_t(x_t, \Pi_t(x_t)) + \sum_{j=t+1}^{T} l_j\right], \tag{7.9}$$

where $\Pi = \{\Pi_t,\ t \in \{1, ..., T-1\}\}$ denotes the time-varying linear controller. In other words, Q_t^{Π} represents the expected cost until the end of the trajectory if following Π after being in state x_t and selecting action u_t. The state-value function V_t^{Π} is the same but conditioned only on x_t. If Π is deterministic, these two functions are strictly equivalent. To simplify the notations, from now on, we remove the Π exponents of Q and V. Under the assumption that the trajectories are Markov Decision Process (MDP), (7.8) and (7.9) can be reformulated as:

$$Q_t(x_t, u_t) = l_t + V_{t+1}(x_{t+1}), \tag{7.10}$$

$$V_t(x_t) = Q_t(x_t, \Pi_t(x_t)). \tag{7.11}$$

To lighten the derivation of the updated controller, we also define an intermediate controller that enables us to compute improvements on the time-varying linear controller Π during each improvement step:

$$\delta u_t = \pi(\delta x_t) = P_t \delta x_t + p_t. \tag{7.12}$$

The P_t term improves the controller response to state variations and the p_t term is a shift in the nominal trajectory.

To compute an improved controller, we first compute quadratic Taylor expansions of both value functions:

$$Q_t(\bar{x}_t + \delta x_t, \bar{u}_t + \delta u_t) \approx \; Q_{0_t} + [Q_{x_t}, Q_{u_t}] \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} + \tfrac{1}{2} \begin{bmatrix} \delta x_t^T, \delta u_t^T \end{bmatrix} \begin{bmatrix} Q_{x,x_t} & Q_{x,u_t} \\ Q_{u,x_t} & Q_{u,u_t} \end{bmatrix} \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}, \tag{7.13}$$

$$V_t^{\Pi}(\bar{x}_t + \delta x_t) \approx V_{0_t} + V_{x_t} \delta x_t + \frac{1}{2} \delta x_t^T V_{x,x_t} \delta x_t. \tag{7.14}$$

These quadratic approximation are computed with a backward recursive method and Q_t can be expressed in terms of V_{t+1} by plugging (7.5), (7.6) and (7.14) into (7.10). This leads to the following expressions for the coefficients of (7.13):

$$\begin{aligned}
Q_{0_t} &= \bar{l}_t + V_{0_{t+1}} + V_{x_{t+1}} \bar{x}_{t+1} + \frac{1}{2} \bar{x}_{t+1}^T V_{x,x_{t+1}} \bar{x}_{t+1}, \\
Q_{x_t} &= L_{x_t} + V_{x_{t+1}} F_{x_t} + \bar{x}_{t+1}^T V_{x,x_{t+1}} F_{x_t}, \\
Q_{x,x_t} &= L_{x,x_t} + F_{x_t}^T V_{x,x_{t+1}} F_{x_t}, \\
Q_{u_t} &= L_{u_t} + V_{x_{t+1}} F_{u_t} + \bar{x}_{t+1}^T V_{x,x_{t+1}} F_{u_t}, \\
Q_{u,u_t} &= L_{u,u_t} + F_{u_t}^T V_{x,x_{t+1}} F_{u_t}, \\
Q_{x,u_t} &= L_{x,u_t} + F_{x_t}^T V_{x,x_{t+1}} F_{u_t}.
\end{aligned} \tag{7.15}$$

The coefficients of (7.14) are then obtained using (7.12), (7.13) and (7.11):

$$\begin{aligned}
V_{0_t} &= Q_{0_t} + Q_{u_t} p_t + \frac{1}{2} p_t^T Q_{u,u_t} p_t, \\
V_{x_t} &= Q_{x_t} + Q_{u_t} P_t + p_t^T Q_{u,u_t} P_t + p_t^T Q_{x,u_t}^T, \\
V_{x,x_t} &= Q_{x,x_t} + P_t^T Q_{u,u_t} P_t + Q_{x,u_t} P_t.
\end{aligned} \tag{7.16}$$

Once all the value functions have been computed backward with initial condition $V_T = L_T(x_T)$, the controller with lowest cost-to-go at each time step needs to be computed.

7.2.3.2 First order controller update

In order to improve the controller, δu_t that minimizes Q_t needs to be computed at each time step t. Hence, we start by isolating the terms of Q_t depending on δu_t in (7.13), which gives the following quadratic function to minimize:

$$f(\delta x_t, \delta u_t) = (Q_{u_t} + \delta x_t^T Q_{x,u_t})\delta u_t + \frac{1}{2}\delta u_t^T Q_{u,u_t}\delta u_t. \tag{7.17}$$

Then, we derive this function with respect to δu_t:

$$\frac{\partial f}{\partial \delta u_t}(.) = Q_{u_t} + \delta x_t^T Q_{x,u_t} + \delta u_t^T Q_{u,u_t}. \tag{7.18}$$

Note that this partial derivative is in the Jacobian form (row wise).

To optimize the controller, several methods (first and second order) have been applied ([Todorov and Li, 2005]). In Section 7.2.3, we present an efficient second order technique, validated in Sections 7.4 and 7.5. However, to ease the characterization of the method, this section introduces a first order method, which is more intuitive. The gradient of f with respect to δu_t is

$$\nabla_{\delta u_t} f(\delta x_t, 0) = Q_{u_t}^T + Q_{x,u_t}^T \delta x_t. \tag{7.19}$$

And steps towards better controllers are made in the opposite direction of the gradient. Hence, for the same input δx_t, the new chosen δu_t should be

$$\delta u_t = -\epsilon(Q_{u_t}^T + Q_{x,u_t}^T \delta x_t). \tag{7.20}$$

Obviously, the model being only valid locally, the step ϵ should be chosen small enough not to violate a "proximity to nominal" constraint.

In practice, we have implemented a slightly different method, similar to super-SAB ([Allard and Faubert, 2004]). The variance for exploration (see Section 7.3) is modified according to the evolution of the partial derivatives signs, i.e. it is increased if the sign remains the same and decreased otherwise. Then, ϵ is tuned to stay inside the variance boundaries. In this way, as we get closer to the optimal solution, the exploration gets tighter and the improvements are more accurate. This controller improvement method appear to converge within a reasonable number of steps for positioning tasks (Section 7.3).

Then, the controller can be updated to produce new commands:

$$u_t = \widetilde{u}_t + \delta u_t, \tag{7.21}$$

where $\widetilde{u}_t$ is the action chosen if following the current controller $\widetilde{\Pi} = (\widetilde{K}_t, \widetilde{k}_t)$. By definition of the current controller, we have

$$\widetilde{u}_t = \widetilde{K}_t x_t + \widetilde{k}_t, \tag{7.22}$$

where x_t is a slight deviation from the nominal trajectory:

$$x_t = \bar{x}_t + \delta x_t. \tag{7.23}$$

By plugging (7.12), (7.22) and (7.23) into (7.21), we obtain:

$$u_t = \widetilde{K}_t x_t + \widetilde{k}_t + P_t \delta x_t + p_t, \tag{7.24}$$

Finally, by adding and subtracting $P_t \bar{x}_t$ to the equation above, the global controller is updated as follows:

$$K_t = \widetilde{K}_t + P_t, \tag{7.25}$$

$$k_t = \widetilde{k}_t + p_t - P_t \bar{x}_t, \tag{7.26}$$

where the values of P_t and p_t can be found by identification between (7.20) and (7.12):

$$P_t = -\epsilon Q^T_{x,u_t}, \tag{7.27}$$

$$p_t = -\epsilon Q^T_{u_t}. \tag{7.28}$$

The new average trajectory is obtained by applying this new controller to the initial state.

7.2.3.3 Second order controller update

Another method, more efficient but more complex, can be implemented to update the controller. Under the quadratic value functions (7.13) and (7.14), it can be shown, by cancelling the derivative (7.18), that the optimal controller under such dynamics and cost is defined by

$$\begin{aligned} K_t &= -Q^{-1}_{u,u_t} Q_{u,x_t}, \\ k_t &= \bar{u}_t - Q^{-1}_{u,u_t} Q_{u_t} - K_t \bar{x}_t. \end{aligned} \tag{7.29}$$

A criterion to compute the new covariance is also needed. The goal being to explore the environment, we follow ([Levine and Koltun, 2013]) and choose the covariance with highest statistical entropy in order to maximize information gained during exploration. Such covariance matrix is:

$$\Sigma_t = Q_{u,u_t}^{-1}. \tag{7.30}$$

However, (7.13) and (7.14) being only valid locally, we need to limit the deviation from the nominal trajectory or the environment might respond completely differently and we might fall into bad behaviors. Such issue is solved in the next section by introducing a proximity constraints on the trajectories, thus making the choice of the initial covariance matrix of major importance. Indeed, it influences the exploration range for all the future steps: if it has large values, next iteration also needs to have large covariance. In our implementation, we start with diagonal covariance matrices (for all $t \in \{0, ...T-1\}$) where all the diagonal entries are the same. The choice of such diagonal entries, denoted σ_{ini}, is studied experimentally for the task of Cartesian positioning in Section 7.4.

7.2.3.4 Limit the deviation from nominal trajectory

The controller derived above is optimal only if the dynamics and cost are respectively linear and quadratic everywhere. The approximations being only valid locally, the controller needs to stay close from the nominal trajectory after update. This problem can be solved by adding a constraint to the cost minimization problem:

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) \leq \epsilon, \tag{7.31}$$

where D_{KL} is the statistical Kullback-Leibler divergence. $\widetilde{\rho}(\tau)$ and $\rho(\tau)$ are the trajectories probability distributions under the current controller and the updated one, and ϵ is a user defined parameter.

In ([Levine et al., 2015b]), it is shown that such constrained optimization problem can be solved rather easily by introducing the modified cost function:

$$L_{\text{mod}_t}(x_t, u_t) = \frac{1}{\eta} L_t(x_t, u_t) - \log(\widetilde{\Pi}_t(x_t, u_t)), \tag{7.32}$$

where $\widetilde{\Pi}$ is the current controller (before update) and η is also user defined. Indeed, using dual gradient descent, we can find a solution to the constrained problem by alternating between the two following steps:

- Compute the optimal unconstrained controller under L_{mod} for a given η
- If the controller does not satisfy (7.31), increase η.

A large η has the effect of increasing the importance on constraint satisfaction, so the larger η is, the closer the new trajectory distribution will be from the previous one.

To evaluate (7.31), one first needs to compute $D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau))$. In the appendix of ([Montgomery and Levine, 2016]), the authors show that

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = \sum_{t=0}^{T} \mathbb{E}_{\Pi_t(x_t)} \left[D_{KL}(\Pi_t(u_t|x_t)||\widetilde{\Pi}_t(u_t|x_t)) \right]. \tag{7.33}$$

In the context of the time-varying linear-Gaussian policies, we show that this formula can be written as follows:

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = \frac{1}{2} \sum_{t=0}^{T} \mathrm{Tr}(A_t \Sigma_{x_t}) + \mu_{x_t}^T A_t \mu_{x_t} + 2\mu_{x_t}^T b_t + c_t, \tag{7.34}$$

where A_t, b_t, c_t, μ_{x_t} and Σ_{x_t} (the mean and covariance of the state distribution at time step t) are computed recursively from the initial state distribution, the Taylor expansions of the dynamics and the expression of the controller. A complete proof as well as the exact recursive formulas for A_t, b_t, c_t, μ_{x_t} and Σ_{x_t} are given in Appendix F.

7.2.3.5 Initialize η and choose ϵ

The way Q is defined from approximation does not guarantee positive definiteness for Q_{u,u_t}, i.e. it might not be eligible to be a covariance matrix. This issue is addressed by increasing η such that the distribution is close enough from the previous one. As the previous trajectory has a positive definite covariance, there exists a η that enforces positive definiteness. This gives a good way to initialize η for a given pass.

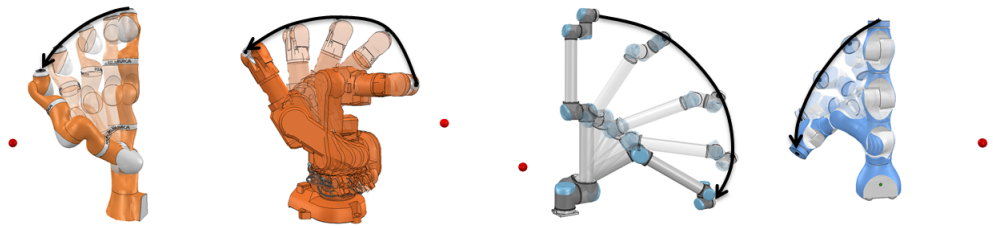
Finally, the choice of ϵ is also paramount. If it is too small, the controller sequence will not progress towards optimality and if it is too large, it might be unstable. The idea is to start with a certain ϵ_{ini} and decrease it if the new accepted controller is worse than the previous one. The choice of ϵ_{ini} is studied experimentally for the task of Cartesian positioning in Section 7.4.

7.3 Simulation validation of quadratic regression for cost computation

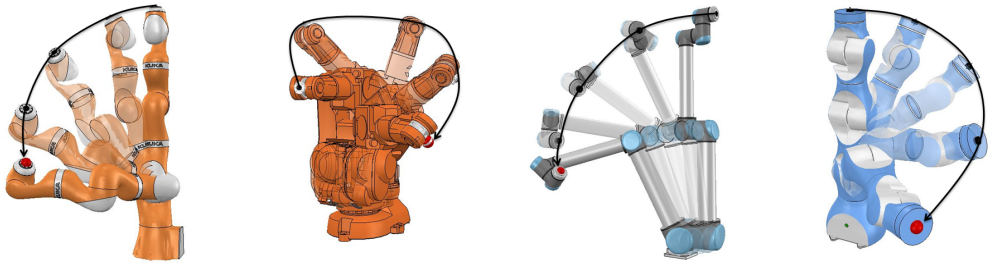
In order to validate the quadratic regression method to compute the cost, we start by implementing the simple gradient descent update on the V-REP software.

7.3.1 Problem definition

In order to show the effectiveness of the method, a robot manipulator positioning task has been implemented using the V-REP simulation software. The robot is controlled in angular position, the state vector is the vector of angular joint coordinates and the control vector is the desired angular positions for the next step. The goal is to reach a target Cartesian point with the end-effector of the robot. The cost function is expressed naturally as the Cartesian distance between the connector and the target. Figure 7.3 represents the initial and final trajectories for the simulated positioning tasks. On this figure, the end-effector is represented by the little black connector at the extremity of the robots and the target by the center of the red floating sphere. The distance (cost function) is extracted directly from the software. In real physical applications, one can imagine it to be provided by any distance measurement sensor (e.g. laser tracker).



(a) Initial, randomly chosen, nominal trajectory



(b) Final trajectory found

Figure 7.3: Trajectory learned in simulation for positioning task for different industrial robots (KUKA LBR iiwa / ABB IRB 140 / UR 10 / F&P P-ARM, from left to right).

If one want to solve such control problem with classical methods, he first needs to build a model of the robot (e.g. DH), giving the end-effector position w.r.t. joint coordinates. Indeed, to be able to get the quadratic approximation of the cost, one needs to map analytically the joint space to the Cartesian space in order to differentiate it. Obviously, no such model has been derived for the simulation and

we implemented the algorithm on four industrial robots with different specifications (number of links, lengths of links, ...) to illustrate that the method is model-free (Figure 7.3). Testing any additional new robot would take only a few additional minutes of work as there is no need to explicit a DH-model. Thanks to this method, we are free of modelling imprecision and only sensor precision is involved in the learning process.

It is also worth underlining that the task would be similar if fixing a reference frame to the end-effector and adding orientation constraint. All that is needed is to take distances between three points instead of only one.

7.3.2 Trajectory found on different robots

The nominal trajectory used to initialize the iLQG algorithm was chosen randomly. We chose to set all desired joint positions to 60° and explore around this initial command. Figure 7.3a represents the initial nominal trajectory. Note that the angular point-to-point (PTP) movement between two joints configuration is considered as a black box, part of the V-REP environment. In this way, the dynamics can really be considered unknown.

Figure 7.3b represents final trajectories found for the positioning task. The algorithm stopped running when a precision of 0.1 mm was reached. As it can be seen, iLQG with no model of the robot and Cartesian cost function succeeds to reach its objective point with different robots. Indeed, they have different lengths, different numbers of links, but the algorithm can still converge towards a solution without being restricted on the cost function definition.

7.3.3 Number of samples needed

Figure 7.4 shows the influence of the number of samples on the rate of convergence on one specific robot (KUKA LBR iiwa). For a sample count above 40, the solution does not evolve anymore, which means that 40 samples is enough to characterize with high precision the local cost function.

On the other hand, Figure 7.4 also shows that if the number of samples gathered is too small (e.g. 5), the cost and dynamics will not be approximated well and the algorithm might not converge towards the desired solution.

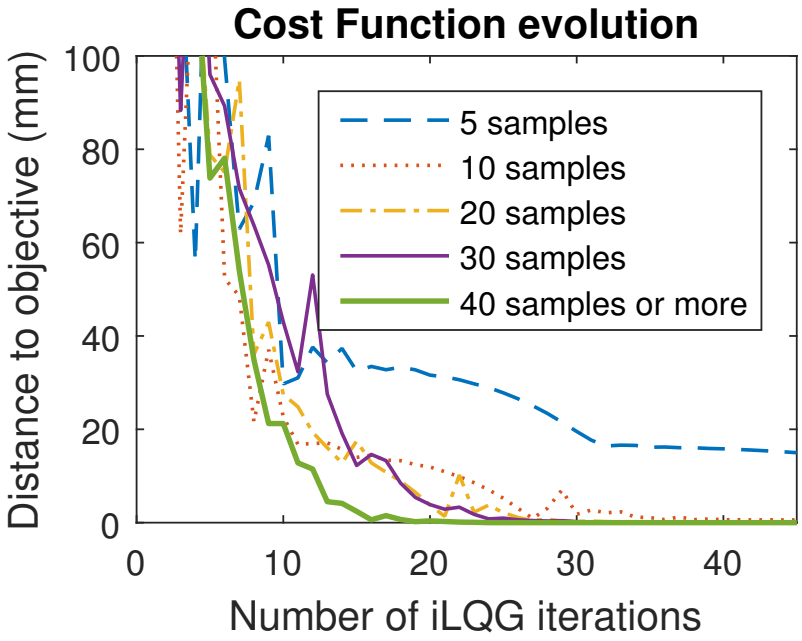


Figure 7.4: Distance to target point with respect to the number of iLQG passes. Different curves represent different number of samples generated for learning the cost and dynamics.

7.3.4 Comparison with state modification method

As mentioned in Section 7.2, another method to avoid robot modeling consists in including the Cartesian distance in the state vector. This technique corresponds to diagram 7.2c of Figure 7.2. In this section, the proposed quadratic regression method is compared with this common technique. Figure 7.5 illustrates the learning curves for both methods under different learning sample sizes. We acknowledge that state modification is less stable. Indeed, in this simulation, results with 30 samples are better than the ones with 40, which does not happen using quadratic approximation. Moreover, the state modification does not show the nice property of being independent of the number of samples when this number becomes high (< 40). Finally, the quadratic regression appears to be more efficient in the final stages of the algorithm, which might come from the fact that a better model of the cost is required to reach high precision in the positioning.

7.4 Parameters tuning for second order methods

Now that we have seen that the method is working and overall better than the other model free method, we try to implement a more complex and elaborated controller

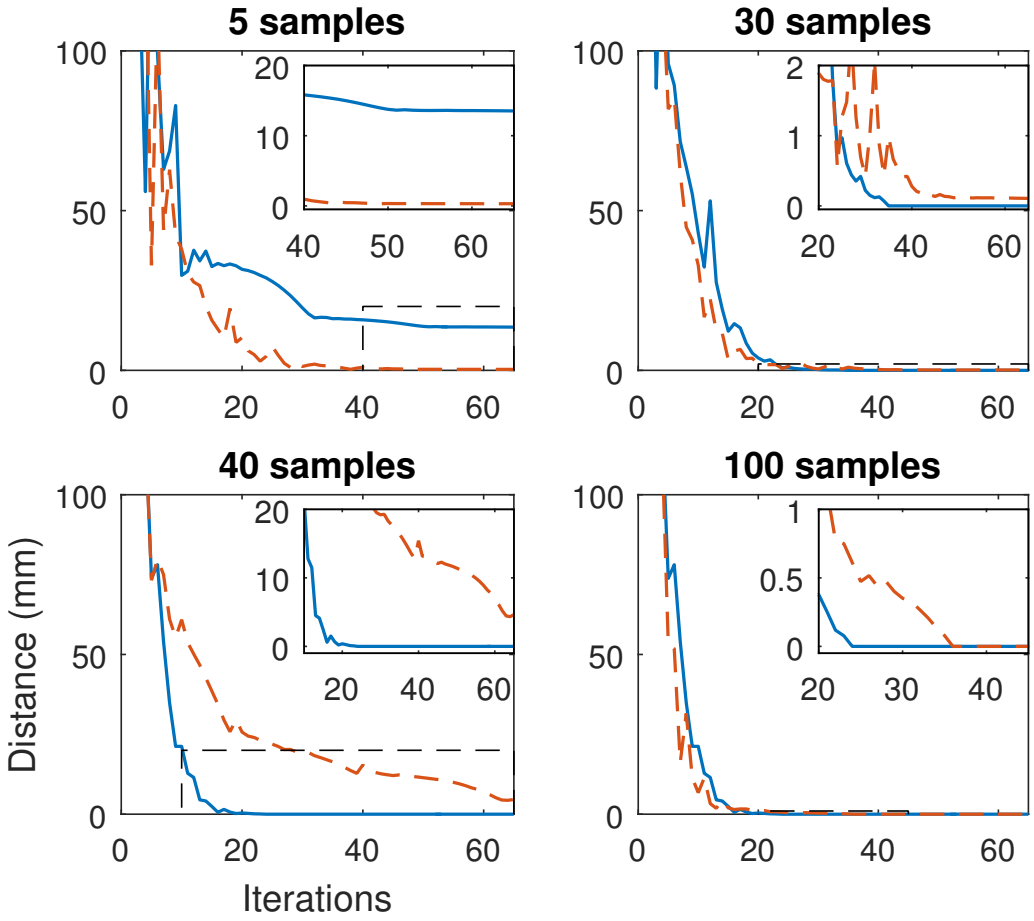


Figure 7.5: Comparisons of two methods: Quadratic regression (plain blue lines) and modified state (dotted orange lines). Right-upper boxes on each figures are zoom on the final stages of iLQG.

update. We use the second order scheme defined in Section 7.2.3.3 and a more complex cost function in order to boost the learning of the positioning task. Such inverse kinematics task is only carried out with the KUKA LBR iiwa robot Figure 7.6.

7.4.1 Cost function

For this problem, the cost function needs to be expressed in terms of the Cartesian distance between the end-effector and the target point. We chose the more elaborated cost function proposed in ([Levine et al., 2015b]):

$$l(d) = d^2 + v \log(d^2 + \alpha), \tag{7.35}$$

where v and α are both real user defined parameters. The squared distance term encourages fast convergence to the neighborhoods of the target and the log term

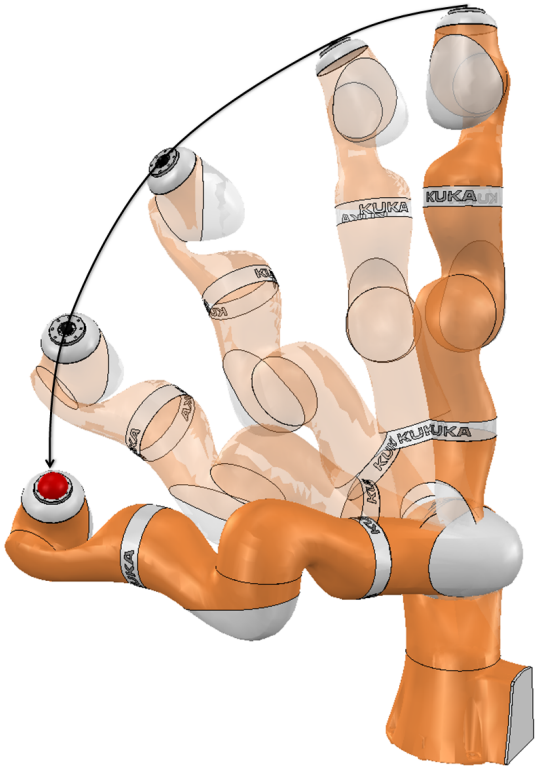


Figure 7.6: Trajectory learnt on V-REP software with a KUKA LBR iiwa.

encourages precise positioning. Hence, v is a trade-off parameter between the two penalties and α is a small positive value that ensures numerical stability. As we do not consider any geometric parameter of the robot, the distance cannot be obtained with direct model considerations and needs to be measured from sensors.

7.4.2 Tune the algorithm parameters

Section 7.3 shows that a number of samples around 40 is a good balance between accurate quadratic regression and exploration time for 7 d.o.f. robots. So we carry out our experiments with $N = 40$. Then, the learning process depends on four parameters, which are studied experimentally in this section:

- the initial covariance, σ_{ini}, which corresponds to the diagonal entries of the covariance matrices of the initial time-varying controller (Section 7.2.3.3),
- the v and α terms from the cost function (Equation 7.35),
- the initial upper bound for the KL divergence between the current and updated trajectories, ϵ_{ini} (Section 7.2.3.5).

Different values of these parameters are tested in simulation using V-REP on the following positioning task:

- Initial position : All 7 angles at 0 (straight position on Figure 7.6)
- Target position : Cartesian vector $[500, 500, 500]^T$ in mm, in the robot frame (red sphere on Figure 7.6)
- Initial mean command : target angular positions = initial positions (no move command).

Figure 7.6 shows a trajectory found by the algorithm.

7.4.3 Results and analysis

After a preliminary study, we choose three values for each parameter and all the 81 possible combinations are tried to choose a good set of parameters for positioning tasks. Results obtained are summarized in Table 7.1. In our simulation, the robot was allowed only 16 trials to reach a precision of 0.1 mm. Thus, we insist that in Table 7.1, an underlined number represents the number of iLQG iterations before convergence whereas other numbers are the remaining distance to objective after 16 iterations.

Table 7.1: Influence of the four user defined parameters on the convergence of the method.

$\sigma_{\text{ini}} = 1$

v	ϵ_{ini}	α 10^{-3}	10^{-5}	10^{-7}
0.1	100	11	16	13
	1000	0.25	12	10
	10000	13	0.27	8
1	100	0.11	14	16
	1000	10	12	10
	10000	0.10	1.69	0.24
10	100	0.11	0.22	0.84
	1000	0.13	12	0.20
	10000	13	0.23	15

$\sigma_{\text{ini}} = 10$

v	ϵ_{ini}	α 10^{-3}	10^{-5}	10^{-7}
0.1	100	0.32	0.15	0.39
	1000	0.45	0.28	0.22
	10000	0.30	0.29	0.31
1	100	0.14	0.32	0.32
	1000	14	1.93	1.70
	10000	1.82	0.99	0.11
10	100	0.34	0.38	0.39
	1000	0.71	0.29	0.53
	10000	0.70	0.14	2.31

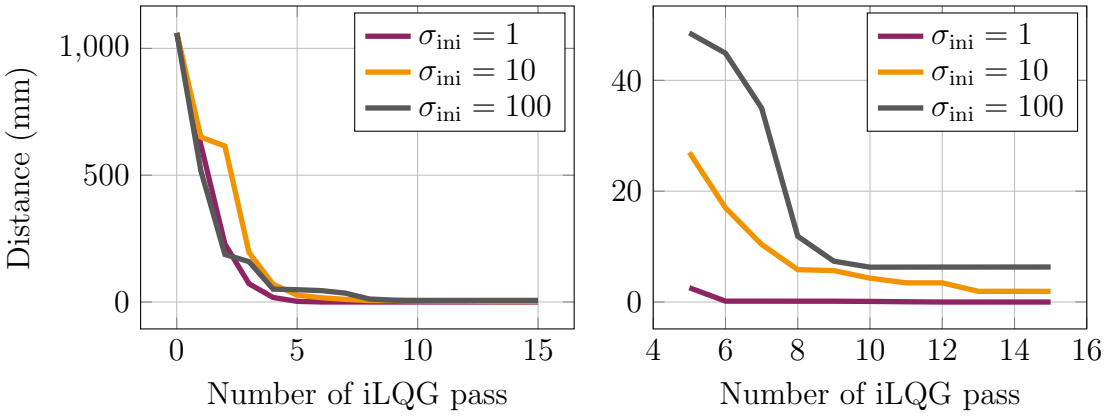
$\sigma_{\text{ini}} = 100$

v	ϵ_{ini}	α 10^{-3}	10^{-5}	10^{-7}
0.1	100	12.79	12.42	17.83
	1000	4.42	0.30	3.50
	10000	2.88	10.93	2.60
1	100	24.37	15.75	10.13
	1000	7.66	6.32	1.87
	10000	2.67	8.37	6.44
10	100	1.93	8.93	10^{11}
	1000	8.03	2.23	3.50
	10000	2.70	4.83	2.60

An underlined number represents the number of iLQG iterations to reach 0.1*mm precision, Other numbers represent the distance remaining after* 16 *iterations.*

Results in Table 7.1 suggest that the best set of parameters for the positioning task is $\sigma_{\text{ini}} = 1$, $v = 0.1$, $\alpha = 10^{-7}$ and $\epsilon_{\text{ini}} = 10000$. Indeed, these parameters have the smallest number of iterations and are thus chosen for the experimental validation in section 7.5.

Together with the raw data in Table 7.1, we also plot the evolution of the distance within the iterations of a simulation for several sets of parameters. Looking at Table 7.1, it seems that the most critical parameter is σ_{ini}. Figure 7.7 shows three learning curves where only σ_{ini} varies, all the other parameters are set to their median value. From here it appears that the initial covariance is not crucial in the early stages of the learning process. However, the right plot, which is a zoom on the final steps, reminds that if the covariance is too large, the algorithm will not converge towards the desired accuracy behavior. Hence, as already suggested by Table 7.1, keep σ_{ini} around 1 seems to be a good choice to obtain the desired accurate behavior.



(a) Full plot **(b)** Zoom on the final iterations

Figure 7.7: Influence of the initial covariance on the learning speed with other parameters set to $v = 1$, $\alpha = 10^{-5}$, $\epsilon_{\text{ini}} = 1000$

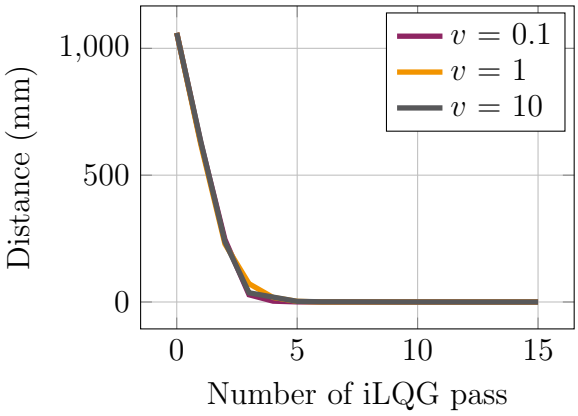
After setting σ_{ini} to 1, we draw the same plots for the other parameters in Figure 7.8. These reveal that v and α do not appear to influence the behavior in this range of values. However, looking at Figure 7.8c, we can see that ϵ_{ini} needs to be kept large enough such that an iLQG iteration can make enough progress towards optimality. For small ϵ_{ini}, convergence is slower near the initial configuration.

7.5 Experimental validation for the method

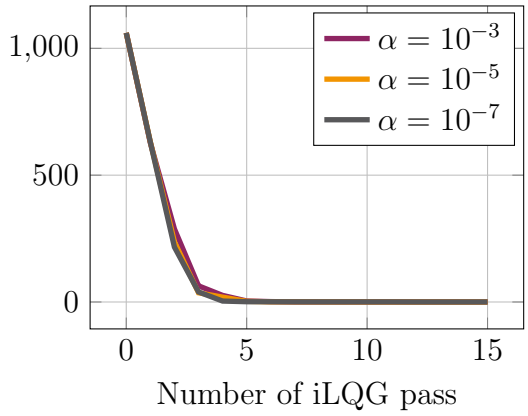
7.5.1 Validity of simulation parameters for a real-world implementation

Experimental setup

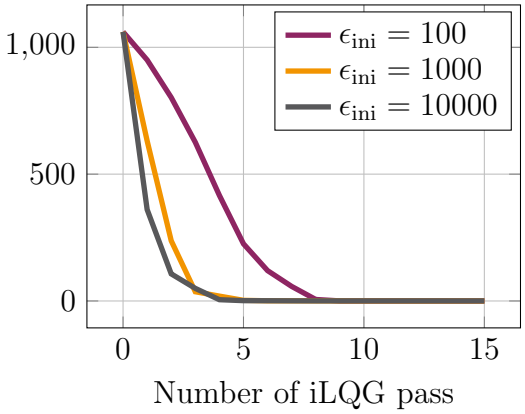
In this section, we evaluate how well the parameters chosen in simulation (Section 7.4) transfer to real-world. Hence, our algorithm is implemented on a real KUKA LBR



(a) Influence of the v term in the cost function



(b) Influence of the α term in the cost function



(c) Influence of the initial ϵ

Figure 7.8: Influence of other parameters on learning speed with $\sigma_{\text{ini}} = 1$. When they do not vary, other parameters take the following values: $v = 1$, $\alpha = 10^{-5}$, $\epsilon_{\text{ini}} = 1000$

iiwa for a similar positioning task under different initial and final configurations, which are shown on Figure 7.9:

- Initial pose : $[140, 0, 0, 0, 0, 0, 0]^T$, angular positions in ° (Figure 7.9a),
- Target position : $[-600, 400, 750]^T$, Cartesian position in mm expressed in the robot frame (Position of the end effector on Figure 7.9b),
- Initial mean command : target angular pose = initial pose (no move).

The choice of changing the initial configuration is motivated by two reasons:

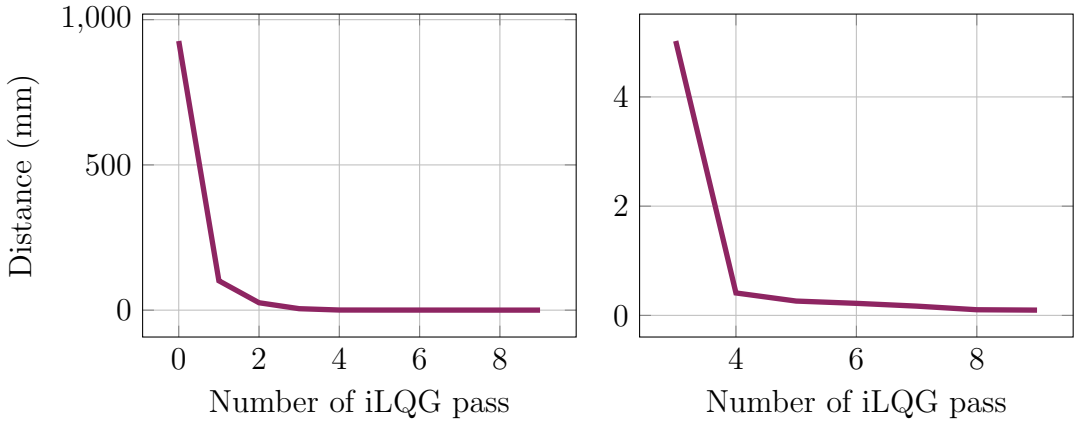
- It shows that the parameters found in Section 7.4 are not case dependent,
- This setup reduces the risk of collision with the working environment.

(a) Initial configuration. **(b)** Final pose learned.

Figure 7.9: Initial configuration and final pose learned for the Cartesian positioning task with the KUKA LBR iiwa.

Results

The learning process defined above resulted in the learning curve on Figure 7.10. We note that it takes as many steps to go from the initial configuration to 1 mm accuracy than from 1 mm to 0.1 mm. The final command, i.e. target angular configuration, provided by the algorithm is $[144.266, 25.351, 2.328, -56.812, 5.385, 24.984, 4.754]^T$. Regarding the learning time, the overall process took approximately 9 minutes, 6 for exploration and 3 for calculations.



(a) Full plot **(b)** Zoom on the final iterations

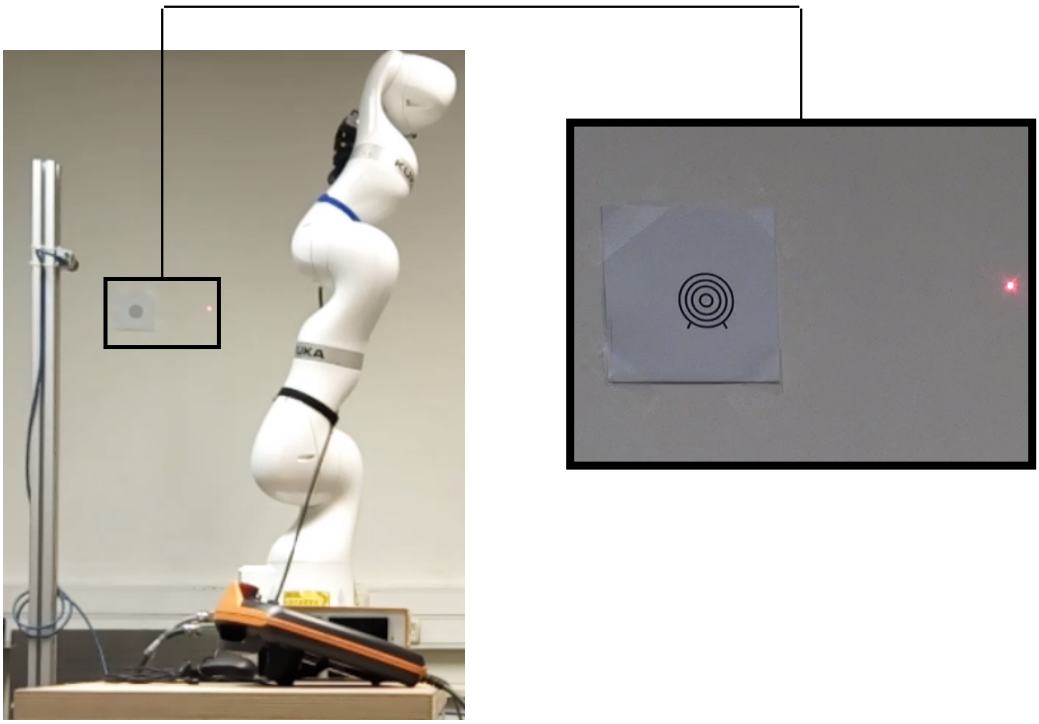
Figure 7.10: Learning curve for iLQG on the robot.

On this experimental validation, the distance is computed from the end-effector position read from the robot internal sensors. Even if it was probably computed thanks to a direct DH model, our algorithm uses it as a black box sensor measurement. Thus, similar results would have been obtained using any other distance measurement sensor (e.g. laser tracker) instead of using the internal robot variables. We just note that, the precision reached is relative to the precision of the measurement tool.

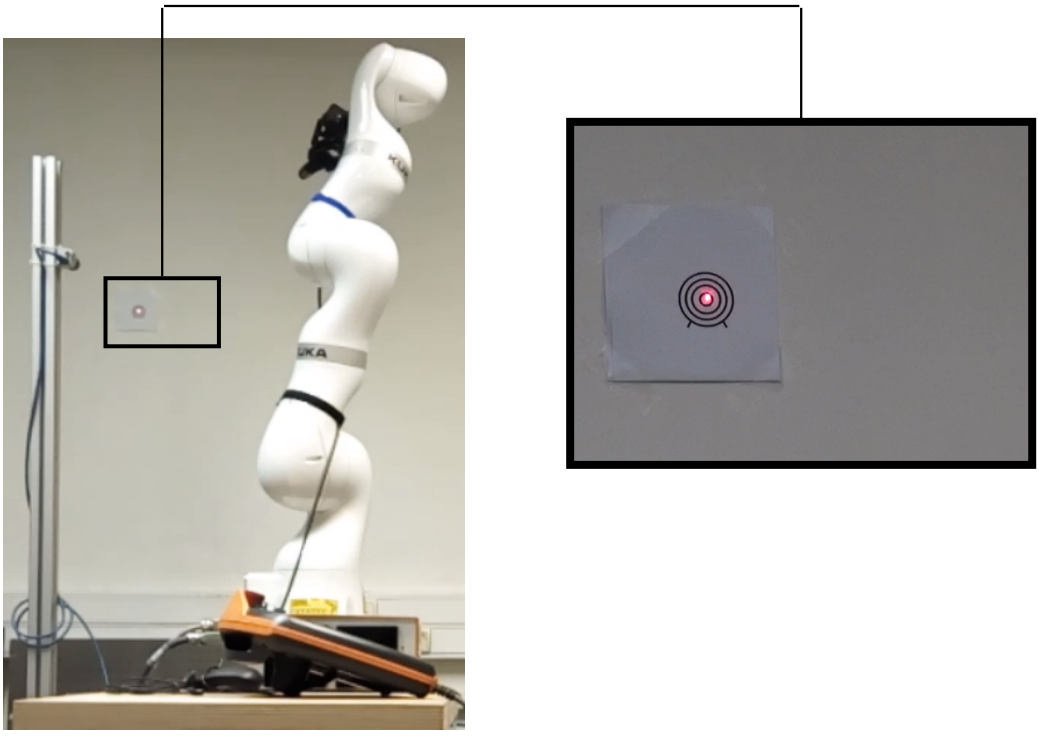
7.5.2 Qualitative validation on a case where no DH model is available

A qualitative validation on a case for which the direct geometric model of the system cannot be computed is proposed in this section. The robot, equipped with a two-finger gripper, is positioned in a room at a random and unknown location. Then, a laser pointer is placed in the gripper in a random orientation and the end-effector is defined by the projection of the laser on a given white wall of the room. A target is stuck on this wall and the objective is to find a robot configuration to reach its center. A camera continuously streams the distance (in pixels) between the red dot and the center of the target thanks to a simple computer vision pipeline. This distance information is used to compute the cost function and provide feedback to the robot. The different elements described above and the initial setup for this tasks can be seen in Figure G.29a.

For this task, the mapping from joint positions to end effector positions cannot be computed because both the location of the robot and the orientation of the laser pointer are unknown. Hence, this example is a good case study to test iLQG with learned dynamics and cost function. The proposed approach converges towards the center of the target in approximately two minutes at a precision of around 2 mm, which is the precision of our simple external measurement system. The final configuration reached can be seen in figure G.29b and a video of the robot solving the task is available at: `https://www.youtube.com/watch?v=Ekda9q3vv6Y`. This implementation demonstrates qualitatively the interest of the method by solving a task where obtaining a model is really hard and we can see that our approach is not impacted by such absence of model.



(a) Initial configuration



(b) Learned final pose

Figure 7.11: Qualitative validation: Laser pointer target reaching task.

7.6 Conclusion

In this chapter, we have introduced a modified version of iLQG to solve the locally optimal trajectory learning problem. This approach consists in regressing the second order Taylor expansion of the cost function from data measurements. Compared to standard ways of computing the cost function, this method has the advantage of being independent from the model of the system and avoids nested approximations. To validate the proposed method, we have studied experimentally the task of Cartesian positioning for serial robots, without using the forward model. We assume that during learning, the robot can only access its angular joint positions and the Cartesian distance between its end-effector and its target, measured by an external sensor.

The simulated experiments conducted in Section 7.3 demonstrate that learning the quadratic cost function is more stable and converges faster to high precision trajectories than including distance as a state variable. Simulation is also leveraged to learn a good set of hyperparameters for second order controller improvements. These parameters are then shown to work for a real robot for a standard positioning task. The high precision reached for this simple positioning task let us hope that such methods will be suitable for more complex industrial tasks. Finally, we demonstrate the model independence of the approach by solving a target reaching task with a laser pointer, for which a direct model cannot be computed in general. Perspectives and future work regarding trajectory learning are discussed in Chapter 9.

Chapter 8

Automatic Construction of Real-World Datasets for 3D Object Localization using Two Cameras

Abstract

Similarly to object understanding and trajectory learning, object localization is an elementary robotic skill that is use in the conceptual definition of many robotic applications, including robotic sorting. Unlike classification, position labels cannot be assigned manually by humans. For this reason, generating supervision for precise object localization is a hard task. This chapter details a method to create large datasets for 3D object localization, with real world images, using an industrial robot to generate position labels. By knowledge of the geometry of the robot, we are able to automatically synchronize the images of the two cameras and the object 3D position. We applied it to generate a screw-driver localization dataset with stereo images, using an industrial robot. This dataset could then serve to train a CNN regressor to learn end-to-end stereo object localization from a set of two standard uncalibrated cameras.

8.1 Introduction

Like object understanding and trajectory learning, object localization is an elementary brick which can serve in building many robotic applications and can be made more autonomous and easier to "program" by unqualified workers. Hence, this chapter addresses the problem of 3D object localization and introduces a method for automatic generation of large stereo localization datasets. By leveraging the precision and repeatability of industrial robots, the proposed methodology enables to save labor time and increase precision for datasets construction. This chapter is an ongoing research and these datasets should be used for training end-to-end stereo localization networks to validate the methodology. However, we believe that this chapter introduces some interesting concepts about dataset construction that can be useful to the community. We now propose a more specific introduction to the topic of 3D localization using a system of stereo cameras.

In the context of autonomous manipulation, 3D object localization plays an essential role. Stereo vision is one of the most efficient methods for 3D reconstruction and thus is a very useful tool when dealing with robotic manipulation. Indeed, stereo vision has received many attention in research over the past decade in different subfields of autonomous manipulation such as grasping (([Azad et al., 2007]), ([Morales et al., 2006])), contact-reach tasks ([Hudson et al., 2012]), and even playing soccer ([Käppeler et al., 2010]).

However, classical methods of stereo vision are hard to design and require a lot of tuning. They can be inaccurate as they are composed of many steps, which are all potential sources of errors (see Section 8.2). For this reason, we strongly believe that stereo localization would benefit from an end-to-end learning approach, which would simplify the design process and hopefully improve the system accuracy. Recently, several end-to-end approaches have been successful in achieving complex tasks, such as robot manipulation ([Levine et al., 2015a]), self-driving cars ([Bojarski et al., 2016]), speech recognition ([Amodei et al., 2016]) or obstacle avoidance ([Muller et al., 2006]). Even for end-to-end learning of manipulation tasks, which includes object localization, the vision part must be pretrained to locate objects if we want reinforcement learning to be scalable to real life applications ([Levine et al., 2015a]). Hence, this work, which mainly focuses on end-to-end learning of 3D stereo object localization, also has great significance within the wider context of robotic autonomous learning of manipulation tasks.

To train a regression model (e.g. convolutional neural network) to learn stereo localization, we need to generate labeled data for localization, i.e., stereo images of the object to be located together with its position in space. Gathering labels for localization is a hard task. Position labels are hard to get as they cannot be written manually without spending precious time doing very precise measurements for each sample. In this chapter, we introduce an approach for building such a dataset, by using an industrial robot to generate labeled data for 3D stereo localization. The main contribution of this chapter is to describe a procedure to gather a lot of labeled stereo data for automatic object localization. This procedure is applied to generate a dataset for screw drivers 3D localization, composed of more than three thousand samples. To provide a baseline for future research to test regression models for end-to-end stereo localization, this dataset is made publicly available and can be downloaded at `https://goo.gl/stu5UE`.

8.2 Motivations for end-to-end stereo localization

8.2.1 Classical stereo vision methods for 3D localization

The use of stereo vision has a long history in robotics manipulation (([Azad et al., 2007]), ([Hudson et al., 2012])). However, currently, stereo localization consists in stacking different methods that can each be inaccurate. As we can see in Figure 8.1, to implement 3D object localization, it is needed to calibrate both cameras, compute accurate stereo matching, build a computer vision pipeline to identify pixels of interest, triangulate, and measure accurate transformation of frames to project the result in the frame of interest.

More recently, researchers have started to use deep learning for stereo vision (([Luo et al., 2016]), ([Knöbelreiter et al., 2016]), ([Zbontar and LeCun, 2016])). However, they mainly focus on stereo matching, which is only solving part of the problem. This trend can also be seen by looking at the problem proposed with the most famous datasets for stereo vision (([Geiger et al., 2012]), ([Scharstein et al., 2014])). To locate an object in space, one also needs to get the precise pixels in one of the image, which can cause errors, even with perfect stereo matching. Indeed, this problem is close to instance segmentation ([Romera-Paredes and Torr, 2016]) and is not easy. In general, end-to-end learning approaches tend to be better than stacking subsystems as it can correct internal errors. For this reason we introduce such a framework for stereo localization in the next section. This chapter is a first step towards implementing it as

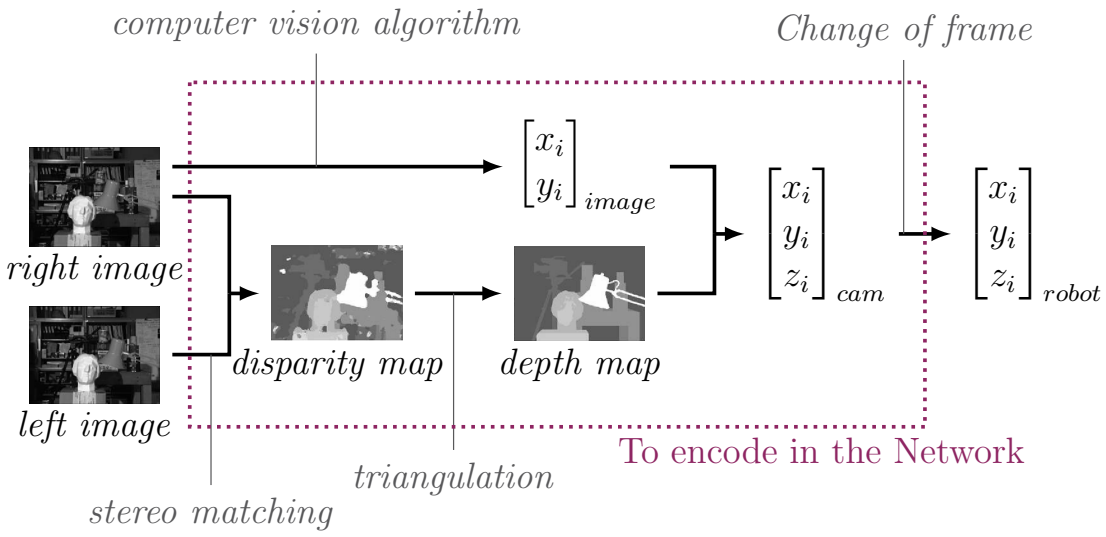


Figure 8.1: Typical pipeline for stereo vision object localization.

it proposes a generic way to produce real-world datasets for stereo object localization using a robot to generate position labels.

8.2.2 End-to-end pipeline for stereo localization

An end-to-end pipeline for stereo object localization would consist in mapping pixels from two images to 3D points in space representing an object pose, using a regression function approximator. Such a pipeline could then be used for grasping, or any robotic manipulation task.

In this chapter, we propose an approach to build datasets for stereo localization learning. Showing the feasibility of gathering large datasets is a first step towards investigating the feasibility of end-to-end object stereo localization. The purple dotted line in Figure 8.1 illustrates what the network is supposed to encode, the relative camera calibration could be added to this frame. After seeing the excellent results produced by CNNs o, various tasks dealing with images, it seems that only the difficulty to gather enough labelled data prevents them to carry out end-to-end stereo localization. The idea implemented in this chapter consists in defining a method, using a precise and accurate industrial robot, to generate labelled data for object localization. We also apply it to the get data for screw-drivers localization. Section 8.3 contains more details about the dataset generation.

8.3 Dataset generation

When building a dataset for supervised learning, it is crucial to make sure that our input data are sufficient to know everything about our outputs. For the case of stereo object localization, all that is required is to have two cameras with fixed focal length and relative positions. We also need to keep the position of the two cameras fixed with respect to the robot and to make sure that both cameras are oriented such that they share partially common fields of view, i.e. the object must be seen by both cameras.

8.3.1 Generate diversity to avoid overfitting

In order to avoid overfitting, the dataset should have as much diversity as possible. To do so, we try to change the following parameters:

- Lighting conditions, by gathering the data in a shop-floor with a glass roof,
- Background, by placing different other objects in the scene.

We also add images where distractors (other tools) are placed in the background. This way, the CNN must learn to locate screwdrivers and not any object. The dataset contains different screwdrivers to help discover the concept of what makes a screwdriver.

Finally, we make sure that the random configurations of the robot explore the full range of positions and orientations allowed within the common view range of the cameras. Figure 8.2 shows a representative subset of the images present in the dataset. Note that each image shown has a corresponding image from the other camera.

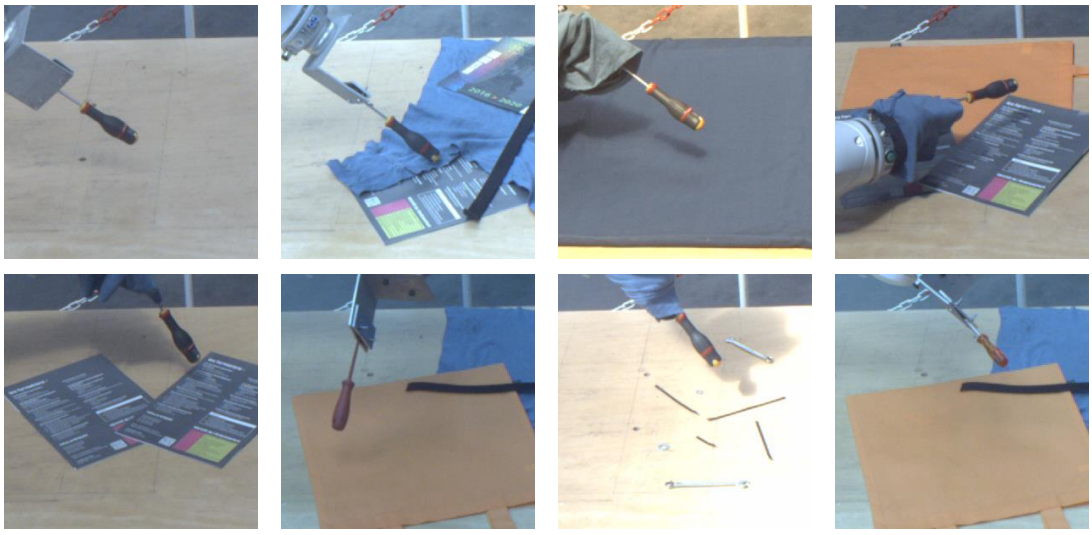


Figure 8.2: Representative subset of the images from the screw driver localization dataset.

8.3.2 Avoid learning the wrong thing by removing the robot

The idea of using a robot to generate supervision opens a broad range of possibilities for object localization. However, the robot, or at least the tool holder, will appear on every image, which is a drawback of the approach. Indeed, we can fear that the regressor learns to locate the robot and just applies some kind of shift to find the object to be located. To avoid such problem, we need to remove the robot from some of the images. We proceed in two different ways:

- Physically, we hide the robot by wrapping some cloth around it (Figure 8.2)
- Computationally, using a computer vision pipeline to remove the robot from images. To do so, we increase the dataset and take four images per sample: robot + tool, robot alone, and the backgrounds corresponding to the two situations. The full pipeline is described in Figure 8.3.

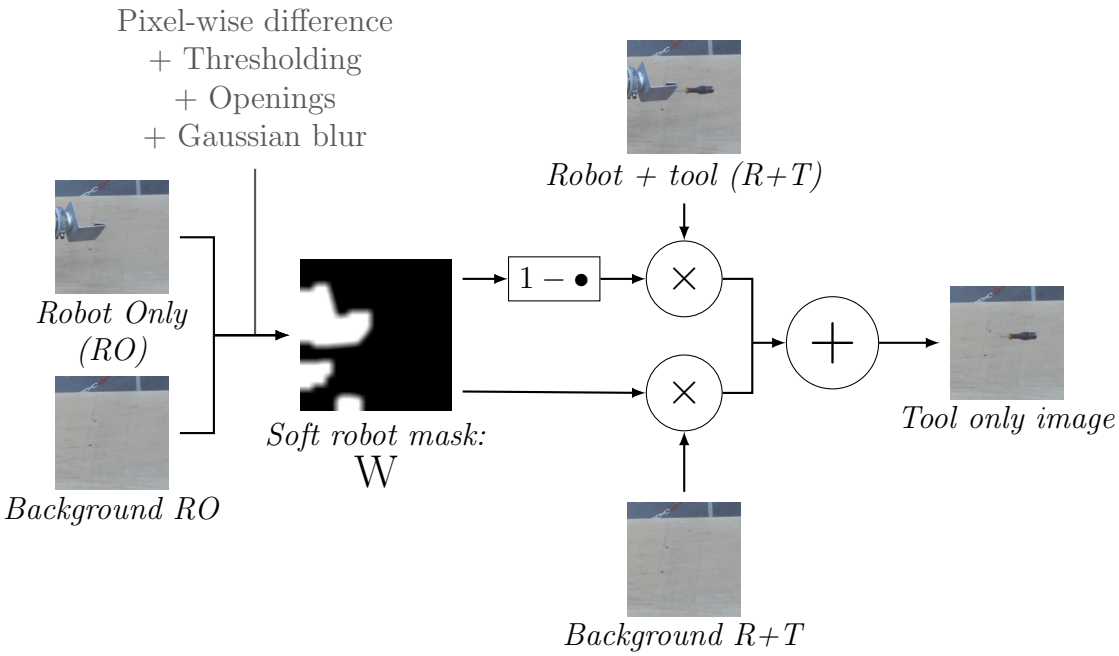


Figure 8.3: Computer vision pipeline using four images to generate an image containing only the studied objects.

The underlying idea behind the robot removal computer vision algorithm is to use the robot only image and the background to compute a mask of the robot and replace the corresponding pixels by background pixels. A similar approach has been proposed by ([Levine et al., 2016]). The necessity to get two background images comes from

the changes in lighting conditions. This algorithm is then fine tuned to remove only the good pixels and get smooth edges.

To ensure that all the images in our dataset do not contain any systemic patterns coming from our robot removal algorithm, it is also important to keep original images (with the robot).

8.3.3 Technical details on data generation

For tool calibration, we assume that the screw driver is orthogonal to the tool holder and we use the very accurate torque sensors of the Kuka LBR iiwa robot to detect when contact is reached with a plaque in several directions. In this way, we can get all the shifts and compute the transformation of frames. This procedure is fully automated and the operator just needs to place the screw driver in the clamp.

When we generate data, we make sure that the object stands within a certain cuboid (the biggest inside the common views), that the robot is not hiding the object in the image, and obviously, that the robot is not colliding with itself or the environment.

We also check, for the cases where we take several images, that the robot is not behind the screw driver so that there is no mask superposition and we do not remove part of the tool with our computer vision algorithm.

8.4 Conclusion

In this chapter, we described a procedure to build a dataset for 3D object localization using stereo vision. By using an industrial robot, we have shown that it is possible to generate data with accurate position labels. We propose an approach to generate data with enough variability, to avoid overfitting, as well as a method to remove the robot from some of the images to prevent our regressor from learning the wrong thing. This methodology is applied to build a dataset for screw driver localization. This dataset can then be used for training a regression model to learn 3D localization from pairs of images coming from our stereo system. Perspectives and future work regarding 3D stereo localization are discussed in Chapter 9.

Part V

Conclusion

Chapter 9

Summary and open questions

Abstract

This final chapter proposes a summary of the contributions developed throughout this manuscript and introduces various research perspectives, which arise naturally from the findings of this thesis.

9.1 Key contributions

Modern industrial robots are flexible machines that can be reprogrammed without mechanical modification and are highly instrumented. These characteristics enable them to successfully achieve a large variety of tasks. However, their current use in industry is still far from the level of autonomy and adaptability that we can expect from such complex systems. In this manuscript, we have proposed several contributions in machine learning to start bridging this gap and to increase robustness of robotic applications to different setups. These contributions are mostly centered around the unsupervised sorting task but have applications in many other domains of robotics.

After introducing the new application of Unsupervised Robotic Sorting (URS), we proposed a first implementation consisting in sorting objects based on their shapes and colors in an unsupervised fashion. This adaptation of a common robotic sorting pipeline to the unsupervised setting lead to the development of a new clustering algorithm that we called Gap-Ratio K-means and that appears useful for noisy datasets containing interval scale data.

Then, to enable our sorting robot to cluster objects with a higher level of abstraction, the problem of image clustering has been studied. The current state-of-the-art methods for complex datasets are using Convolutional Neural Networks (CNN) pretrained on Imagenet to extract features from complex images. For this reason, we first carried out a benchmark study to evaluate how well the different CNN architectures and layers transfer to new unsupervised image datasets. The absence of strong trend in the results motivated the development of an ensemble clustering approach to leverage information from several of such CNNs. This new approach was shown to be state-of-the-art at image clustering on several public datasets.

After developing good image clustering pipelines on standard datasets, we proposed to embed them in practical URS implementations. A first implementation has been created in a shopfloor environment and its robustness to various external factors has been tested. The results from this study suggested that the quality of the sorting is highly dependent on the point of view under which the objects are observed. This finding motivated the development of an optimal view selection method, consisting in creating a large multi-view dataset and use it to train a neural network to choose optimal camera poses. This approach was shown to perform better than fixed and random camera poses on classes of objects that were not seen at training time, which suggests that the network has learned to use geometric features for view selection.

Besides the crucial skill of object understanding, we introduced contributions in two other areas of robotics, which are also essential to improve robustness and adaptability of industrial applications. On the one hand, a standard trajectory learning method was adapted to learn a quadratic model of the cost function instead of computing it analytically. This way, the method becomes independent of the robot's model, thus making the system easier to program in more situations. On the other hand, a method using a robot manipulator to autonomously produce datasets for 3D stereo localization was developed. This technique was used to create one such dataset for screwdriver localization, which was released to the community.

In summary, the different contributions presented in this manuscript address different tasks, which are all important skills for industrial robotics. Furthermore, we would like to emphasize that all the proposed methods have the same underlying purpose: increasing the validity range of the different skills without reprogramming. Indeed,

- unsupervised robotic sorting generalizes standard sorting applications to any set of objects, removing the need to reconfigure the application when the classes change,
- optimal view selection has the potential to make robotic vision systems independent of the observed objects poses,
- the proposed trajectory learning method is generic and does not need to be adapted to the learning agent or its configuration,
- autonomous dataset generation can possibly be an interesting method for developing new specific industrial vision applications with minimal human efforts.

9.2 Directions for future work

9.2.1 Gap-Ratio K-means algorithm

The clustering algorithm proposed in Chapter 2 demonstrated promising results on the unsupervised robotic sorting from color and shape features. Furthermore, the additional experiments carried out in Section 2.4.5 suggest that Gap-Ratio (GR) K-means can also work for datasets such as Iris, that do not present the kind of features specific to the URS implementation. Based on this observation, it would be interesting to investigate further the scope of validity of GR K-means and to try to find a statistical test to choose if GR K-means should be used over other variants.

Another interesting extension of GR K-means would be to combine it with data orthogonalization methods (such as ICA ([Oja and Hyvarinen, 1997])). Indeed, GR weights are computed along different dimensions of the feature space and if features have strong correlation, gaps might disappear and variance might be spread along several dimensions. For this reason, it seems appealing to try to decorrelate data using orthogonalization methods.

Finally, it could also be interesting to consider not only the largest gap along one dimension but also the next ones, according to the number of different classes desired. Indeed, if within three classes the two separations come from the same features, even more importance should be given to this set of features. Some modifications of the equations in Section 2.3 should enable to try such approach.

9.2.2 Image clustering

Experimental results from Chapter 4 illustrate that different CNNs, pretrained on the same task, often contain different and complementary information about a target dataset. Differences may arise from a number of sources including the architecture (number of layers, layer shape, presence of skip connections, etc.), the regularization method, or the loss functions used for training. Investigating which parameters influence knowledge transfer to unsupervised tasks is an interesting axis of research for future work and such knowledge may help to design better CNN architectures.

We also note that pretrained CNNs are used as feature extractors for many applications, not just clustering. Using multiple pretrained CNNs to define a multi-view learning problem may be appealing for other tasks where complementary information present in pretrained feature extractors can improve performance.

Although JULE ([Yang et al., 2016]) appears to be a good algorithm to solve MVC, we acknowledged that it fails for some datasets. Hence, it would be interesting to try different deep clustering methods or to adapt the parameters of JULE for different IC tasks.

Finally, as the number of available pretrained CNNs will keep increasing and many researchers have limited resources, the case where the number of available feature extractors is much larger than the number of available GPUs needs to be considered. Hence, investigating an optimal strategy for architectures selection and resources allocation to maximize the available information is a promising research direction.

9.2.3 Unsupervised robotic sorting

The experiments carried out in Chapter 6 demonstrated the positive impact of semantic view selection on the URS results. Likewise, in Chapter 5, we showed that using multiple viewpoints of an object can help to understand its nature. Hence, a natural perspective would be to extend the semantic view selection problem to the one of multi-view selection. Implementing such a multi-view approach in the formalism of the "next best view selection" literature ([Krainin et al., 2011]) would also allow to detect when the top view is already good, thus avoiding unnecessary computation. The dataset presented in Chapter 6 could be used to train a recurrent neural network to address this problem.

Now that the decision making module of the unsupervised sorting application has been shown to work, it will also be interesting to include it in a more complete pick-and-place pipeline, which handles automatic scene segmentation, objects localization and robotic grasp detection.

9.2.4 Trajectory learning

To further validate the quadratic cost regression approach, several research perspectives can be considered. On the one hand, it would be interesting to use an external measurement tool for the basic positioning task in order to compare our positioning method precision with other techniques. Indeed, the precision of the inverse kinematics of the robot cannot be defined with internal robot measurements. On the other hand, it would also be interesting to study the behavior of the method for more complex manipulation tasks involving contact.

Another directions of improvement of the proposed trajectory learning method is to try to decrease sample complexity. Indeed, the cost quadratic regression has more learnable parameters than the dynamics linear regression and thus requires more examples to produce a valid estimate. This involves additional exploration time. To solve this issue, we could leverage the approach proposed in ([Levine and Abbeel, 2014]), which consists in building a prior on the regressed variables in the form of a Gaussian mixture model. We conducted preliminary experiments with this method and the first results suggest that the overhead exploration cost due to quadratic regression can be highly attenuated with such Bayesian regression approach.

We finally note that the local trajectory learning problem studied in Chapter 7 is an important step in the Guided Policy Search (GPS) algorithm (([Levine et al., 2015b]), ([Levine and Abbeel, 2014])). GPS is a very promising method that can

learn successful visuomotor policies for complex robotic tasks by using learned time-varying linear-Gaussian trajectories to guide the training of a complex neural policy. Hence, embedding our method in the framework of GPS is an interesting direction for future research because the improvements proposed on the trajectory optimization method could improve and accelerate GPS.

9.2.5 Stereo localization

The logical continuation of this work on autonomous datasets generation is to exploit the screwdriver localization dataset that was created and train a CNN architecture to encode the 3D stereo localization pipeline. Another perspective is to apply this process as a preprocessing step for reinforcement learning of manipulation tasks, likewise in ([Levine et al., 2015a]). By doing this, we can expect to get better results with a policy using stereo images instead of a single camera.

Bibliography

[Aljalbout et al., 2018] Aljalbout, E., Golkov, V., Siddiqui, Y., and Cremers, D. (2018). Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*.

[Allard and Faubert, 2004] Allard, R. and Faubert, J. (2004). Neural networks: different problems require different learning rate adaptive methods. In *Proc. SPIE*, volume 5298, pages 516–527.

[Amodei et al., 2016] Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182.

[Arthur and Vassilvitskii, 2007] Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

[Avrithis et al., 2015] Avrithis, Y., Kalantidis, Y., Anagnostopoulos, E., and Emiris, I. Z. (2015). Web-scale image clustering revisited. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1502–1510.

[Azad et al., 2007] Azad, P., Asfour, T., and Dillmann, R. (2007). Stereo-based 6d object localization for grasping with humanoid robot systems. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 919–924. IEEE.

[Bahrin et al., 2016] Bahrin, M. A. K., Othman, M. F., Azli, N. N., and Talib, M. F. (2016). Industry 4.0: A review on industrial automation and robotic. *Jurnal Teknologi*, 78(6-13):137–143.

[Balakirsky et al., 2013] Balakirsky, S., Kootbally, Z., Kramer, T., Pietromartire, A., Schlenoff, C., and Gupta, S. (2013). Knowledge driven robotics for kitting applications. *Robotics and Autonomous Systems*, 61(11):1205–1214.

[Banerjee et al., 2015] Banerjee, A. G., Barnes, A., Kaipa, K. N., Liu, J., Shriyam, S., Shah, N., and Gupta, S. K. (2015). An ontology to enable optimized task partitioning in human-robot collaboration for warehouse kitting operations. In *Next-Generation Robotics II; and Machine Intelligence and Bio-inspired Computation: Theory and Applications IX*, volume 9494, page 94940H. International Society for Optics and Photonics.

[Berkhin, 2006] Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer.

[Bezdek and Hathaway, 2002] Bezdek, J. C. and Hathaway, R. J. (2002). Some notes on alternating optimization. In *AFSS International Conference on Fuzzy Systems*, pages 288–300. Springer.

[Bohg et al., 2014] Bohg, J., Morales, A., Asfour, T., and Kragic, D. (2014). Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309.

[Bojarski et al., 2016] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

[Bonert et al., 2000] Bonert, M., Shu, L., and Benhabib, B. (2000). Motion planning for multi-robot assembly systems. *International Journal of Computer Integrated Manufacturing*, 13(4):301–310.

[Ceci et al., 2015] Ceci, M., Pio, G., Kuzmanovski, V., and Džeroski, S. (2015). Semi-supervised multi-view learning for gene network reconstruction. *PloS one*, 10(12):e0144031.

[Chao et al., 2017] Chao, G., Sun, S., and Bi, J. (2017). A survey on multi-view clustering. *arXiv preprint arXiv:1712.06246*.

[Chen and Burdick, 1995] Chen, I.-M. and Burdick, J. W. (1995). Determining task optimal modular robot assembly configurations. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 1, pages 132–137. IEEE.

[Chen et al., 2009] Chen, X., Yin, W., Tu, P., and Zhang, H. (2009). Weighted k-means algorithm based text clustering. In *Information Engineering and Electronic Commerce, 2009. IEEC'09. International Symposium on*, pages 51–55. IEEE.

[Chen and Dong, 2013] Chen, Y. and Dong, F. (2013). Robot machining: recent development and future research issues. *The International Journal of Advanced Manufacturing Technology*, 66(9-12):1489–1497.

[Chollet, 2015] Chollet, F. (2015). Keras.

[Chollet, 2016] Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*.

[Chua et al., 2017] Chua, K.-W., Bub, D. N., Masson, M. E., and Gauthier, I. (2017). Grasp representations depend on knowledge and attention. *Journal of Experimental Psychology: Learning, Memory, and Cognition*.

[De Brabanter et al., 2013] De Brabanter, K., De Brabanter, J., De Moor, B., and Gijbels, I. (2013). Derivative estimation with local polynomial fitting. *The Journal of Machine Learning Research*, 14(1):281–301.

[de Sousa et al., 2017] de Sousa, G. B., Olabi, A., Palos, J., and Gibaru, O. (2017). 3d metrology using a collaborative robot with a laser triangulation sensor. *Procedia Manufacturing*, 11:132–140.

[Deisenroth et al., 2013] Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142.

[Deisenroth et al., 2011] Deisenroth, M. P., Rasmussen, C. E., and Fox, D. (2011). Learning to control a low-cost manipulator using data-efficient reinforcement learning. *Robotics: Science and Systems 2012*.

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

[Dhanachandra and Chanu, 2017] Dhanachandra, N. and Chanu, Y. J. (2017). A survey on image segmentation methods using clustering techniques. *European Journal of Engineering Research and Science*, 2(1).

[Dombre and Khalil, 2013] Dombre, E. and Khalil, W. (2013). *Robot manipulators: modeling, performance analysis and control*. John Wiley & Sons.

[Dong et al., 2016] Dong, J., Mukadam, M., Dellaert, F., and Boots, B. (2016). Motion planning as probabilistic inference using gaussian processes and factor graphs. In *Robotics: Science and Systems*, volume 12.

[Dragusu et al., 2012] Dragusu, M., Mihalache, A. N., and Solea, R. (2012). Practical applications for robotic arms using image processing. In *System Theory, Control and Computing (ICSTCC), 2012 16th International Conference on*, pages 1–6. IEEE.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). Unsupervised learning and clustering. *Pattern classification*, pages 519–598.

[Dutagaci et al., 2010] Dutagaci, H., Cheung, C. P., and Godil, A. (2010). A benchmark for best view selection of 3d objects. In *Proceedings of the ACM workshop on 3D object retrieval*, pages 45–50. ACM.

[E. Rohmer, 2013] E. Rohmer, S. P. N. Singh, M. F. (2013). V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.

[Ecins et al., 2016] Ecins, A., Fermüller, C., and Aloimonos, Y. (2016). Cluttered scene segmentation using the symmetry constraint. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 2271–2278. IEEE.

[Eitel et al., 2015a] Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., and Burgard, W. (2015a). Multimodal deep learning for robust rgb-d object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 681–687. IEEE.

[Eitel et al., 2015b] Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., and Burgard, W. (2015b). Multimodal deep learning for robust rgb-d object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 681–687. IEEE.

[Elatta et al., 2004] Elatta, A., Gen, L. P., Zhi, F. L., Daoyuan, Y., and Fei, L. (2004). An overview of robot calibration. *Information Technology Journal*, 3(1):74–78.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[Everingham et al., 2007] Everingham, M., Zisserman, A., Williams, C. K., Van Gool, L., Allan, M., Bishop, C. M., Chapelle, O., Dalal, N., Deselaers, T., Dorkó, G., et al. (2007). The pascal visual object classes challenge 2007 (voc2007) results.

[Fisher, 1936] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

[Flickner et al., 1995] Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., et al. (1995). Query by image and video content: The qbic system. *computer*, 28(9):23–32.

[Fowlkes and Mallows, 1983] Fowlkes, E. B. and Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383):553–569.

[Freedman, 2009] Freedman, D. A. (2009). *Statistical models: theory and practice*. cambridge university press.

[Fukui and Wada, 2014] Fukui, T. and Wada, T. (2014). Commonality preserving image-set clustering based on diverse density. In *International Symposium on Visual Computing*, pages 258–269. Springer.

[Gao et al., 2015] Gao, H., Nie, F., Li, X., and Huang, H. (2015). Multi-view subspace clustering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4238–4246.

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Giannoccaro et al., 2013] Giannoccaro, N., Spedicato, L., and Lay-Ekuakille, A. (2013). A smart robotic arm for automatic sorting of objects with different tags. In *4th Imeko TC19 Symposium on Environmental Instrumentation and Measurements Protecting Environment, Climate Changes and Pollution Control, June*, pages 3–4.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

[Goldberger et al., 2006] Goldberger, J., Gordon, S., and Greenspan, H. (2006). Unsupervised image-set clustering using an information theoretic framework. *IEEE transactions on image processing*, 15(2):449–458.

[Gong et al., 2015] Gong, Y., Pawlowski, M., Yang, F., Brandy, L., Bourdev, L., and Fergus, R. (2015). Web scale photo hash clustering on a single machine. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 19–27.

[Groover et al., 1986] Groover, M. P., Weiss, M., Nagel, R. N., and Odrey, N. G. (1986). *Industrial robotics: technology, programming, and applications*, volume 12. McGraw-Hill New York.

[Grunes et al., 1995] Grunes, H., Tepman, A., and Lowrance, R. (1995). Robot assembly. US Patent 5,447,409.

[Guérin and Boots, 2018] Guérin, J. and Boots, B. (2018). Improving image clustering with multiple pretrained cnn feature extractors. In *BMVC 2018 (29th British Machine Vision Conference), Newcastle upon Tyne 3rd-6th September, UK*.

[Guérin et al., 2016] Guérin, J., Gibaru, O., Nyiri, E., and Thiery, S. (2016). Learning local trajectories for high precision robotic tasks: application to kuka lbr iiwa cartesian positioning. In *IECON 2016 (42nd Annual Conference of the IEEE Industrial Electronics Society), October 23th-26th, Florence, Italy*, pages 5316–5321.

[Guérin et al., 2018a] Guérin, J., Gibaru, O., Nyiri, E., Thiery, S., and Boots, B. (2018a). Semantically meaningful view selection. In *IROS 2018 (International Conference on Intelligent Robots and Systems), October 1st-5th, Madrid, Spain*.

[Guérin et al., 2018b] Guérin, J., Gibaru, O., Nyiri, E., Thiery, S., and Palos, J. (2018b). Automatic construction of real-world datasets for 3d object localization using two cameras. In *IECON 2018 (44th Annual Conference of the IEEE Industrial Electronics Society), October 21th-23th, Washington DC, USA*.

[Guérin et al., 2017] Guérin, J., Gibaru, O., Thiery, S., and Nyiri, E. (2017). Clustering for different scales of measurement - the gap-ratio weighted k-means algorithm.

In *CCSEIT 2017 (7th International Conference on Computer Science, Engineering and Information Technology), May 27th-28th, Vienna, Austria.*

[Guérin et al., 2017] Guérin, J., Gibaru, O., Thiery, S., and Nyiri, E. (2017). Locally optimal control under unknown dynamics with learnt cost function: application to industrial robot positioning. In *Journal of Physics Conference Series*, volume 783, page 012036.

[Guérin et al., 2018c] Guérin, J., Gibaru, O., Thiery, S., and Nyiri, E. (2018c). CNN features are also great at unsupervised classification. In *AIFU2018 (4th International Conference on Artificial Intelligence and Applications), February 17th-18th, Melbourne, Australie.*

[Guérin et al., 2018d] Guérin, J., Thiery, S., Nyiri, E., and Gibaru, O. (2018d). Unsupervised robotic sorting: Towards autonomous decision making robots. *International Journal of Artificial Intelligence and Applications (IJAIA)*, 9(2).

[Guo et al., 2017] Guo, X., Gao, L., Liu, X., and Yin, J. (2017). Improved deep embedded clustering with local structure preservation. In *International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 1753–1759.

[Gupta and Sukhatme, 2012] Gupta, M. and Sukhatme, G. S. (2012). Using manipulation primitives for brick sorting in clutter. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3883–3889. IEEE.

[Hazan et al., 2016] Hazan, E. et al. (2016). Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[Hu et al., 2017] Hu, W., Miyato, T., Tokui, S., Matsumoto, E., and Sugiyama, M. (2017). Learning discrete representations via information maximizing self augmented training. *arXiv preprint arXiv:1702.08720.*

[Huang et al., 2017] Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3.

[Hudson et al., 2012] Hudson, N., Howard, T., Ma, J., Jain, A., Bajracharya, M., Myint, S., Kuo, C., Matthies, L., Backes, P., Hebert, P., et al. (2012). End-to-end dexterous manipulation with deliberate interactive estimation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2371–2378. IEEE.

[Husome et al., 1978] Husome, R. G., Fleming, R. J., and Swanson, R. E. (1978). Color sorting system. US Patent 4,131,540.

[Itseez, 2015] Itseez (2015). Open source computer vision library. `https://github.com/itseez/opencv`.

[Jayaweera and Webb, 2010] Jayaweera, N. and Webb, P. (2010). Metrology-assisted robotic processing of aerospace applications. *International Journal of Computer Integrated Manufacturing*, 23(3):283–296.

[Jubien et al., 2014] Jubien, A., Gautier, M., and Janot, A. (2014). Dynamic identification of the kuka lwr robot using motor torques and joint torque sensors data. In *World Congress of the International Federation of Automatic Control (IFAC)*, pages 1–6.

[Käppeler et al., 2010] Käppeler, U.-P., Höferlin, M., and Levi, P. (2010). 3d object localization via stereo vision using an omnidirectional and a perspective camera. In *Proceedings of the 2nd Workshop on Omnidirectional Robot Vision, Anchorage, Alaska*, pages 7–12.

[Kim and Eustice, 2015] Kim, A. and Eustice, R. M. (2015). Active visual slam for robotic area coverage: Theory and experiment. *The International Journal of Robotics Research*, 34(4-5):457–475.

[Kim et al., 1998] Kim, D.-W., Choi, J.-S., and Nnaji, B. (1998). Robot arc welding operations planning with a rotating/tilting positioner. *International journal of production research*, 36(4):957–979.

[Kim et al., 2014] Kim, G., Sigal, L., and Xing, E. P. (2014). Joint summarization of large-scale collections of web images and videos for storyline reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4225–4232.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Knöbelreiter et al., 2016] Knöbelreiter, P., Reinbacher, C., Shekhovtsov, A., and Pock, T. (2016). End-to-end training of hybrid cnn-crf models for stereo. *arXiv preprint arXiv:1611.10229*.

[Kober and Peters, 2012] Kober, J. and Peters, J. (2012). Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer.

[Krainin et al., 2011] Krainin, M., Curless, B., and Fox, D. (2011). Autonomous generation of complete 3d object models using next best view manipulation planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5031–5037. IEEE.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Kuffner and LaValle, 2000] Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE.

[Kumar and Daumé, 2011] Kumar, A. and Daumé, H. (2011). A co-training approach for multi-view spectral clustering. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 393–400.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Lee and Grauman, 2009] Lee, Y. J. and Grauman, K. (2009). Shape discovery from unlabeled image collections. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2254–2261. IEEE.

[Lenz et al., 2015] Lenz, I., Lee, H., and Saxena, A. (2015). Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724.

[Levine and Abbeel, 2014] Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. In *Neural Information Processing Systems (NIPS)*.

[Levine et al., 2015a] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2015a). End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*.

[Levine and Koltun, 2013] Levine, S. and Koltun, V. (2013). Guided policy search. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1–9.

[Levine et al., 2016] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2016). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, page 0278364917710318.

[Levine et al., 2015b] Levine, S., Wagener, N., and Abbeel, P. (2015b). Learning contact-rich manipulation skills with guided policy search. *CoRR*, abs/1501.05611.

[Lewis et al., 2012] Lewis, F. L., Vrabie, D., and Syrmos, V. L. (2012). *Optimal Control*. John Wiley & Sons.

[Li and Todorov, 2004] Li, W. and Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics*, pages 222–229.

[Lichman, 2013] Lichman, M. (2013). UCI machine learning repository.

[Lin, 2009] Lin, C.-K. (2009). H-infty reinforcement learning control of robot manipulators using fuzzy wavelet networks. *Fuzzy Sets and Systems*, 160(12):1765–1786.

[Liu et al., 2016] Liu, H., Shao, M., Li, S., and Fu, Y. (2016). Infinite ensemble for image clustering. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[Liu et al., 2012] Liu, M.-Y., Tuzel, O., Veeraraghavan, A., Taguchi, Y., Marks, T. K., and Chellappa, R. (2012). Fast object localization and pose estimation in heavy clutter for robotic bin picking. *The International Journal of Robotics Research*, 31(8):951–973.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

[Lucke et al., 2008] Lucke, D., Constantinescu, C., and Westkämper, E. (2008). Smart factory-a step towards the next generation of manufacturing. In *Manufacturing systems and technologies for the new frontier*, pages 115–118. Springer.

[Lukka et al., 2014] Lukka, T. J., Tossavainen, T., Kujala, J. V., and Raiko, T. (2014). Zenrobotics recycler–robotic sorting using machine learning. In *Proceedings of the International Conference on Sensor-Based Sorting (SBS)*.

[Luo et al., 2016] Luo, W., Schwing, A. G., and Urtasun, R. (2016). Efficient deep learning for stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5695–5703.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Mair, 1988] Mair, G. M. (1988). *Industrial robotics*. Prentice Hall.

[Majarena et al., 2010] Majarena, A. C., Santolaria, J., Samper, D., and Aguilar, J. J. (2010). An overview of kinematic and calibration models using internal/external sensors or constraints to improve the behavior of spatial parallel mechanisms. *Sensors*, 10(11):10256–10297.

[McCarthy and Feigenbaum, 1990] McCarthy, J. and Feigenbaum, E. A. (1990). In memoriam: Arthur samuel: Pioneer in machine learning. *AI Magazine*, 11(3):10.

[Mitrovic et al., 2010] Mitrovic, D., Klanke, S., and Vijayakumar, S. (2010). Adaptive optimal feedback control with learned internal dynamics models. In *From Motor Learning to Interaction Learning in Robots*, pages 65–84. Springer.

[Montgomery and Levine, 2016] Montgomery, W. H. and Levine, S. (2016). Guided policy search via approximate mirror descent. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4008–4016. Curran Associates, Inc.

[Morales et al., 2006] Morales, A., Asfour, T., Azad, P., Knoop, S., and Dillmann, R. (2006). Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5663–5668. IEEE.

[Mouli and Raju, 2013] Mouli, C. C. and Raju, K. N. (2013). A review on wireless embedded system for vision guided robot arm for object sorting. *International Journal of Scientific & Engineering Research*.

[Muller et al., 2006] Muller, U., Ben, J., Cosatto, E., Flepp, B., and Cun, Y. L. (2006). Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746.

[Munzer et al., 2017] Munzer, T., Toussaint, M., and Lopes, M. (2017). Efficient behavior learning in human–robot collaboration. *Autonomous Robots*, pages 1–13.

[Murtagh, 1983] Murtagh, F. (1983). A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359.

[Nayar et al., 1996] Nayar, S., Nene, S., and Murase, H. (1996). Columbia object image library (coil 100). *Department of Comp. Science, Columbia University, Tech. Rep. CUCS-006-96*.

[Nilsback and Zisserman, 2009] Nilsback, M.-E. and Zisserman, A. (2009). Delving deeper into the whorl of flower segmentation. *Image and Vision Computing*.

[Nkomo and Collier, 2012] Nkomo, M. and Collier, M. (2012). A color-sorting scara robotic arm. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 763–768. IEEE.

[Nof, 1999] Nof, S. Y. (1999). *Handbook of industrial robotics*, volume 1. John Wiley & Sons.

[Nubiola and Bonev, 2013] Nubiola, A. and Bonev, I. A. (2013). Absolute calibration of an abb irb 1600 robot using a laser tracker. *Robotics and Computer-Integrated Manufacturing*, 29(1):236–245.

[Oja and Hyvarinen, 1997] Oja, E. and Hyvarinen, A. (1997). A fast fixed-point algorithm for independent component analysis. *Neural computation*, 9(7):1483–1492.

[Olabi et al., 2010] Olabi, A., Béarée, R., Gibaru, O., and Damak, M. (2010). Feedrate planning for machining with industrial six-axis robots. *Control Engineering Practice*, 18(5):471–482.

[Park et al., 2007] Park, J.-J., Kim, J.-H., and Song, J.-B. (2007). Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning. *International Journal of Control Automation and Systems*, 5(6):674.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pereira et al., 2014] Pereira, V., Fernandes, V. A., and Sequeira, J. (2014). Low cost object sorting robotic arm using raspberry pi. In *Global Humanitarian Technology Conference-South Asia Satellite (GHTC-SAS), 2014 IEEE*, pages 1–6. IEEE.

[Pfitzner et al., 2009] Pfitzner, D., Leibbrandt, R., and Powers, D. (2009). Characterization and evaluation of similarity measures for pairs of clusterings. *Knowledge and Information Systems*, 19(3):361.

[Quattoni and Torralba, 2009] Quattoni, A. and Torralba, A. (2009). Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 413–420. IEEE.

[Ren and Fan, 2011] Ren, S. and Fan, A. (2011). K-means clustering algorithm based on coefficient of variation. In *Image and Signal Processing (CISP), 2011 4th International Congress on*, volume 4, pages 2076–2079. IEEE.

[Romera-Paredes and Torr, 2016] Romera-Paredes, B. and Torr, P. H. S. (2016). Recurrent instance segmentation. In *European Conference on Computer Vision*, pages 312–329. Springer.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[Scharstein et al., 2014] Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nesic, N., Wang, X., and Westling, P. (2014). High-resolution stereo datasets with subpixel-accurate ground truth. In Jiang, X., Hornegger, J., and Koch, R., editors, *GCPR*, volume 8753 of *Lecture Notes in Computer Science*, pages 31–42. Springer.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[Sciavicco and Siciliano, 2000] Sciavicco, L. and Siciliano, B. (2000). Modelling and control of robot manipulators.

[Seldin et al., 2003] Seldin, Y., Starik, S., and Werman, M. (2003). Unsupervised clustering of images using their joint segmentation. In *Proceedings of the 3rd International Workshop on Statistical and Computational Theories of Vision (SCTV 2003)*.

[Sermanet et al., 2018] Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., and Levine, S. (2018). Time-contrastive networks: Self-supervised learning from video. *Proceedings of International Conference in Robotics and Automation (ICRA)*.

[Shi et al., 2016] Shi, Y., Long, P., Xu, K., Huang, H., and Xiong, Y. (2016). Data-driven contextual modeling for 3d scene understanding. *Computers & Graphics*, 55:55–67.

[Shum et al., 2016] Shum, A., Wang, Y., and Hsieh, S.-J. (2016). Design, build and remote control of a miniature automated robotic sorting system. *Design, Build*, 141(3).

[Sicard and Levine, 1988] Sicard, P. and Levine, M. D. (1988). An approach to an expert robot welding system. *IEEE transactions on systems, man, and cybernetics*, 18(2):204–222.

[Siciliano and Khatib, 2016] Siciliano, B. and Khatib, O. (2016). *Springer handbook of robotics*. Springer.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Singh et al., 2016] Singh, T., Dhaytadak, D., Kadam, P., and Sapkal, R. (2016). Object sorting by robotic arm using image processing. *International Research Journal of Engineering and Technology (IRJET)*.

[Stevens, 1946] Stevens, S. S. (1946). On the theory of scales of measurement.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge.

[Szabo and Lie, 2012] Szabo, R. and Lie, I. (2012). Automated colored object sorting application for robotic arms. In *Electronics and Telecommunications (ISETC), 2012 10th International Symposium on*, pages 95–98. IEEE.

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

[Tan et al., 2015] Tan, H., Xu, Y., Mao, Y., Tong, X., Griffin, W. B., Kannan, B., and DeRose, L. A. (2015). An integrated vision-based robotic manipulation system for sorting surgical tools. In *Technologies for Practical Robot Applications (TePRA), 2015 IEEE International Conference on*, pages 1–6. IEEE.

[Tao et al., 2017] Tao, Z., Liu, H., Li, S., Ding, Z., and Fu, Y. (2017). From ensemble clustering to multi-view clustering. In *Proc. of the Twenty-Sixth Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 2843–2849.

[Tassa et al., 2012] Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE.

[Theodoridis and Koutroumbas, 2006a] Theodoridis, S. and Koutroumbas, K. (2006a). Chapter 1-introduction. *Pattern Recognition*, pages 1–12.

[Theodoridis and Koutroumbas, 2006b] Theodoridis, S. and Koutroumbas, K. (2006b). Chapter 11-clustering: Basic concepts. *Pattern Recognition*, pages 595–625.

[Theodoridis and Koutroumbas, 2006c] Theodoridis, S. and Koutroumbas, K. (2006c). Chapter 14-clustering algorithms iii: Schemes based on function optimization. *Pattern Recognition*, pages 701–763.

[Theodoridis et al., 2010] Theodoridis, S., Pikrakis, A., Koutroumbas, K., and Cavouras, D. (2010). *Introduction to pattern recognition: a matlab approach*. Academic Press.

[Thomaz and Giraldi, 2010] Thomaz, C. E. and Giraldi, G. A. (2010). A new ranking method for principal components analysis and its application to face image analysis. *Image and Vision Computing*, 28(6):902–913.

[Tishby et al., 2000] Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. *arXiv preprint physics/0004057*.

[Todorov and Li, 2005] Todorov, E. and Li, W. (2005). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306 vol. 1.

[Tsarouchi et al., 2017] Tsarouchi, P., Matthaiakis, A.-S., Makris, S., and Chryssolouris, G. (2017). On a human-robot collaboration in an assembly cell. *International Journal of Computer Integrated Manufacturing*, 30(6):580–589.

[Vapnik and Vapnik, 1998] Vapnik, V. N. and Vapnik, V. (1998). *Statistical learning theory*, volume 1. Wiley New York.

[Vega-Pons and Ruiz-Shulcloper, 2011] Vega-Pons, S. and Ruiz-Shulcloper, J. (2011). A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(03):337–372.

[Wallén, 2008] Wallén, J. (2008). *The history of the industrial robot*. Linköping University Electronic Press.

[Wang et al., 2017] Wang, X., Lu, L., Shin, H.-c., Kim, L., Bagheri, M., Nogues, I., Yao, J., and Summers, R. M. (2017). Unsupervised joint mining of deep features and image labels for large-scale radiology image categorization and scene recognition. *arXiv preprint arXiv:1701.06599*.

[Wang et al., 2016] Wang, Y., Zhang, W., Wu, L., Lin, X., Fang, M., and Pan, S. (2016). Iterative views agreement: An iterative low-rank based structured optimization method to multi-view spectral clustering. *arXiv preprint arXiv:1608.05560*.

[Wechsler et al., 2012] Wechsler, H., Phillips, J. P., Bruce, V., Soulie, F. F., and Huang, T. S. (2012). *Face recognition: From theory to applications*, volume 163. Springer Science & Business Media.

[Welinder et al., 2010] Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. (2010). Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology.

[Westerlund, 2000] Westerlund, L. (2000). *The extended arm of man: a history of industrial robot.* Informationsförlaget.

[Xie et al., 2016] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning*, pages 478–487.

[Xu and Wunsch, 2005] Xu, R. and Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678.

[Xu et al., 2014] Xu, Z., Tao, D., Zhang, Y., Wu, J., and Tsoi, A. C. (2014). Architectural style classification using multinomial latent logistic regression. In *European Conference on Computer Vision*, pages 600–615. Springer.

[Yang et al., 2016] Yang, J., Parikh, D., and Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5147–5156.

[Yegnanarayana, 2009] Yegnanarayana, B. (2009). *Artificial neural networks.* PHI Learning Pvt. Ltd.

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

[Zawadzki and Żywicki, 2016] Zawadzki, P. and Żywicki, K. (2016). Smart product design and production control for effective mass customization in the industry 4.0 concept. *Management and Production Engineering Review*, 7(3):105–112.

[Zbontar and LeCun, 2016] Zbontar, J. and LeCun, Y. (2016). Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2.

[Zhang et al., 2012] Zhang, W., Wang, X., Zhao, D., and Tang, X. (2012). Graph degree linkage: Agglomerative clustering on a directed graph. In *European Conference on Computer Vision*, pages 428–441. Springer.

[Zhang et al., 2013] Zhang, W., Zhao, D., and Wang, X. (2013). Agglomerative clustering via maximum incremental path integral. *Pattern Recognition*, 46(11):3056–3065.

[Zhao et al., 2017] Zhao, H., Ding, Z., and Fu, Y. (2017). Multi-view clustering via deep matrix factorization. In *AAAI*, pages 2921–2927.

[Zhihong et al., 2017] Zhihong, C., Hebin, Z., Yanbo, W., Binyan, L., and Yu, L. (2017). A vision-based robotic grasping system using deep learning for garbage sorting. In *Control Conference (CCC), 2017 36th Chinese*, pages 11223–11226. IEEE.

[Zoph et al., 2017] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2017). Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*.

Part VI

Appendix

Appendix A

A beginner's definition of Machine Learning

Every morning, a fruit retailing company receives apples and oranges mixed in the same containers. Then, fruits go on a conveyor one by one and need to be placed in the crates corresponding to their type. For such task, we can think of different levels of automation. A first, very simple solution is to have someone stand by the end of the conveyor, who decides whether a fruit is an apple or an orange. Even though such method would work perfectly, it has the drawback of requiring a worker fully dedicated to this tedious task. To automate such procedure, another possibility is to automatically measure characteristics of the different fruits, such as color, size, etc. Such characteristics are called features and are then used to determine where the fruits should go. To do so, an engineer can analyze the features, plot them, compute statistics, etc. and come up with more or less complex rules to decide whether a fruit is an apple or an orange. Such automation can save valuable time and money for the company, but is limited to apples and oranges. If the company starts receiving melons, they would need to call the engineer again, so that he can come up with new rules that take melons into account. In this simple example, using Machine Learning (ML) can be thought of as automating the work of the engineer. By automatically analyzing some labelled examples, i.e. features measurements and corresponding categories, a machine learning algorithm can come up with general rules for classifying previously unseen fruits.

The famous example of apples and oranges classification is basic, but yet is a good illustration of why ML receives so much interest. It enables to define complex programs and models for specific cases, by generalizing from examples. It is easy, even for a non-specialist, to teach a well-designed ML algorithm to carry out new tasks. Arthur Samuel ([McCarthy and Feigenbaum, 1990]), one of the pioneers of ML who

wrote the first self-learning program in 1959, defines ML as the "field of study that gives computers the ability to learn without being explicitly programmed". In other words, a ML program is not written for a specific task but for a class of problems; it can modify itself to solve different problems of the same nature. This very generic definition contains all the subfield of ML which we are about to define.

There exists many different ways of grouping different ML algorithms. One possible classification is to consider the kind of inputs received by the algorithm and the outputs it produces. From there, we can define three main categories:

- **Supervised Learning:** From a training set, composed of pairs of data objects and desired outputs, a supervised learning algorithm learns general rules to map inputs to outputs. Such rules should generalize to predict outputs for previously unseen data. It is called supervised learning because it generalizes from known examples. The classification of apples and oranges described above is an example of supervised learning.

- **Unsupervised learning:** The goal of unsupervised learning is mainly to group data based on their similarity. Inputs do not have labels, the idea is to maximize similarity between objects in the same groups. For instance, finding two groups among a set of fruits features, without initial labelled examples is an unsupervised learning problem.

- **Reinforcement learning:** The last major area of ML is Reinforcement Learning (RL). It serves as a tool to solve control problems and games. The RL framework is composed of an agent, who evolves by taking actions in an environment. The environment responds by sending a score to reward (or punish) the agent according to a user defined criteria. The rewards measure how good the agent's action are with respect to the task at hand. Then, by interacting with the environment, the agent should change its behavior to choose actions which maximize reward over its lifetime. In other words, RL consists in learning through interaction. A good text book to better understand RL concepts is ([Sutton and Barto, 1998]).

Although less frequent, we also find the field of *semi-supervised learning*, which mixes labelled and unlabelled data, as well as *active learning*, where the algorithm can request specific data from the user.

When defining ML, it is also interesting to consider the kind of problems it can solve. Apart from the RL problems, which were already introduced, two other main problems are solved using ML:

- **Classification:** Given a data object (vector, matrix, pixels, etc.), the goal of classification is to identify the group it belongs to. A good definition of the classification problem as well as tools to solve it can be found in ([Theodoridis et al., 2010]). Classification can be either supervised or unsupervised, in which case it is also called clustering.

- **Regression:** Regression, or function estimation, consists in finding the output of a function for a given input. More details can be found in ([Vapnik and Vapnik, 1998]). If the outputs are finite and discrete, a regression problem becomes a classification problem, however, the special structure of classification problems is widely used and should be kept as a separate problem. Most of the time, regressions with continuous outputs are supervised learning problems.

We conclude this brief ML introduction with a word about deep learning, a subfield of supervised learning which has received much attention lately. As shown in the initial example of apples and oranges, ML is an interesting tool to automate model creation and parameters tuning, it is a nice way to enable people with little knowledge to create usable, case specific programs. In addition to this nice application, some ML algorithms are also able to learn things humans cannot program. For example, learning functions to associate raw images pixel representations to categories. The most widely used tool for such application are artificial Neural Networks (NN) ([Yegnanarayana, 2009]) and more particularly deep neural networks ([LeCun et al., 2015]). Such NN models can be arbitrarily complex and are able to discover highly complex models and structures in the data if provided with enough examples. A good application to illustrate deep learning is image classification. For example, using deep NN with image pixels as inputs (([Szegedy et al., 2016]), ([Huang et al., 2017])), very low error rates can be reached for the problem of ImageNet classification ([Russakovsky et al., 2015]). ImageNet is a dataset composed of over one million high-resolution images, which needs to be classified within 1000 classes. Such deep NN predictors are composed of tens of millions of parameters, and thus are impossible to code manually.

Appendix B

Complete result tables from Chapter 3 benchmark study

Table B.1: Results on natural object recognition datasets.

		VOC2007						COIL100					
		KM			Agg			KM			Agg		
		nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*
VGG16	L1	0.516	0.656	*82.5*	0.525	0.549	*96.8*	0.895	0.821	*457*	0.928	0.863	*476*
	L2	0.578	0.557	*18.3*	0.608	0.615	*16*	0.943	0.895	*72*	0.940	0.875	*71*
	L3	0.673	0.636	*16.4*	0.651	0.653	*16.1*	0.945	0.900	*77*	0.956	0.919	*70*
VGG19	L1	0.541	0.605	*83*	0.537	0.533	*98*	0.889	0.791	*440*	0.922	0.845	*476*
	L2	0.625	0.593	*17.6*	0.618	0.642	*16.1*	0.937	0.881	*67*	0.949	0.892	*71*
	L3	0.661	0.628	*18.5*	0.650	0.631	*16.8*	0.939	0.883	*62*	0.948	0.894	*70*
Inception	L1	0.216	0.364	*825*	0.298	0.382	*856*	0.799	0.642	*4,340*	0.846	0.719	*4,455*
	L2	0.423	0.447	*565*	0.542	0.592	*531*	0.913	0.828	*2,852*	0.939	0.873	*2,517*
	L3	0.603	0.561	*8.6*	0.692	0.681	*8.6*	0.928	0.868	*37*	0.953	0.904	*35*
Xception	L1	0.291	0.358	*356*	0.410	0.510	*403*	0.832	0.692	*1,965*	0.875	0.763	*1,960*
	L2	0.538	0.509	*901*	0.625	0.616	*822*	0.920	0.842	*3,754*	0.942	0.878	*3,909*
	L3	0.687	0.630	*8.7*	0.719	0.709	*8.1*	0.938	0.881	*32*	0.955	0.911	*35*
Resnet50	L1	0.279	0.493	*758*	0.315	0.367	*783*	0.850	0.718	*4,475*	0.888	0.782	*3,876*
	L2	0.413	0.548	*99*	0.494	0.491	*97*	0.884	0.771	*489*	0.927	0.846	*474*
	L3	0.680	0.641	*8.4*	0.656	0.666	*8*	0.957	0.926	*35*	0.967	0.940	*35*
BoF		0.083	0.325	*0.5*	0.072	0.342	*0.4*	-	-	-	-	-	-

Table B.2: Results on Scene recognition datasets.

		Archi						MIT					
		KM			Agg			KM			Agg		
		nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*
VGG16	L1	0.425	0.451	*218*	0.420	0.429	*210*	0.383	0.400	*1,826*	0.410	0.349	*2,259*
	L2	0.420	0.398	*35*	0.431	0.413	*31.2*	0.481	0.357	*285*	0.471	0.360	*328*
	L3	0.430	0.371	*39*	0.414	0.382	*31.4*	0.515	0.406	*283*	0.492	0.398	*334*
VGG19	L1	0.416	0.447	*219*	0.431	0.449	*211*	0.388	0.389	*1,749*	0.411	0.339	*2,235*
	L2	0.415	0.392	*37*	0.426	0.400	*31.4*	0.484	0.380	*288*	0.480	0.420	*331*
	L3	0.408	0.359	*34.1*	0.398	0.362	*31.2*	0.510	0.389	*289*	0.491	0.400	*331*
Inception	L1	0.184	0.207	*1,747*	0.178	0.250	*1,909*	0.209	0.492	*170.4*	0.378	0.291	*36,824*
	L2	0.412	0.388	*1,142*	0.401	0.394	*1,119*	0.495	0.419	*10,005*	0.498	0.399	*12,243*
	L3	0.420	0.363	*16.1*	0.421	0.395	*15.5*	0.570	0.446	*138*	0.561	0.466	*164*
Xception	L1	0.165	0.246	*595*	0.174	0.243	*859*	0.430	0.390	*7,307*	0.457	0.378	*9,177*
	L2	0.435	0.381	*1,636*	0.435	0.424	*1,756*	0.500	0.572	*174.8*	0.548	0.455	*29,548*
	L3	0.442	0.384	*15.5*	0.433	0.401	*15.6*	0.597	0.471	*121*	0.587	0.488	*164*
Resnet50	L1	0.248	0.282	*1,827*	0.270	0.285	*1,674*	0.193	0.598	*165.7*	0.393	0.321	*23949*
	L2	0.377	0.368	*238*	0.389	0.402	*210*	0.367	0.365	*1,684*	0.395	0.315	*2,252*
	L3	0.455	0.410	*17*	0.447	0.414	*22*	0.539	0.426	*133*	0.529	0.443	*164*
BoF		0.100	0.555	*0.9*	0.102	0.681	*1.1*	-	-	-	-	-	-

Table B.3: Results on fine-grained recognition datasets.

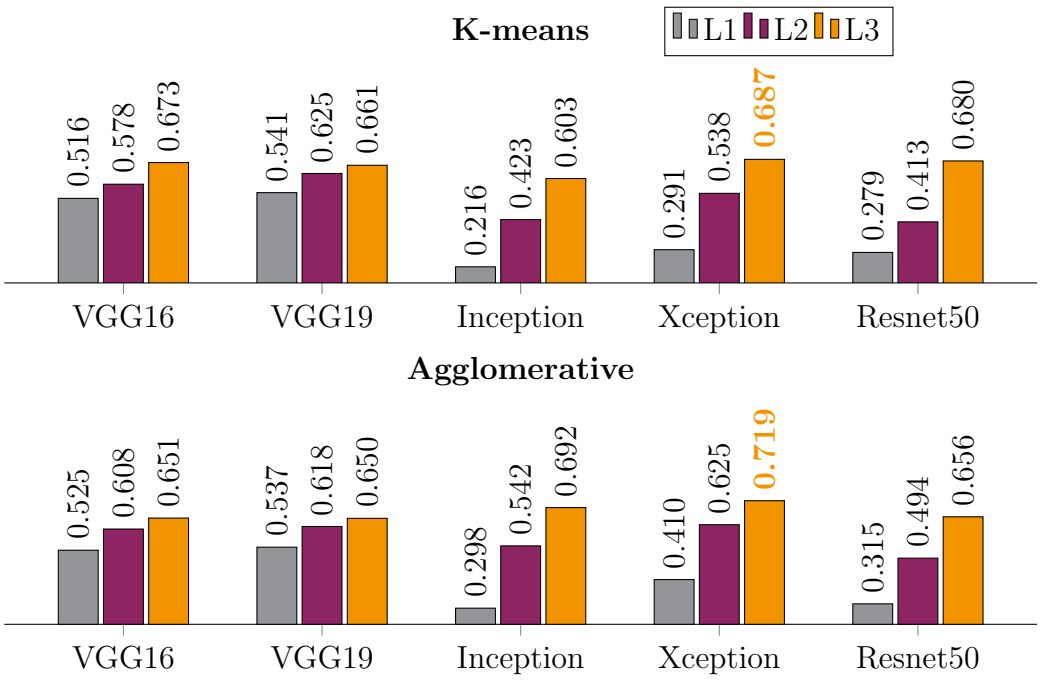
		Flowers						Birds					
		KM			Agg			KM			Agg		
		nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*
VGG16	L1	0.588	0.609	*36.5*	0.616	0.623	*17.1*	0.375	0.160	*2,309*	0.420	0.174	*1,288*
	L2	0.657	0.652	*5.5*	0.661	0.685	*2.5*	0.529	0.301	*399*	0.542	0.325	*190*
	L3	0.688	0.680	*5.7*	0.644	0.651	*2.7*	0.557	0.314	*371*	0.557	0.332	*189*
VGG19	L1	0.567	0.710	*35*	0.614	0.660	*17.1*	0.390	0.170	*2,369*	0.428	0.182	*1,282*
	L2	0.652	0.699	*5.5*	0.676	0.717	*2.5*	0.534	0.303	*377*	0.544	0.315	*189*
	L3	0.652	0.696	*5.3*	0.646	0.688	*2.5*	0.562	0.331	*383*	0.557	0.333	*187*
Inception	L1	0.241	0.367	*364*	0.329	0.385	*155*	0.143	0.144	*145.7*	0.306	0.078	*17,966*
	L2	0.560	0.586	*219*	0.604	0.630	*90.2*	0.464	0.192	*12,483*	0.484	0.212	*6,904*
	L3	0.650	0.645	*2.6*	0.677	0.700	*1.1*	0.558	0.326	*164*	0.569	0.342	*95*
Xception	L1	0.425	0.501	*139*	0.495	0.599	*69.7*	0.341	0.106	*8,919*	0.366	0.122	*5,194*
	L2	0.633	0.643	*298*	0.661	0.690	*140*	0.481	0.307	*189.8*	0.548	0.291	*11,159*
	L3	0.674	0.681	*2.9*	0.670	0.704	*1.2*	0.633	0.428	*159*	0.644	0.452	*94*
Resnet50	L1	0.543	0.611	*298*	0.589	0.626	*137*	0.146	0.413	*202.3*	0.331	0.102	*10,994*
	L2	0.510	0.713	*37.6*	0.607	0.679	*17.2*	0.375	0.142	*2,630*	0.421	0.157	*1,273*
	L3	0.684	0.668	*2.7*	0.708	0.721	*2.3*	0.521	0.290	*197*	0.529	0.312	*94*
BoF		0.177	0.419	*0.331*	0.179	0.388	*0.103*	-	-	-	-	-	-

Table B.4: Results on face recognition datasets.

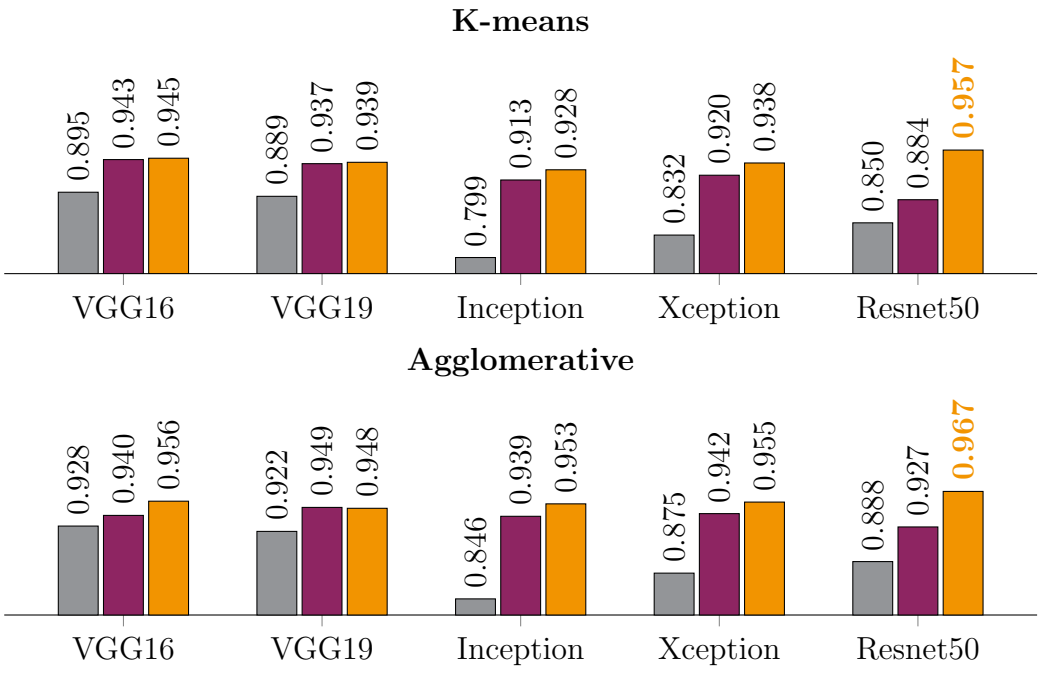
		Umist						FEI					
		KM			Agg			KM			Agg		
		nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*	nmi	pur	*time*
VGG16	L1	0.575	0.494	*11.9*	0.699	0.600	*3.2*	0.875	0.793	*304*	0.909	0.858	*72*
	L2	0.669	0.543	*1.7*	0.717	0.560	*0.5*	0.899	0.834	*52*	0.900	0.835	*11*
	L3	0.669	0.529	*1.5*	0.689	0.532	*1.5*	0.887	0.804	*45*	0.897	0.825	*11*
VGG19	L1	0.623	0.490	*12.7*	0.717	0.624	*3.1*	0.888	0.821	*311*	0.923	0.878	*93*
	L2	0.719	0.597	*1.5*	0.766	0.671	*0.4*	0.905	0.837	*47*	0.918	0.862	*15*
	L3	0.700	0.577	*1.5*	0.740	0.635	*0.4*	0.908	0.848	*45*	0.915	0.857	*13*
Inception	L1	0.588	0.499	*104*	0.641	0.530	*28.1*	0.881	0.788	*3,366*	0.920	0.868	*663*
	L2	0.661	0.532	*60.9*	0.694	0.565	*16.5*	0.923	0.879	*1,843*	0.939	0.904	*383*
	L3	0.698	0.563	*0.8*	0.760	0.647	*0.2*	0.921	0.854	*21*	0.941	0.899	*5.3*
Xception	L1	0.589	0.471	*47.2*	0.642	0.511	*12.9*	0.789	0.630	*1,393*	0.831	0.690	*296*
	L2	0.688	0.588	*89.9*	0.745	0.635	*25.5*	0.930	0.893	*2,712*	0.934	0.896	*599*
	L3	0.683	0.593	*0.8*	0.731	0.607	*0.2*	0.913	0.852	*20*	0.928	0.884	*5.2*
Resnet50	L1	0.533	0.443	*87.9*	0.597	0.459	*24.9*	0.843	0.718	*3,137*	0.892	0.816	*579*
	L2	0.555	0.463	*11.3*	0.623	0.510	*3.1*	0.903	0.840	*324*	0.931	0.897	*72*
	L3	0.658	0.532	*770*	0.717	0.579	*0.2*	0.910	0.841	*25*	0.919	0.859	*5.3*
BoF		0.542	0.459	*0.306*	0.638	0.522	*0.03*	-	-	-	-	-	-

Appendix C

Complete NMI scores for different feature representations from Chapter 3 benchmark study

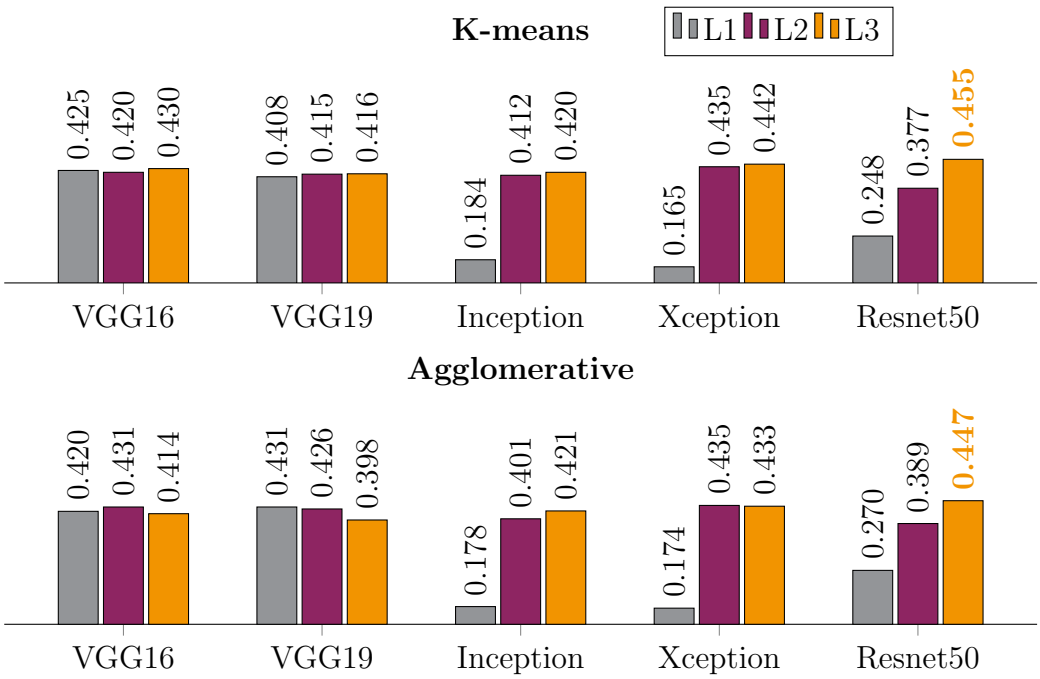


(a) VOC2007

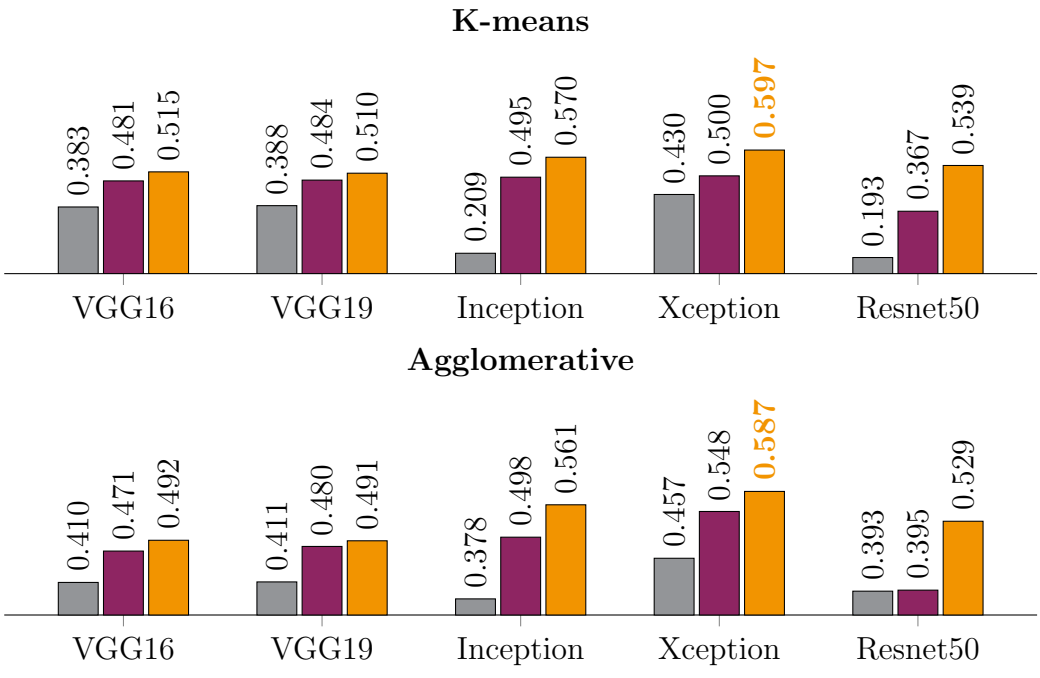


(b) COIL100

Figure C.1: NMI scores on natural object recognition datasets.

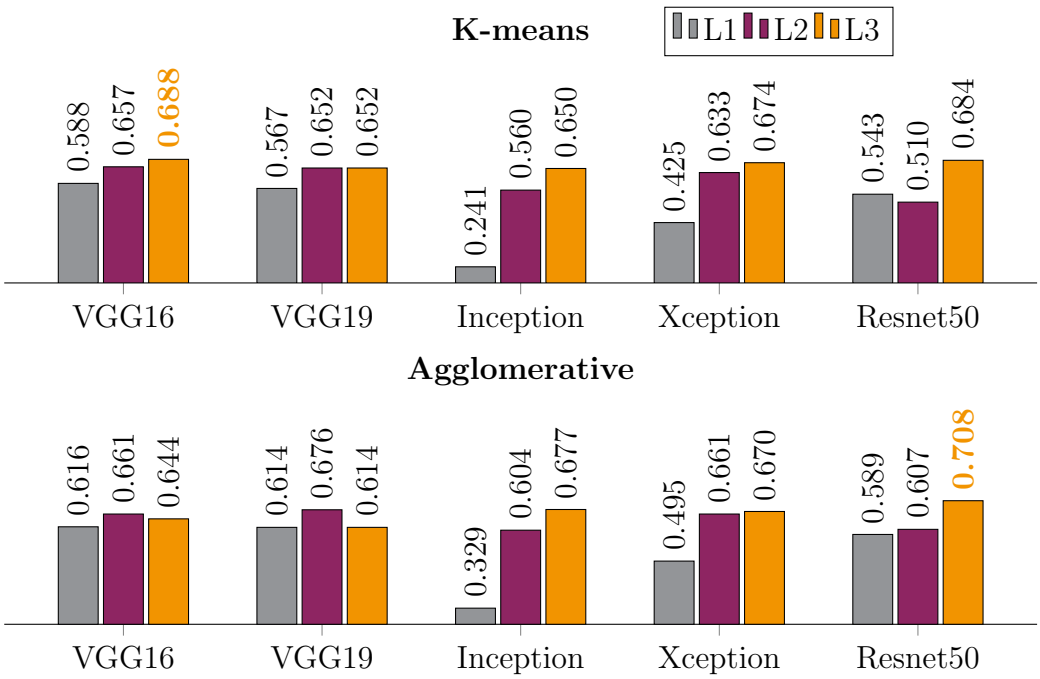


(a) Archi

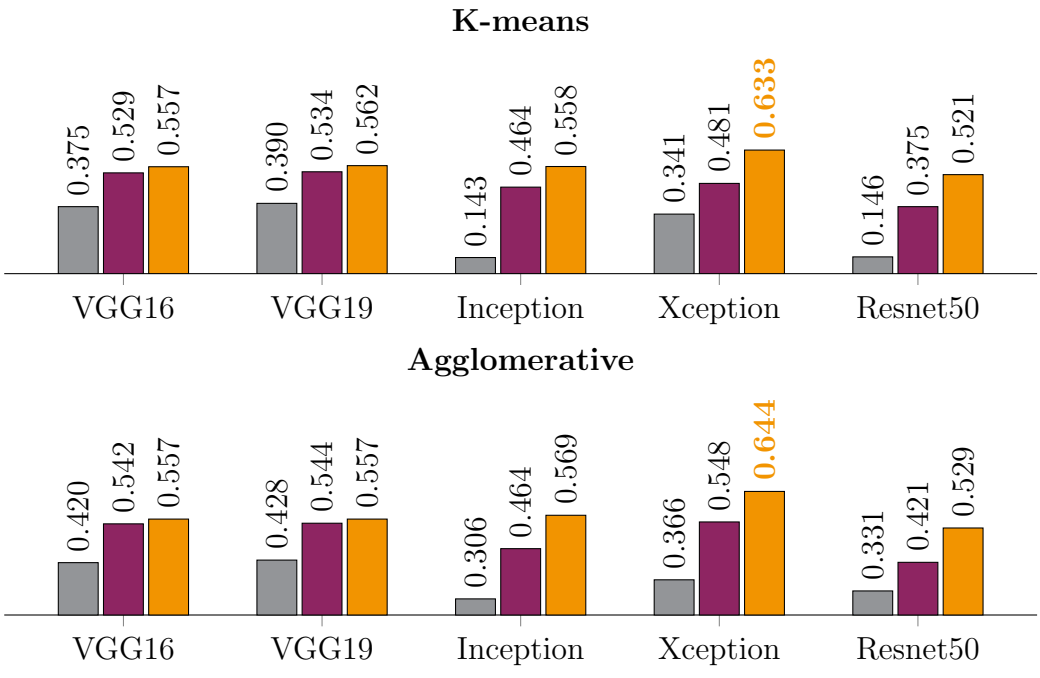


(b) MIT

Figure C.2: NMI scores on scene recognition datasets.

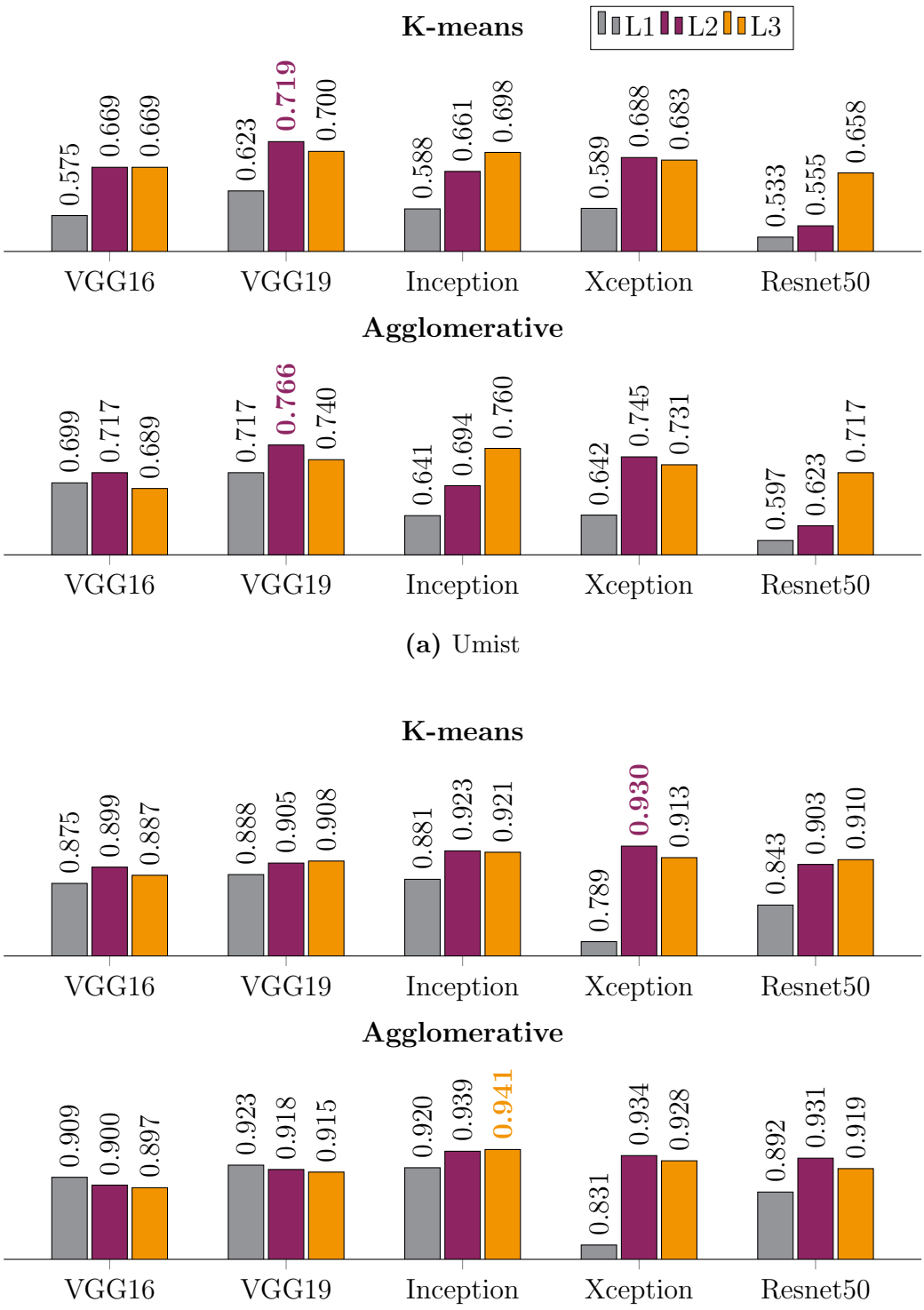


(a) Flowers



(b) Birds

Figure C.3: NMI scores on natural fine-grained datasets.



(a) Umist

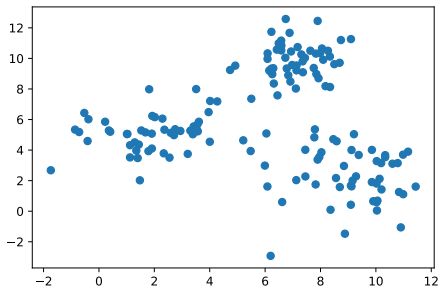
(b) FEI

Figure C.4: NMI scores on face recognition datasets.

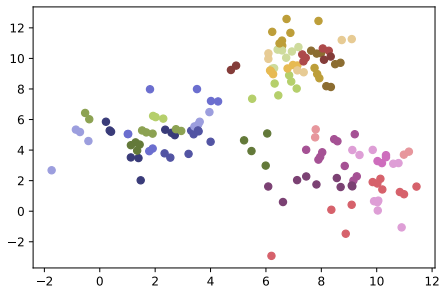
Appendix D

Illustration of the JULE methodology on a 2d made-up example

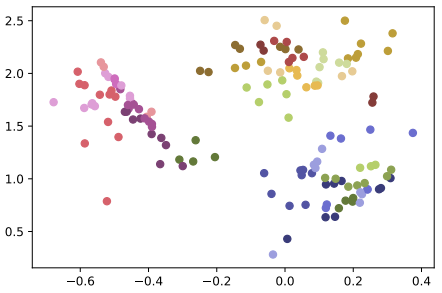
This appendix aims at illustrating the JULE methodology (see Section 4.3.3.1 for more details). To do so, we propose to visualize the evolution of a made up 2 dimensional dataset throughout the JULE pipeline. The initial data on which JULE is applied can be seen in Figure D.1a. These data are transformed into another 2d space, using JULE to train a feed forward neural network of dimensions $2 - 100 - 2$ with relu activations and L2 regularization. The result of cluster initialization can be seen in both the initial space and the latent space respectively in Figures D.1b and D.1c. Figures D.1d to D.1h are then represented in the latent space. The unfold rate is set to 0.75, which justifies the fact that the expected number of clusters 3 is reached after two merging passes.



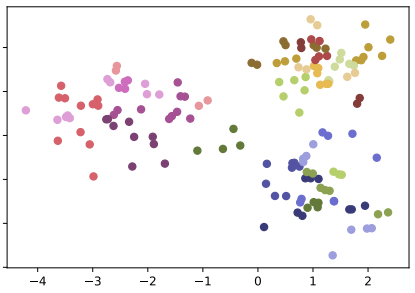
(a) Initial data ($N_c = 150$)



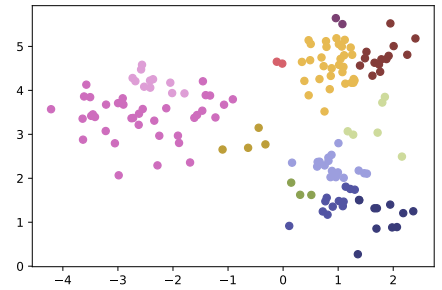
(b) Clusters initialization ($N_c = 42$)



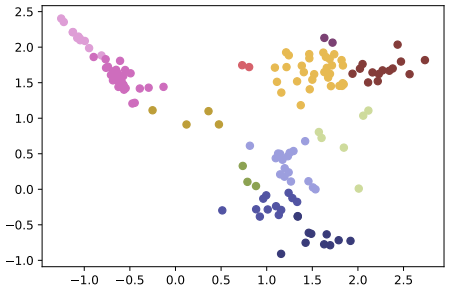
(c) NN initialization ($N_c = 42$)



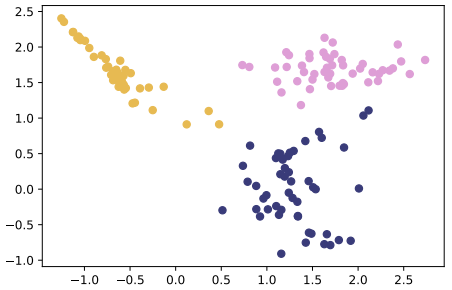
(d) First NN training ($N_c = 42$)



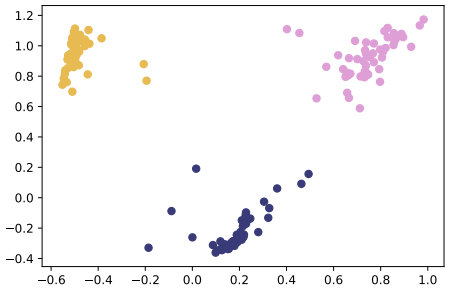
(e) First clusters merging ($N_c = 9$)



(f) Second NN training ($N_c = 9$)



(g) Second clusters merging ($N_c = 3$)



(h) Final NN training ($N_c = 3$)

Figure D.1: Made-up 2d example to illustrate the JULE methodology. *N_c stands for the number of clusters at a given stage of the JULE pipeline.*

Appendix E

Complete result tables from Chapter 4

Table E.1: Results on natural object recognition datasets.
(N/A: *incompatible methods*)

	VOC2007				COIL100			
	JULE		Agg		JULE		Agg	
	nmi	pur	nmi	pur	nmi	pur	nmi	pur
VGG16	0.695	0.761	0.651	0.709	0.994	0.970	0.956	0.879
VGG19	0.676	0.734	0.650	0.711	0.994	0.978	0.948	0.846
Inception	0.764	0.819	0.692	0.757	0.988	0.948	0.953	0.851
Xception	0.764	0.785	0.719	0.798	0.988	0.949	0.955	0.855
Resnet50	0.719	0.762	0.656	0.717	0.996	0.980	0.967	0.899
Densenet121	0.748	0.778	0.734	0.792	0.991	0.964	0.963	0.880
Densenet169	0.742	0.791	0.713	0.763	0.993	0.970	0.962	0.880
Densenet201	0.769	0.800	0.729	0.789	0.993	0.970	0.972	0.901
InceptionResnet	0.763	0.809	0.643	0.699	0.982	0.928	0.927	0.791
Nasnet	0.752	0.811	0.668	0.690	0.979	0.906	0.943	0.809
MVEC	0.821	0.813	0.794	0.816	0.996	0.980	0.967	0.874
CC	0.745	0.806	N/A	N/A	0.996	0.980	N/A	N/A
MVnet	0.817	0.826	N/A	N/A	0.995	0.980	N/A	N/A
MVnet-retrain	0.827	0.860	N/A	N/A	0.996	0.980	N/A	N/A

Table E.2: Results on scene recognition datasets.
(N/A: *incompatible methods*)

	Archi				MIT			
	JULE		Agg		JULE		Agg	
	nmi	pur	nmi	pur	nmi	pur	nmi	pur
VGG16	0.393	0.331	0.414	0.402	0.473	0.385	0.492	0.445
VGG19	0.403	0.349	0.398	0.384	0.485	0.422	0.491	0.445
Inception	0.433	0.343	0.421	0.407	0.521	0.430	0.561	0.508
Xception	0.447	0.402	0.433	0.417	0.574	0.483	0.587	0.539
Resnet50	0.422	0.333	0.447	0.449	0.496	0.404	0.529	0.480
Densenet121	0.455	0.376	0.455	0.462	0.505	0.406	0.542	0.485
Densenet169	0.436	0.367	0.471	0.462	0.531	0.462	0.572	0.523
Densenet201	0.465	0.431	0.480	0.480	0.475	0.346	0.587	0.531
InceptionResnet	0.416	0.346	0.402	0.376	0.545	0.473	0.537	0.489
Nasnet	0.436	0.362	0.408	0.397	0.615	0.524	0.611	0.563
MVEC	0.482	0.362	0.506	0.479	0.605	0.454	0.644	0.564
CC	0.459	0.378	N/A	N/A	0.533	0.430	N/A	N/A
MVnet	0.482	0.432	N/A	N/A	0.590	0.491	N/A	N/A
MVnet-retrain	0.488	0.449	N/A	N/A	0.594	0.495	N/A	N/A

Table E.3: Results on fine-grained recognition datasets.
(N/A: *incompatible methods*)

	Flowers				Birds			
	JULE		Agg		JULE		Agg	
	nmi	pur	nmi	pur	nmi	pur	nmi	pur
VGG16	0.704	0.658	0.644	0.619	0.503	0.158	0.557	0.309
VGG19	0.677	0.642	0.646	0.649	0.519	0.192	0.557	0.310
Inception	0.759	0.718	0.677	0.674	0.542	0.199	0.569	0.305
Xception	0.705	0.675	0.670	0.675	0.609	0.278	0.644	0.417
Resnet50	0.718	0.690	0.708	0.675	0.450	0.129	0.529	0.261
Densenet121	0.833	0.801	0.799	0.776	0.562	0.193	0.625	0.387
Densenet169	0.831	0.815	0.797	0.782	0.529	0.184	0.605	0.373
Densenet201	0.810	0.795	0.767	0.751	0.540	0.184	0.634	0.394
InceptionResnet	0.598	0.535	0.562	0.535	0.470	0.146	0.516	0.232
Nasnet	0.662	0.584	0.578	0.525	0.500	0.180	0.547	0.261
MVEC	0.850	0.758	0.798	0.731	0.589	0.178	0.665	0.343
CC	0.762	0.712	N/A	N/A	0.569	0.229	N/A	N/A
MVnet	0.879	0.860	N/A	N/A	0.662	0.271	N/A	N/A
MVnet-retrain	0.834	0.766	N/A	N/A	0.616	0.273	N/A	N/A

Table E.4: Results on face recognition datasets.
(N/A: *incompatible methods*)

	UMist				FEI			
	JULE		Agg		JULE		Agg	
	nmi	pur	nmi	pur	nmi	pur	nmi	pur
VGG16	0.891	0.802	0.689	0.550	0.924	0.775	0.897	0.753
VGG19	0.874	0.798	0.740	0.630	0.934	0.798	0.915	0.789
Inception	0.841	0.722	0.760	0.616	0.953	0.854	0.941	0.860
Xception	0.835	0.732	0.731	0.609	0.953	0.850	0.928	0.833
Resnet50	0.949	0.908	0.717	0.577	0.953	0.869	0.919	0.804
Densenet121	0.906	0.845	0.684	0.553	0.959	0.895	0.928	0.827
Densenet169	0.933	0.880	0.734	0.602	0.960	0.879	0.926	0.823
Densenet201	0.904	0.819	0.720	0.600	0.957	0.861	0.942	0.865
InceptionResnet	0.889	0.803	0.775	0.642	0.913	0.748	0.892	0.738
Nasnet	0.886	0.809	0.725	0.558	0.946	0.819	0.922	0.798
MVEC	0.942	0.877	0.763	0.610	0.966	0.903	0.949	0.865
CC	0.936	0.870	N/A	N/A	0.960	0.882	N/A	N/A
MVnet	0.966	0.922	N/A	N/A	0.962	0.891	N/A	N/A
MVnet-retrain	0.984	0.967	N/A	N/A	0.963	0.893	N/A	N/A

Appendix F

KL divergence between Gaussian trajectories

Constructive proof of formula (7.34)

F.1 Problem statement

This appendix presents a method to compute the Kullback-Leibler divergence (D_{KL}) between two Gaussian trajectory distributions, ρ and $\widetilde{\rho}$. The initial states of these two trajectories are drawn from the same distribution

$$\rho(x_0) = \mathcal{N}(\mu_{x_0},\ \Sigma_{x_0}), \tag{F.1}$$

where $\mathcal{N}(\mu_{x_0}, \Sigma_{x_0})$ is the normal distribution of mean vector μ_{x_0} and covariance matrix Σ_{x_0}. We also assume that these two trajectory distributions share the same dynamics

$$\rho(x_{t+1}|x_t, u_t) = \mathcal{N}(F_{xu_t} xu_t + f_t,\ \Sigma_t^{\text{dyn}}), \tag{F.2}$$

where x_t and u_t are respectively the state and action vectors at time step t and $xu_t = [x_t^T, u_t^T]^T$. In the context of learnt iLQG introduced in Chapter 7, this assumption is justified the small deviation rule for policy updates. Finally, both trajectory generate actions by following different time-varying linear-Gaussian controllers

$$\Pi(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, \Sigma_{u_t}), \tag{F.3}$$

$$\widetilde{\Pi}(u_t|x_t) = \mathcal{N}(\widetilde{K_t} x_t + \widetilde{k}_t, \widetilde{\Sigma}_{u_t}). \tag{F.4}$$

With these notations, the trajectory distributions can be written

$$\rho(\tau) = \rho(x_0) \prod_{t=0}^{T-1} \Pi(u_t|x_t)\rho(x_{t+1}|x_t, u_t), \tag{F.5}$$

$$\widetilde{\rho}(\tau) = \rho(x_0) \prod_{t=0}^{T-1} \widetilde{\Pi}(u_t|x_t)\rho(x_{t+1}|x_t, u_t). \tag{F.6}$$

F.2 Trajectories divergence in terms of conditional action distributions divergences

In ([Montgomery and Levine, 2016]), the authors show that KL divergence between the two trajectory distributions can be written as

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = \sum_{t=0}^{T-1} \mathbb{E}_{\rho(x_t)} \left[D_{KL}(\Pi(u_t|x_t)||\widetilde{\Pi}(u_t|x_t)) \right], \tag{F.7}$$

where $\rho(x_t)$ is the probability distribution of the state at time step t under the controller Π. Their proof is the following:

Proof of Formula (F.7). By definition of the KL divergence, we can write

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = \mathbb{E}_{\rho(\tau)} \left[\ln(\rho(\tau)) - \ln(\widetilde{\rho}(\tau)) \right]. \tag{F.8}$$

Then, by expanding the trajectories (F.5) and (F.6) in (F.8) and by cancelling the initial state and dynamics, we get

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = \mathbb{E}_{\rho(\tau)} \left[\sum_{t=0}^{T-1} \ln(\Pi(u_t|x_t)) - \ln(\widetilde{\Pi}(u_t|x_t)) \right]. \tag{F.9}$$

By linearity of the expectation, (F.9) becomes

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = \sum_{t=0}^{T-1} \mathbb{E}_{\rho(x_t,u_t)} \left[\ln(\Pi(u_t|x_t)) - \ln(\widetilde{\Pi}(u_t|x_t)) \right]. \tag{F.10}$$

Since $\rho(x_t, u_t)$ denotes the joint distribution of states and actions at time step t for trajectory distribution ρ, it can be written as $\rho(x_t, u_t) = \rho(x_t)\Pi(u_t|x_t)$. Hence, (F.10) becomes

$$D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = \sum_{t=0}^{T-1} \mathbb{E}_{\rho(x_t)} \left[\mathbb{E}_{\Pi(u_t|x_t)} \left[\ln(\Pi(u_t|x_t)) - \ln(\widetilde{\Pi}(u_t|x_t)) \right] \right]. \tag{F.11}$$

Finally, we note that the definition of the KL divergence between Π and $\widetilde{\Pi}$ at time step t is defined by

$$D_{KL}(\Pi(u_t|x_t)||\widetilde{\Pi}(u_t|x_t)) = \mathbb{E}_{\Pi(u_t|x_t)}\left[\ln(\Pi(u_t|x_t)) - \ln(\widetilde{\Pi}(u_t|x_t))\right]. \tag{F.12}$$

Substituting (F.12) into (F.11) leads to Formula (F.7).

F.3 Expected conditional actions distributions divergence in terms of state distributions

For all t in $\{0, ..., T\}$, the expected value of the KL divergence between the conditional action distributions in (F.7) can be expressed by

$$\mathbb{E}_{\rho(x_t)}\left[D_{KL}(\Pi(u_t|x_t)||\widetilde{\Pi}(u_t|x_t))\right] = \frac{1}{2}\left[\mathrm{Tr}(A_t\Sigma_{x_t}) + \mu_{x_t}^T A_t \mu_{x_t} + 2\mu_{x_t}^T b_t + c_t\right], \tag{F.13}$$

where μ_{x_t} and Σ_{x_t} are respectively the mean and variance of the state distribution at time step t (see Section F.4), and where

$$\begin{aligned}
A_t &= (\widetilde{K_t} - K_t)^T \widetilde{\Sigma}_{u_t}^{-1} (\widetilde{K_t} - K_t), \\
b_t &= (\widetilde{K_t} - K_t)^T \widetilde{\Sigma}_{u_t}^{-1} (\widetilde{k_t} - k_t), \\
c_t &= \alpha_t + \beta_t, \\
\alpha_t &= \ln \frac{|\widetilde{\Sigma}_{u_t}|}{|\Sigma_{u_t}|} + \mathrm{Tr}(\widetilde{\Sigma}_{u_t}^{-1}\Sigma_{u_t}) - \dim(u_t), \\
\beta_t &= (\widetilde{k_t} - k_t)^T \widetilde{\Sigma}_{u_t}^{-1} (\widetilde{k_t} - k_t).
\end{aligned} \tag{F.14}$$

And where Tr(.) represents the trace of a matrix, |.| its determinant and dim(.) the dimension of a vector.

Proof of Formula (F.13). Using the formula of KL divergence between two multivariate Gaussian distributions we have

$$\begin{aligned}
\mathbb{E}_{\rho(x_t)}\left[D_{KL}(\Pi(u_t|x_t)||\widetilde{\Pi}(u_t|x_t))\right] = \mathbb{E}_{\rho(x_t)}\Bigg[\frac{1}{2}\Big[\ln \frac{|\widetilde{\Sigma}_{u_t}|}{|\Sigma_{u_t}|} + \mathrm{Tr}(\widetilde{\Sigma}_{u_t}^{-1}\Sigma_{u_t}) - \dim(u_t) \\
+ \left[(\widetilde{K_t} - K_t)x_t + (\widetilde{k_t} - k_t)\right]^T \widetilde{\Sigma}_{u_t}^{-1}\left[(\widetilde{K_t} - K_t)x_t + (\widetilde{k_t} - k_t)\right]\Big]\Bigg].
\end{aligned} \tag{F.15}$$

By linearity of the expectation and because α_t, defined in (F.14), does not depend on x_t, we can write

$$\mathbb{E}_{\rho(x_t)}\left[D_{KL}(\Pi(u_t|x_t)||\widetilde{\Pi}(u_t|x_t))\right] = \frac{1}{2}(\alpha_t + \Theta_t), \tag{F.16}$$

where

$$\Theta_t = \mathbb{E}_{\rho(x_t)}\left[\left[(\widetilde{K_t} - K_t)x_t + (\widetilde{k_t} - k_t)\right]^T \widetilde{\Sigma}_{u_t}^{-1}\left[(\widetilde{K_t} - K_t)x_t + (\widetilde{k_t} - k_t)\right]\right]. \quad \text{(F.17)}$$

By expending (F.17), we can leverage the linearity of expectation and the symmetry of covariance matrices to obtain

$$\begin{aligned}\Theta_t = &\mathbb{E}_{\rho(x_t)}\left[x_t^T(\widetilde{K_t} - K_t)^T\widetilde{\Sigma}_{u_t}^{-1}(\widetilde{K_t} - K_t)x_t\right] \\ &+ 2\mathbb{E}_{\rho(x_t)}\left[x_t^T(\widetilde{K_t} - K_t)^T\widetilde{\Sigma}_{u_t}^{-1}(\widetilde{k_t} - k_t)\right] \\ &+ (\widetilde{k_t} - k_t)^T\widetilde{\Sigma}_{u_t}^{-1}(\widetilde{k_t} - k_t)\end{aligned} \quad \text{(F.18)}$$

Then, we use the formula of the expectation of a quadratic form, and with the notations (F.14), we get

$$\Theta_t = \mathrm{Tr}(A_t\Sigma_{x_t}) + \mu_{x_t}^T A_t \mu_{x_t} + 2\mu_{x_t}^T b_t + \beta_t \quad \text{(F.19)}$$

The final formula (F.13) is obtained by injecting (F.19) into (F.16).

F.4 Recursive computation of state distributions

With Equation (F.13), we now only need the expressions of μ_{x_t} and Σ_{x_t} for all t in $\{0, ..., T\}$ to be able to compute $D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau))$. Starting from the known moments of $\rho(x_0)$, these can be computed recursively:

$$\mu_{x_{t+1}} = F_{xu_t}\mu_{xu_t} + f_t, \quad \text{(F.20)}$$

$$\Sigma_{x_{t+1}} = F_{xu_t}\Sigma_{xu_t}(F_{xu_t})^T + \Sigma_t^{dyn}, \quad \text{(F.21)}$$

where f_t is the constant term in the dynamics regression, which in practice corresponds to the value of state of the current nominal trajectory, and where

$$\mu_{xu_t} = \begin{bmatrix}\mu_{x_t} \\ \hline K_t\mu_{x_t} + k_t\end{bmatrix}, \quad \text{(F.22)}$$

$$\Sigma_{xu_t} = \left[\begin{array}{c|c}\Sigma_{x_t} & \Sigma_{x_t}K_t^T \\ \hline K_t\Sigma_{x_t} & K_t\Sigma_{x_t}K_t^T + \Sigma_{u_t}\end{array}\right]. \quad \text{(F.23)}$$

Proof of Formulas (F.20) and (F.21). Let's assume that for time step t, μ_{x_t} and Σ_{x_t}, the mean vector and covariance matrix of the state distribution, are known. Then, we have

$$\begin{aligned}\mathbb{E}_{\rho(x_t,u_t)}[u_t] &= \mathbb{E}_{\rho(x_t,u_t)}[K_t x_t + k_t + \omega_t] \\ &= K_t\mu_{x_t} + k_t.\end{aligned} \quad \text{(F.24)}$$

Where $\omega_t = \mathcal{N}(0, \Sigma_{u_t})$ is the noise of the stochastic controller Π. Likewise, because x_t and ω_t are independent, we also have

$$\begin{aligned} \mathrm{Var}_{\rho(x_t,u_t)}[u_t] &= \mathrm{Var}_{\rho(x_t,u_t)}[K_t x_t + k_t] + \mathrm{Var}_{\rho(x_t,u_t)}[\omega_t] \\ &= K_t \Sigma_{x_t} K_t^T + \Sigma_{u_t}, \end{aligned} \tag{F.25}$$

and

$$\begin{aligned} \mathrm{Cov}_{\rho(x_t,u_t)}[x_t, u_t] &= \mathbb{E}_{\rho(x_t,u_t)}[(x_t - \mathbb{E}[x_t])(u_t - \mathbb{E}[u_t])] \\ &= \mathbb{E}\left[(x_t - \mathbb{E}[x_t])(K_t(x_t - \mathbb{E}[x_t]) + \omega_t - \mathbb{E}[\omega_t])\right] \\ &= \mathbb{E}\left[(x_t - \mathbb{E}[x_t])(x_t - \mathbb{E}[x_t])^T K_t^T\right] + \mathbb{E}\left[(x_t - \mathbb{E}[x_t])(\omega_t - \mathbb{E}[\omega_t])^T\right] \\ &= \Sigma_{x_t} K_t^T. \end{aligned} \tag{F.26}$$

Where the last line comes from the independence between x_t and ω_t ($\mathrm{Cov}[x_t, \omega_t] = 0$).

Finally, from (F.24), (F.25) and (F.26), we can write μ_{xu_t} and Σ_{xu_t} in the form of (F.22) and (F.23). Then, if Ω_t denotes a Gaussian noise drawn from $\mathcal{N}(0, \Sigma_t^{\mathrm{dyn}})$ we can compute

$$\begin{aligned} \mu_{x_{t+1}} &= \mathbb{E}_{\rho(x_t,u_t)}[x_{t+1}] \\ &= \mathbb{E}_{\rho(x_t,u_t)}[F_{xu_t} x u_t + f_t + \Omega_t] \\ &= F_{xu_t} \mu_{xu_t} + f_t, \end{aligned} \tag{F.27}$$

and, by independence of xu_t and Ω_t,

$$\begin{aligned} \Sigma_{x_{t+1}} &= \mathrm{Var}[F_{xu_t} x u_t + f_t + \Omega_t] \\ &= F_{xu_t} \Sigma_{xu_t} (F_{xu_t})^T + \Sigma_t^{dyn}. \end{aligned} \tag{F.28}$$

F.5 Summary algorithm

To conclude this appendix, we propose the following pseudo-code for the computation of the KL divergence between Gaussian trajectories.

Inputs : $\mu_{x_0}, \Sigma_{x_0}, \left\{F_{xu_t}, f_t, \Sigma_t^{\text{dyn}}, K_t, k_t, \Sigma_{u_t}, \widetilde{K}_t, \widetilde{k}_t, \widetilde{\Sigma}_{u_t} \ ; \ \forall t \in \{0, ..., T\}\right\}$
Outputs : $D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau))$

Initialization: $D = 0, \mu_{x_t} = \mu_{x_0}, \Sigma_{x_t} = \Sigma_{x_0}$
for $t \in \{0, ..., T-1\}$ **do**
  Compute A_t, b_t and c_t using (F.14)
  Compute $\mathbb{E}_{\rho(x_t)}\left[D_{KL}(\Pi(u_t|x_t)||\widetilde{\Pi}(u_t|x_t))\right]$ using (F.13)
  Compute μ_{xu_t} and Σ_{xu_t} using (F.22) and (F.23)
  Compute $\mu_{x_{t+1}}$ and $\Sigma_{x_{t+1}}$ using (F.20) and (F.21)

  $D \leftarrow D + \mathbb{E}_{\rho(x_t)}\left[D_{KL}(\Pi(u_t|x_t)||\widetilde{\Pi}(u_t|x_t))\right]$
  $\mu_{x_t} \leftarrow \mu_{x_{t+1}}$
  $\Sigma_{x_t} \leftarrow \Sigma_{x_{t+1}}$
end

Return : $D_{KL}(\rho(\tau)||\widetilde{\rho}(\tau)) = D$

Algorithm 1: Pseudo-code for computation of KL divergence between two Gaussian trajectories.

Annexe G

Résumé étendu en français

Cette annexe propose une synthèse du mémoire de thèse rédigée en langue française. Dans un soucis de concision, seule la méthodologie et les principaux résultats sont présentés. On notera que, par souci de cohérence du manuscrit, les acronymes en anglais sont utilisés dans ce résumé. Les traductions en français sont rappelées dans le glossaire, au début de la thèse.

Les travaux présentés ont été financés en partie par le projet européen ColRobot (robotique collaborative pour l'assemblage et la préparation de kits dans le contexte de la fabrication intelligente). Ce projet fait partie du programme de l'union européenne pour la recherche et l'innovation à horizon 2020 sous le financement No.688807.

Les travaux ont été menés en grande partie au sein du LISPEN (Laboratoire d'Ingénierie des Systèmes Physiques et Numériques), dans l'équipe des Arts et Métiers de Lille. Une partie des recherches présentées dans ce mémoire ont également été conduites au Robot Learning Lab, au Georgia Institute of Technology, à Atlanta. Cette période à Atlanta a été financée par Fulbright Hauts de France et la fondation Arts et Métiers.

Table des matières

Liste des figures

Liste des tableaux

G.1 Introduction : vers des robots plus polyvalents

G.1.1 L'apprentissage automatique au service de l'adaptabilité des robots

Un robot industriel est un mécanisme motorisé complexe, conçu de manière à pouvoir être programmé pour différentes tâches. Cette flexibilité mécanique lui permet d'exécuter une large gamme de mouvements, ce qui a permis un essor important de la robotique dans les usines au cours des dernières décennies. Pendant de nombreuses années, la recherche autour de la programmation robotique s'est principalement concentrée sur l'étude de la cinématique et du contrôle de ces machines complexes. Cependant, cette vision du robot comme une simple machine de précision, dénuée de capacité de perception et d'adaptation, a limité leur utilisation à des environnements cloisonnés et à des tâches très répétitives, nécessitant de gros volumes de production pour être rentable.

Ces dernières années ont vu émerger un nouveau contexte industriel, appelé *industrie 4.0*, qui propose une vision plus souple des unités de production, nécessitant des robots plus polyvalents, capables de collaborer avec d'autres machines ainsi que des humains. Avec l'apparition de ce nouveau besoin, les systèmes robotisés modernes ont été équipés de nombreux capteurs afin de leur donner des capacités de perception et ainsi leur permettre de résoudre une plus grande diversité de tâches.

Dans cette thèse, nous proposons de mener une réflexion autour de la généralisation des applications robotiques à des plages de fonctionnement plus larges. De part ses nombreux récents succès en terme de modélisation de systèmes complexes, l'apprentissage automatique (ML) est un cadre intéressant pour mener cette étude. En effet, le ML permet de définir des tâches de manière plus systémique et a démontré des résultats moins dépendants d'une tâche donnée. Une introduction à l'apprentissage automatique est proposée en Annexe A. Le but de cette thèse est d'introduire de nouvelles méthodes de ML, permettant d'élargir la gamme de fonctionnement et le domaine d'applicabilité de diverses applications robotiques, et ainsi de mieux exploiter la richesse dans la conception de ces machines. Les améliorations introduites dans ce manuscrit sont présentées principalement dans le contexte du tri robotisé, introduit ci-après.

G.1.2 Tri robotique non supervisé

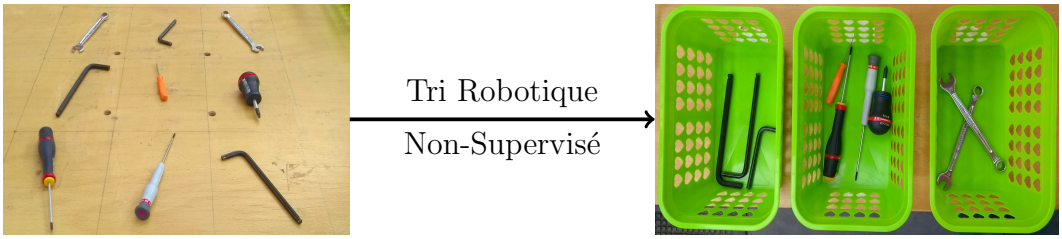
Malgré que le tri automatisé soit parmi les plus anciennes applications de robotique industrielle, il reste une tâche complexe. En effet, le schéma complet d'une application

de tri est composé de nombreuses sous-tâches :

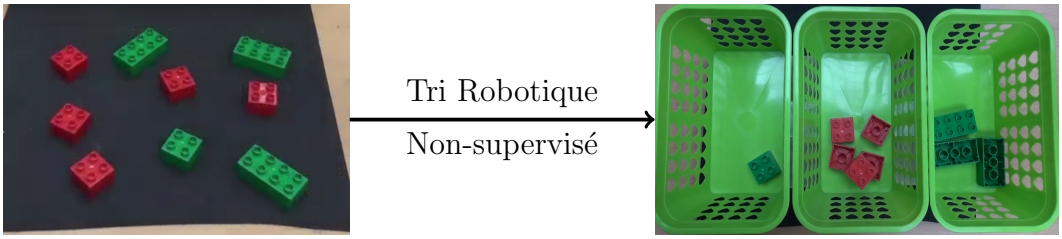
- segmentation de scène,
- identification et localisation d'objets,
- collecte de données sur les différents objets,
- préhension,
- prise de décision sur la façon de trier les objets.

Dans cette thèse, nous nous intéressons principalement au module de prise de décision et proposons une approche plus générique que ce qui est fait classiquement. En définissant la tâche de tri de manière non-supervisée, l'application est alors programmée pour fonctionner avec tous types d'objets au lieu d'être spécialisée pour certaines catégories.

En d'autres termes, le problème étudié dans cette thèse consiste à résoudre un problème de partitionnement de données (clustering), appliqué à des objets physiques. Pour cette raison, ce problème est appelé tri robotique non-supervisé (URS). Deux exemples d'URS sont présentés en Figure G.1 et des développements pour résoudre ce problème sont proposés dans les chapitres 2 à 6.



(a) Outils dans un poste de travail



(b) Briques Lego de différentes formes et couleurs

Figure G.1: Deux exemples de problèmes de Tri Robotique Non-Supervisé (URS)

Dans la suite de ce résumé, les différents développements effectués autour de l'URS sont présentés de manière synthétique. Les résultats clés sont également rapportés et discutés. Pour des explications plus complètes, le lecteur peut se référer au corps du texte de la thèse, en langue anglaise, qui présente en détail les méthodes employées et discute les résultats en profondeur.

G.2 Partitionnement de données à partir de caractéristiques de couleur et de forme.

G.2.1 Définition du problème et des difficultés

Dans ce chapitre, une première mise en oeuvre d'URS est proposée dans laquelle un robot doit trier des objets à partir de caractéristiques de couleur et de forme. Cette représentation d'objet est utilisée fréquemment dans la littérature du tri robotisé. Cette application est non-supervisée, ce qui signifie qu'aucune règle concernant la manière de trier n'est prédéfinie et qu'aucun exemple de tri correct n'est fourni au préalable. Un exemple d'une telle tâche est présenté en Figure G.2.

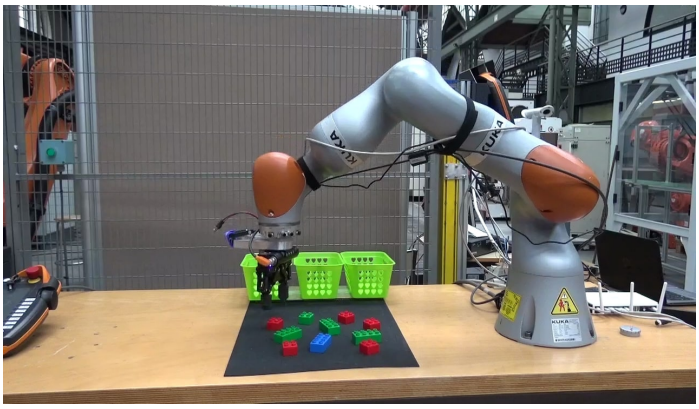


Figure G.2: Robot utilisant notre algorithme des K-moyennes pondéré par rapport des écarts pour trier des objets à partir de caractéristiques de couleur et de forme. Une vidéo de démonstration peut être vue à l'url suivant : `https://www.youtube.com/watch?v=korkcYs1EHM`

L'approche proposée pour résoudre ce problème d'URS se base sur l'algorithme de partitionnement de données des K-moyennes. Cependant, le type de données étudié dans ce chapitre ainsi que les conditions de fonctionnement de l'application présentent de nombreuses difficultés pour cet algorithme. Tout d'abord, les caractéristiques de forme et de couleur ont des ordres de grandeurs très différents, ce qui impose de normaliser les données. En parallèle, les objets sont triés dans un environnement industriel dans lequel la lumière n'est pas maîtrisée, ce qui implique une grande dispersion des données (Figure G.3). La combinaison de cette forte dispersion avec la nécessité d'une normalisation impose l'utilisation d'algorithmes des K-moyennes pondérées, une variante des de la méthode des K-moyennes classique, pour palier à la perte d'information due à la normalisation. Enfin, les caractéristiques étudiées appartiennent à différentes échelles de mesure : les caractéristiques de couleur (RGB) sont des variables

d'intervalles alors que les distances sont des variables de rapports. Cette spécificité empêche l'utilisation du coefficient de variation (CV) dans la pondération des K-moyennes. Pour résoudre la combinaison de ces trois difficultés, un nouvel algorithme, appelé *algorithme des K-Moyennes Pondérées par Rapport des Écarts* (GRKM), est proposé. Une présentation résumée de cet algorithme ainsi que les principaux résultats sur la tâche étudiée sont présentés dans cette annexe. Pour une présentation plus en profondeur, le lecteur peut se référer au Chapitre 2 de cette thèse.

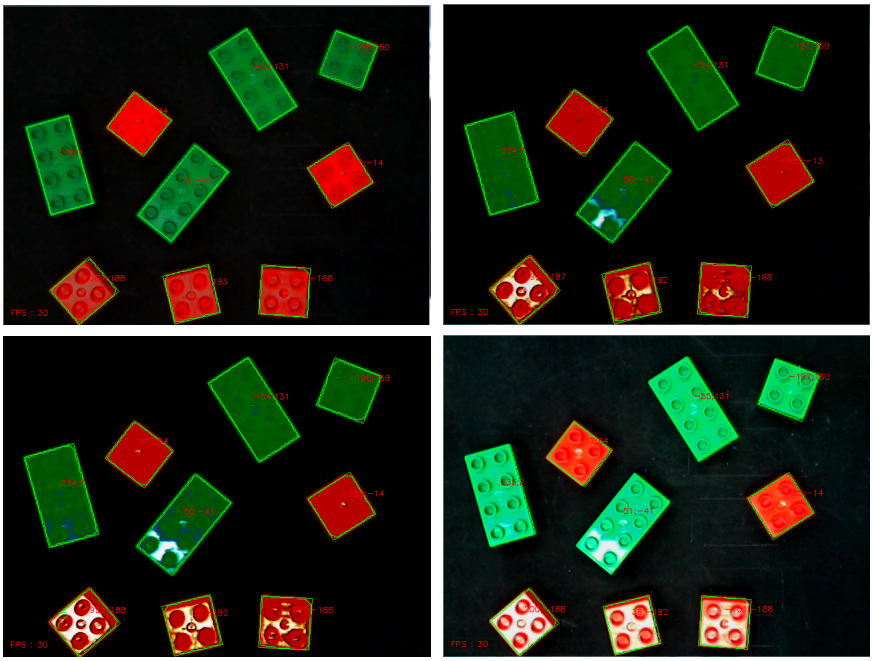


Figure G.3: Exemples de jeux de données pour différentes conditions de lumière.

G.2.2 Approche proposée et résultats

L'algorithme GRKM proposé dans ce chapitre se base sur le calcul des écarts entre chaque paire de points consécutifs pour chaque dimension. Le rapport de l'écart maximum sur l'écart moyen est ensuite calculé pour chaque dimension puis utilisé pour créer des poids qui sont réinjectés dans les données normalisées afin de trouver les dimensions les plus importantes. L'utilisation des écarts est motivée par l'intuition qu'un grand écart pour une dimension peut signifier que les valeurs viennent de deux distributions différentes et donc que les points appartiennent à deux classes différentes. L'avantage d'utiliser des données relatives est qu'elles peuvent fonctionner pour différents types d'échelles. A travers de nombreux exemples simulés, qui sont présentés dans le corps du texte, la pertinence de cette approche est démontrée

pour différents problèmes de partitionnement. Cette approche est ensuite testée sur l'application robotique de tri de blocs Lego. L'étude est menée sur un cas pratique comportant trois catégories, présentées en Figure G.3. Cette expérience de tri est répétée 98 fois avec différentes conditions de lumière et différentes dispositions des objets. Les résultats obtenus sont comparés avec les méthodes de KM classique et de CVKM. Ils sont présentés en Figure G.4, où l'on peut également voir une étude de robustesse quand une des composantes de couleur se rapproche de zéro (histogramme de droite).

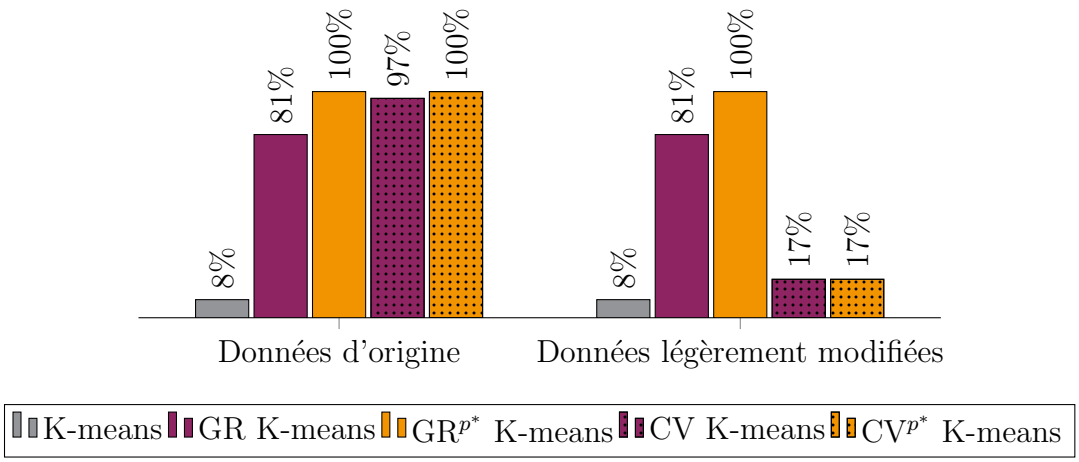


Figure G.4: Pourcentage d'expériences avec aucune classification erronée. L'expérience a été reproduite 98 fois sous différentes conditions de lumière et avec différents arrangements des blocs Lego. Le taux d'erreur présenté est moyenné sur ces 98 tentatives.

Pour étudier la qualité des poids proposés par rapport à d'autres méthodes de la littérature, une étude sur la stabilité de ces poids par exponentiation est également proposée. Les résultats peuvent être trouvés sur la Figure G.5.

Les résultats expérimentaux démontrent la robustesse de l'approche GRKM face aux KM classiques ainsi qu'au CVKM. En effet, pour des données bruitées, avec des conditions de lumière variables et quand une des composantes RGB se rapproche de zéro, les poids GR semblent mieux appropriés. De plus, les poids obtenus par GRKM sont plus stables pour des exposants plus grands, ce qui révèle un meilleur équilibrage des poids que pour la méthode CVKM. Le GRKM est également testé avec des jeux de données classiques de la littérature (Fischer Iris et Wine Dataset). Les résultats obtenus, commentés plus en détails dans le corps de la thèse, mettent en évidence l'influence du conditionnement des données dans le choix d'une approche.

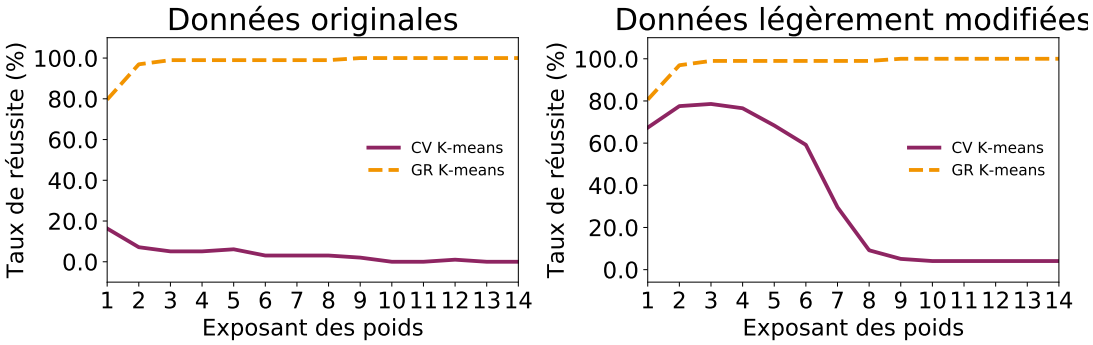


Figure G.5: Comparaison de la stabilité des poids GR et CV. Étude de l'influence des exposants pour le partitionnement de briques Lego par K-means pondéré exponentiel.

G.2.3 Conclusions et limitations

Dans ce chapitre, un premier exemple d'URS où les objets sont triés à partir de caractéristiques de couleur et de forme a été introduit. Les jeux de données générés par cette application présentent de nombreuses difficultés pour le problème de partitionnement de données. Ainsi, pour outrepasser le fait que les données sont bruitées et que certaines dimensions des vecteurs de caractéristiques sont des variables d'intervalle, un nouvel algorithme de K-moyennes pondérées a été introduit. Cette méthode, appelée K-moyennes pondérées par rapport des écarts, a permis d'obtenir de meilleurs résultats sur une mise en oeuvre physique de l'URS. Des perspectives d'utilisation et d'amélioration concernant l'algorithme GRKM sont présentées dans le Chapitre G.9.

Bien qu'ayant motivé le développement du GRKM, la représentation des objets choisie dans ce chapitre est très spécifique et ne permet pas de trier n'importe quel type d'objets. Par exemple, il est peu probable que les différents modèles de tournevis sur la Figure G.1 soient correctement classés avec une représentation de forme et couleur. En effet, un problème plus générique d'URS requiert l'utilisation de caractéristiques ayant un niveau d'abstraction plus élevé, permettant de révéler la nature sémantique des différents objets étudiés. De ce fait, dans les chapitres à venir, le module de prise de décision de l'URS est vu comme un problème de partitionnement d'images (IC), où chaque objet est représenté par une image le représentant. Les récents succès obtenus pour le transfert de connaissance depuis ImageNet vers d'autres tâches de vision par ordinateur motive l'utilisation de réseaux de convolutions profonds (CNN) pour extraire des caractéristiques des images du nouveau problème d'IC. Dans le prochain chapitre, une étude sur l'optimisation du transfert de connaissance d'ImageNet vers une nouvelle tâche non-supervisée est étudiée.

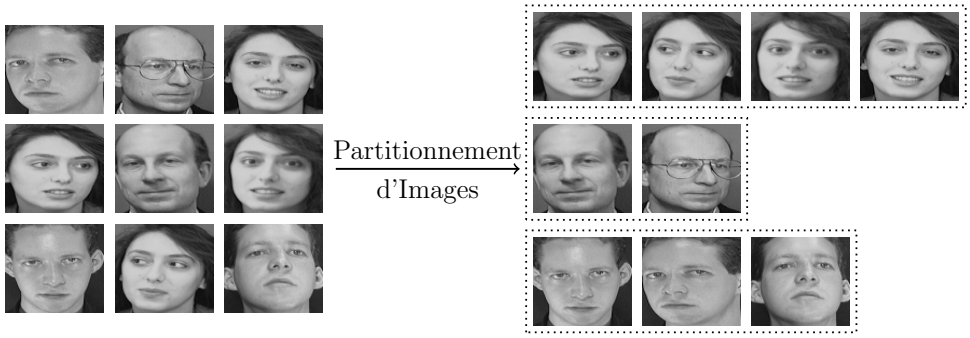
G.3 Extraction de caractéristiques par CNN pré-entraînés pour le partitionnement d'images.

G.3.1 Introduction et état de l'art

Pour résoudre le problème de tri robotique non-supervisé (URS) dans sa version la plus générique, les différents objets à classer sont représentés par des images. Après avoir collecté des photos des objets à trier, un problème d'URS devient un problème de partitionnement d'images (IC), dont des exemples d'entrées-sorties sont présentés en Figure G.6.

(a) VOC2007

(b) MIT



(c) ORL

Figure G.6: Définition du problème de partitionnement d'images. Exemples d'entrées et des sorties attendues associées pour trois jeux d'images naturelles.

Les articles récents qui adressent le problème d'IC se basent sur des caractéristiques extraites par des réseaux de convolution (CNN) pré-entraînés sur ImageNet. Ces caractéristiques sont ensuite utilisées en entrée d'un algorithme de partitionnement pour produire la classification finale. La plupart de ces articles propose des innovations sur les algorithmes de partitionnement mais ne justifie pas le choix du CNN pour l'extraction de caractéristiques. Dans ce chapitre, une étude est menée pour connaître l'influence du choix du CNN sur les résultats d'IC. L'étude proposée vise à savoir si

- le choix de l'architecture peut changer les résultats du partitionnement,
- certaines couches sont plus adaptées que d'autres pour l'extraction,
- il existe des interactions entre les types des données étudiés et les différentes d'architectures.

G.3.2 Expérience proposée et résultats

Pour étudier ces interactions, le schéma présenté en Figure G.7 est mis en oeuvre avec différents jeux de données (DS), architectures de CNN (NN), couches d'extractions (L), algorithmes de partitionnement (C) et métriques. Le but de cette étude est de trouver des règles pour choisir NN et L.

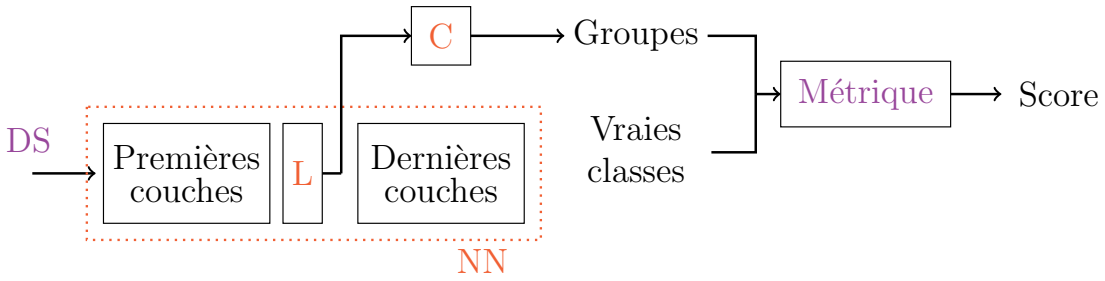


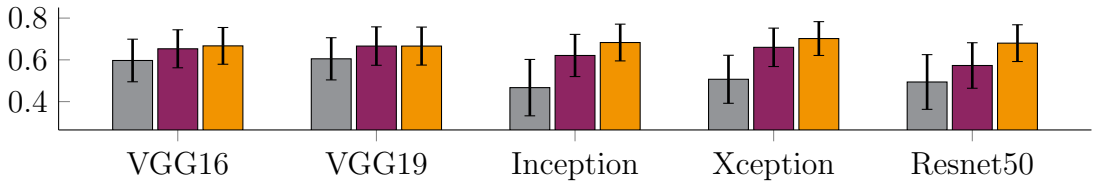
Figure G.7: Schéma descriptif de l'étude sur l'extraction de caractéristiques.

L'étude proposée est menée avec

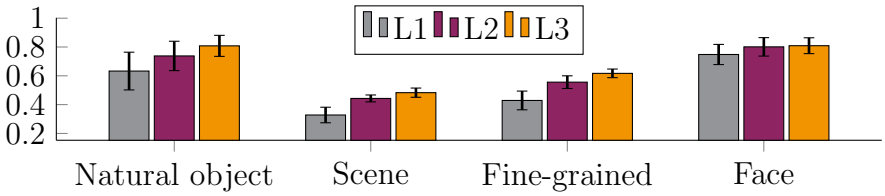
- 8 jeux de données, appartenant à 4 sous-tâches de partitionnement d'images,
- 5 architectures et 3 couches d'extraction pour chacune d'elles,
- 2 algorithmes de partitionnement,
- 2 métriques de validation externes.

Le descriptif exhaustif de tous les éléments de l'étude peut être trouvé dans le corps du texte.

Les résultats complets obtenus sont présentés en annexe B. Ces résultats sont synthétisés sur les Figures G.8, G.10 et G.9, où les moyennes et écarts types pour différents couples de paramètres sont représentés.



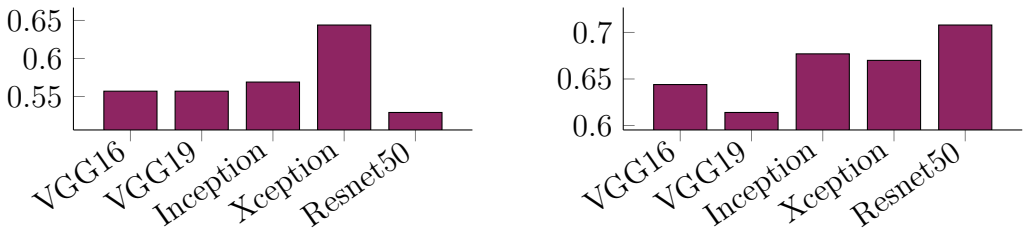
(a) Interaction couche/architecture
(moyenne et écart type sur les différentes tâches et algorithmes).



(b) Interaction couche/tâche
(moyenne et écart type sur les architectures, DS et algorithmes).

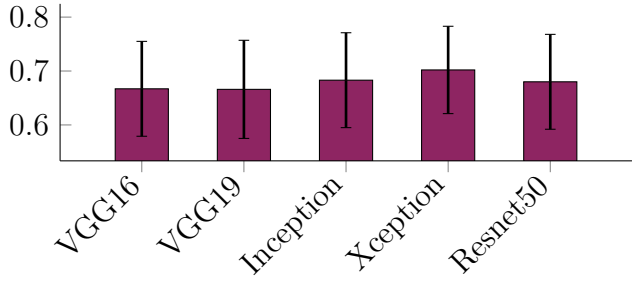
Figure G.8: Influence de la couche d'extraction sur le partitionnement (NMI).

Les résultats obtenus pour les différentes couches (Figure G.8) montrent que la dernière couche avant la classification finale ImageNet est la meilleure, quel que soit le réseau utilisé et quelle que soit la tâche. Ce résultat donne une première indication importante quant au choix de la couche d'extraction. En regardant la Figure G.9, il apparaît que le choix de l'architecture est également crucial pour obtenir de bons résultats. En revanche, celui-ci apparaît beaucoup moins trivial. Les écarts types étant plus grands que les différences moyennes de performance entre les réseaux, il est impossible de conclure sur le choix de l'architecture.

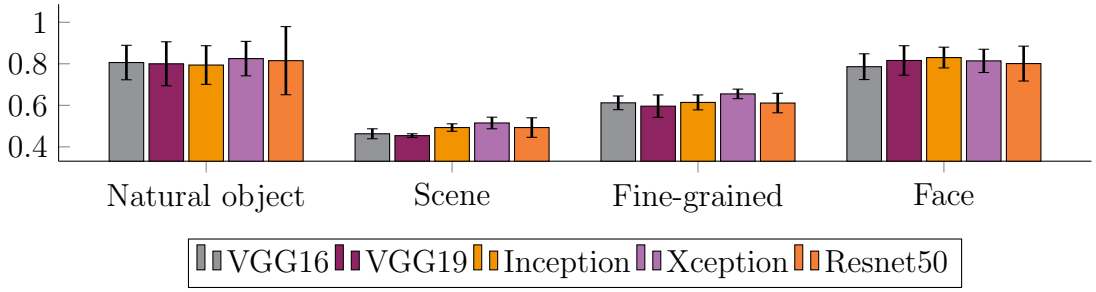


(a) Birds - Agglomerative clustering **(b)** Flowers - Agglomerative clustering

Figure G.9: Exemples où le choix de l'architecture du réseau de convolution est crucial pour la réussite du partitionnement.



(a) Moyenne et écart type sur les différentes tâches et algorithmes



(b) Interaction architecture/tâche
(moyenne et écart type sur les différents jeux de données et algorithmes)

Figure G.10: Influence du choix de l'architecture du réseau de convolution (L3) sur les résultats de partitionnement (NMI).

G.3.3 Conclusions

L'utilisation d'un CNN pré-entraîné pour extraire des caractéristiques est maintenant devenue une pratique courante pour résoudre des problèmes d'IC. Cependant, le choix de l'architecture et de la couche d'extraction est souvent arbitraire. Dans ce chapitre, nous avons mené des expériences sur huit jeux de données classiques de vision provenant de quatre sous-tâches de classification. Le premier résultat intéressant de cette étude est que la dernière couche avant le softmax semble produire les caractéristiques les plus intéressantes pour le partitionnement. Les expériences ont également démontré que le choix de l'architecture d'extraction de caractéristiques est crucial pour le tri non-supervisé. Cependant, les résultats obtenus ne permettent pas de conclure quant au choix d'une telle architecture. Pour pallier à ce problème, une nouvelle méthode ensembliste est proposée dans le chapitre suivant.

G.4 Améliorer le partitionnement d'images en utilisant plusieurs CNN pré-entraînés.

G.4.1 Introduction

L'utilisation de réseaux profonds pré-entraînés pour l'extraction de caractéristiques permet d'améliorer grandement les résultats d'IC. Cependant, les expériences menées dans le chapitre précédent ont démontré que le choix d'une bonne architecture pour un problème donné est particulièrement complexe. Une méthode pour s'affranchir de ce choix est introduite dans ce chapitre. En suivant l'intuition que différents réseaux pré-entraînés peuvent contenir des informations complémentaires, nous proposons d'utiliser plusieurs CNNs simultanément pour générer plusieurs représentations des données initiales. Cette approche revient à transformer le problème initial en un problème de partitionnement multi-vues (MVC). Le problème de MVC ainsi généré est résolu par la méthode de partitionnement profond JULE, que nous avons modifiée pour accepter des données multi-vues. Les résultats obtenus par la méthodologie proposée sont les meilleurs résultats connus dans la littérature pour plusieurs jeux de données. Cette approche présente également l'avantage de générer une représentation unifiée de basse dimension du jeu de données de départ.

G.4.2 Intuition sur l'utilisation de plusieurs CNNs

Dans le corps de la thèse, quelques justifications théoriques à propos de l'utilisation de plusieurs réseaux sont apportées. Une première expérience est ensuite menée sur le jeu de données UMist pour justifier de la complémentarité des différentes architectures. Elle consiste en une étude de performance de trois réseaux pour les différentes classes de UMist prises indépendemment. Les résultats sont présentés sur la Figure G.11, où l'on peut voir que les différentes architectures ne réagissent pas de la même manière pour différentes catégories : VGG16 et Densenet121 se comportent mieux avec la classe 4 de UMist, alors que InceptionResnet est l'architecture la plus performante pour ces données. Ces résultats suggèrent que les informations contenues dans chacun des réseaux sont complémentaires.

Afin de confirmer cette intuition, la méthode ensembliste MVEC, présentée sur la Figure G.12, est mise en oeuvre pour différents jeux de données. Pour chaque jeu de données, nous faisons varier le nombre d'architectures utilisées pour l'extraction et essayons toutes les permutations parmi 10 architectures étudiées. Les résultats de ces expériences sont présentés sur la Figure G.13, où il apparaît que pour tous les jeux

	NMI	PUR	FM	FM_{C_4}
InceptionResnet	**0.775**	**0.642**	**0.537**	0.442
VGG16	0.689	0.550	0.372	**0.653**
Densenet121	0.684	0.553	0.384	**0.700**

(a) InceptionResnet **(b)** VGG16 **(c)** Densenet121

Figure G.11: Visualisation t-SNE 2d des caractéristiques extraites par trois CNNs pré-entraînés pour le jeu de données UMist. Ces caractéristiques forment différentes vues des données qui sont *complémentaires*.

de données, augmenter le nombre de CNNs permet d'obtenir de meilleurs résultats NMI. Un plus grand nombre de réseaux permet également de diminuer la variance des résultats, ce qui signifie que la performance de l'algorithme de partitionnement est moins dépendante du choix des architectures. On peut également voir qu'avec une mise en oeuvre multi-GPUs, le temps de partitionnement ne croit pas linéairement avec le nombre de CNNs utilisés.

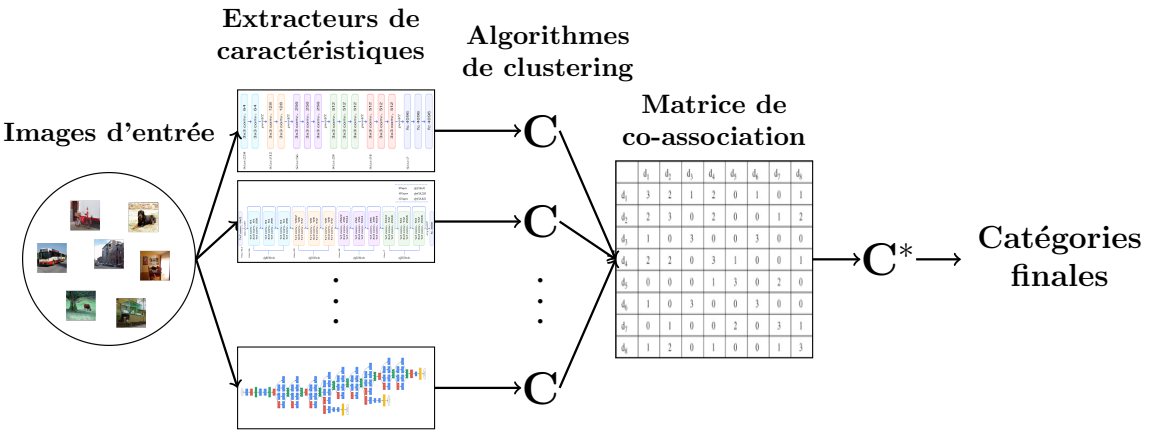


Figure G.12: Schéma de la méthode MVEC utilisée pour étudier l'influence de l'utilisation de plusieurs CNNs.

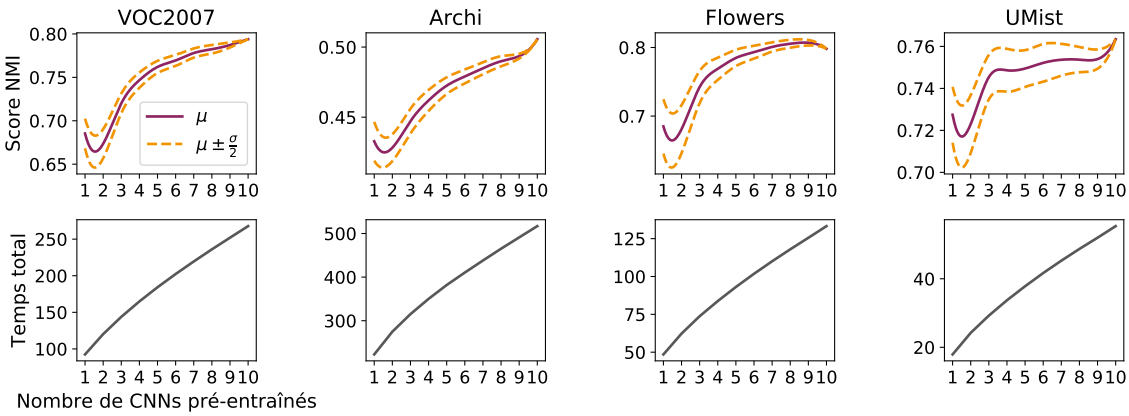


Figure G.13: Évolution du score NMI et du temps total (en secondes) pour différents nombres de CNNs pré-entraînés pour l'extraction de caractéristiques.

G.4.3 Méthode complète et résultats

Après avoir validé l'utilisation de plusieurs extracteurs de caractéristiques en parallèle, nous proposons de partitionner les données multi-vues générées avec la méthode de partitionnement profond JULE. Une architecture parallèle de perceptron multi-couches est définie et entraînée grâce à une version modifiée de JULE, qui fonctionne pour des données multi-vues. Le schéma complet de la méthode peut être trouvé sur la Figure G.14.

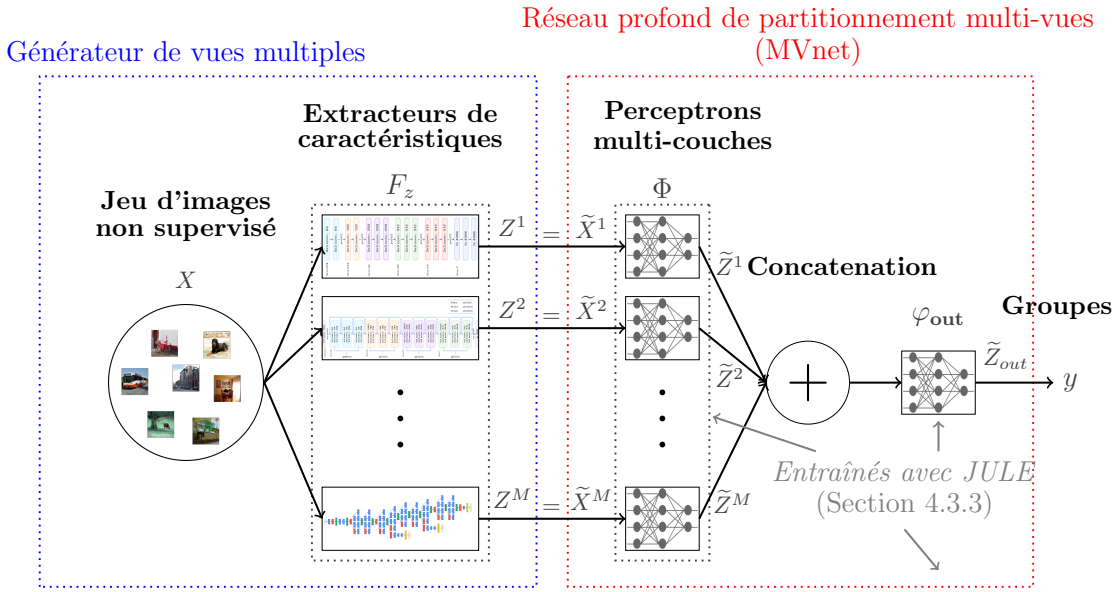
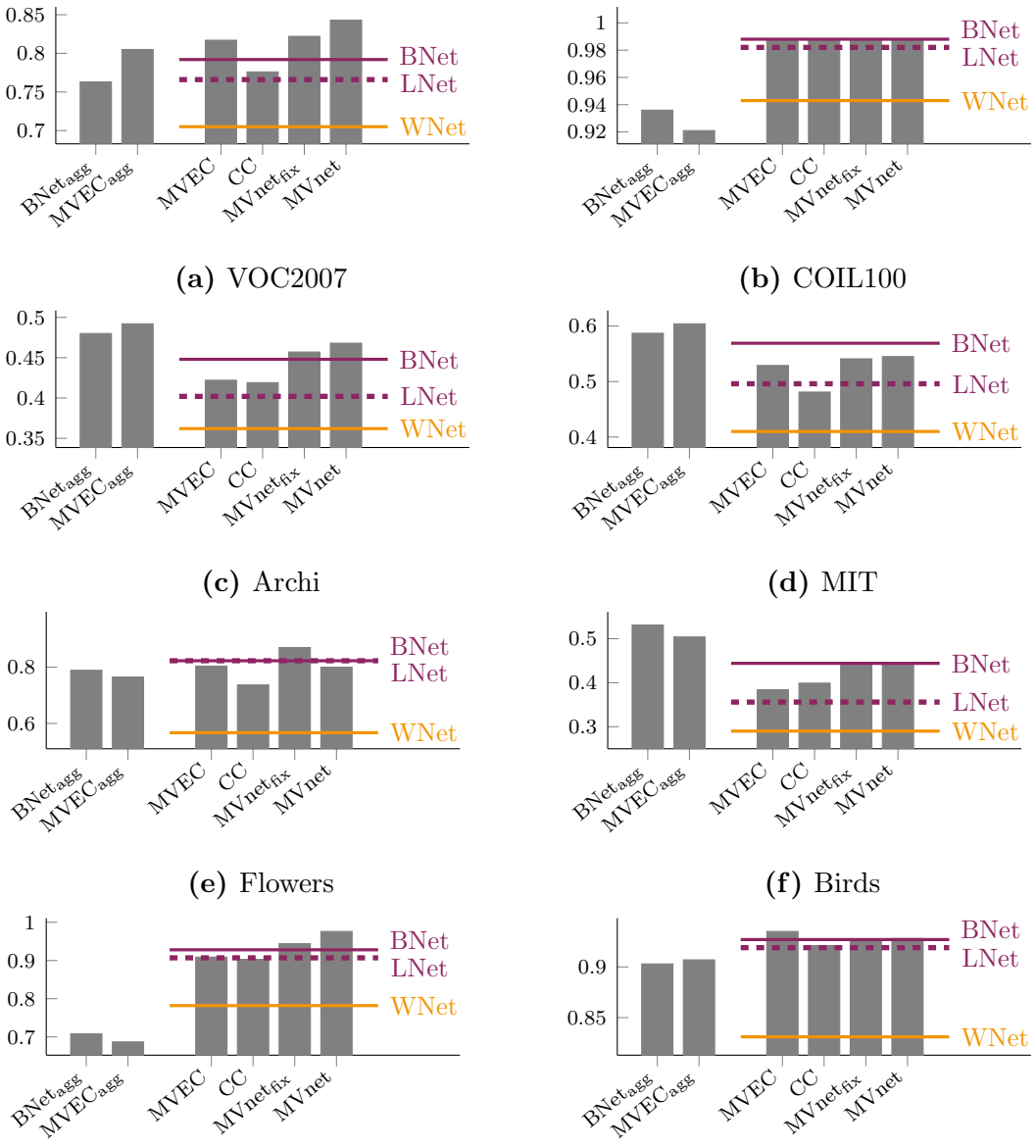


Figure G.14: Méthode proposée (DMVC) pour résoudre le problème de partitionnement d'images. Génération de vues multiples + Partitionnement multi-vues profond.

Cette méthode est ensuite testée sur huit jeux de données de classification d'images. Les résultats obtenus sont comparés avec ceux obtenus par une extraction simple (BNet, LNet et WNet), et ceux obtenus par d'autres méthodes multi-vues (CC et MVEC). Les résultats sont présentés sur la Figure G.15, où l'on peut voir que dans de nombreux cas, notre approche est meilleure que le meilleur réseau connu (LNet), et qu'elle permet souvent de surpasser le meilleur réseau (BNet). De plus, MVnet est également plus efficace que les autres méthodes d'agrégation testées.



(a) VOC2007 **(b)** COIL100

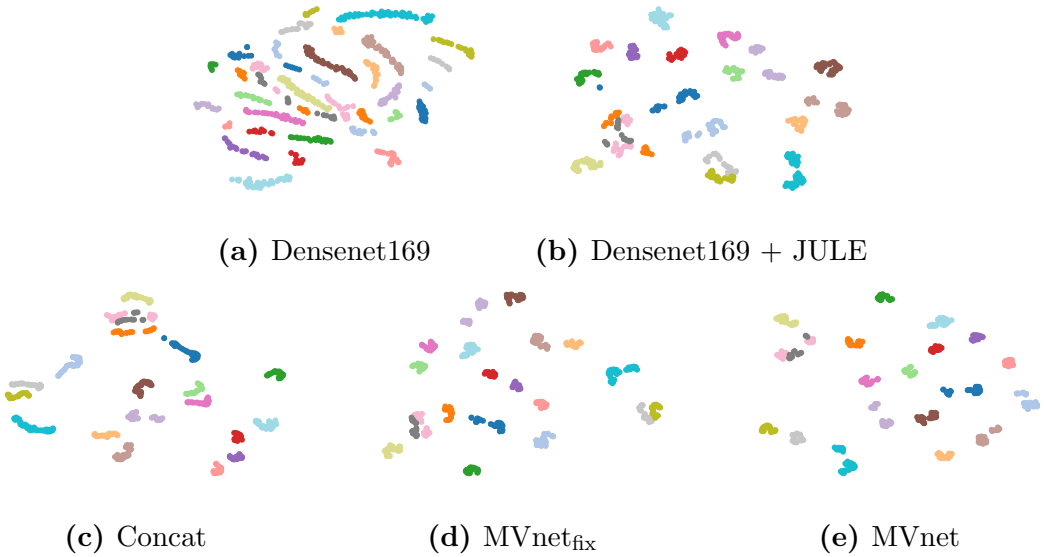
(c) Archi **(d)** MIT

(e) Flowers **(f)** Birds

(g) UMist **(h)** FEI

Figure G.15: Scores $MIX_{0.5}$ pour différentes méthodes de MVC et différents jeux de données.

L'autre intérêt majeur de l'utilisation de DMVC par rapport à MVEC est qu'il permet d'obtenir une nouvelle représentation unifiée des données de départ. La visualisation t-SNE des différentes caractéristiques durant le processus de partitionnement DMVC peut être vue sur la Figure G.16, où l'on remarque que les représentations extraites par DMVC sont de plus en plus compacts et séparent de mieux en mieux les différentes classes.



(a) Densenet169 **(b)** Densenet169 + JULE

(c) Concat **(d)** $\text{MVnet}_{\text{fix}}$ **(e)** MVnet

Figure G.16: Visualisation t-SNE 2d des caractéristiques extraites du jeu de données UMist à différentes étapes de la structure DMVC.

G.4.4 Conclusions

Dans ce chapitre, une approche en deux temps pour la résolution du problème de partitionnement d'images a été présentée. Plusieurs représentations des données initiales sont extraites par différents CNNs, puis un réseau de neurones est entraîné de manière non supervisé, en utilisant JULE, afin de résoudre le problème de partitionnement multi-vues. La méthode proposée présente l'avantage d'enlever la décision critique du choix de l'extracteur et permet d'obtenir d'excellents résultats pour différents exemples d'IC.

La classification d'images non-supervisée est une brique technologique essentielle pour adresser le problème d'URS. Cependant, pour se rapprocher de l'objectif de cette thèse, à savoir créer un module de tri autonome robotisé, il est également important d'étudier la manière de collecter les images qui vont être envoyées à l'algorithme d'IC. Cette thématique de prise de données avec un robot équipé d'une caméra est le sujet des deux prochains chapitres.

G.5 Tri robotisé non-supervisé à partir de poses caméra fixes.

G.5.1 Introduction

La formulation classique d'un problème de classification par apprentissage, qu'il soit supervisé ou non, cherche à construire un modèle pour associer des données numériques (images) à des prédictions (catégories). Cependant, pour une mise en oeuvre physique d'une application de classification, les entrées du système n'existent pas a priori et doivent d'abord être mesurées à partir d'objets physiques. Pour le problème d'URS étudié dans cette thèse, les objets à trier sont représentés par des images et les prédictions finales sont obtenues en résolvant un problème d'IC, qui a été adressé précédemment. Pour mettre en oeuvre une telle application sur un vrai robot, le processus d'acquisition des données doit également être défini et son choix est primordial pour obtenir une solution robuste. En effet, suivant la scène dans laquelle le robot évolue, les poses des objets et de la caméra, et les conditions de lumières, les images qui sont envoyées à l'algorithme d'IC peuvent être très différentes. Une représentation schématique du module de prise de décision d'URS est présentée sur la Figure G.17. Les paramètres en violet varient et ne peuvent pas être contrôlés alors que ceux en orange doivent être définis pour mettre en oeuvre l'application. L'objectif d'une application d'URS est donc de définir un algorithme d'IC ainsi que des règles pour la prise d'images qui réalisent ensemble une classification robuste et consistante dans le domaine de variation des paramètres non maîtrisés.

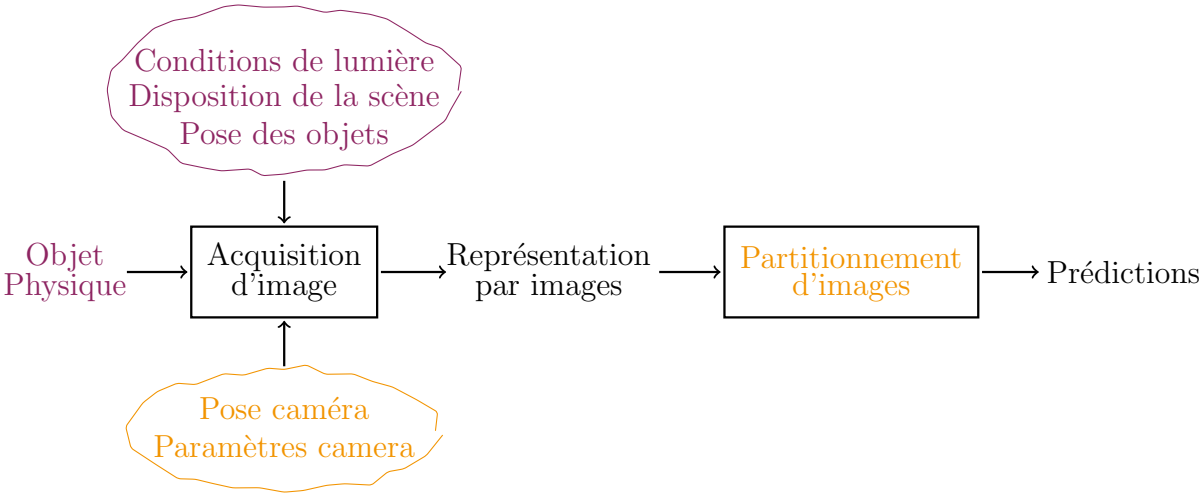
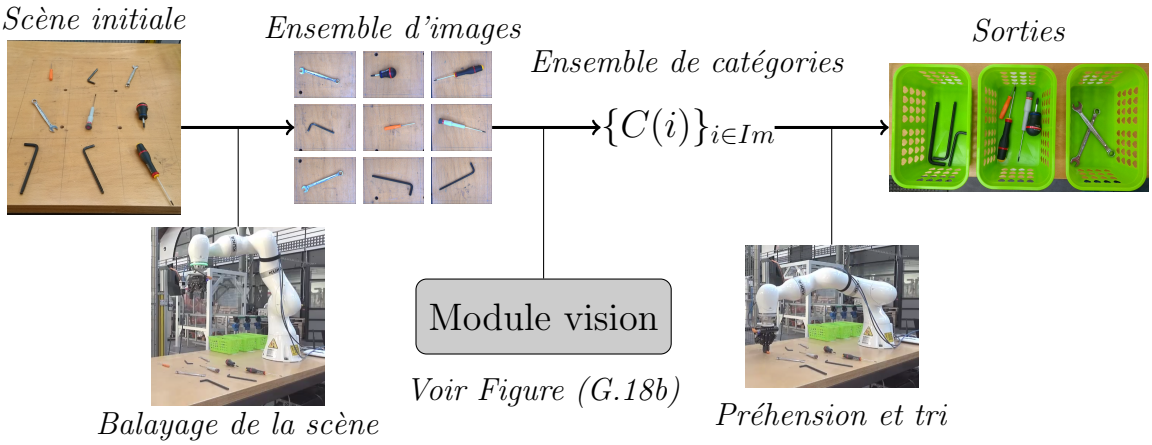


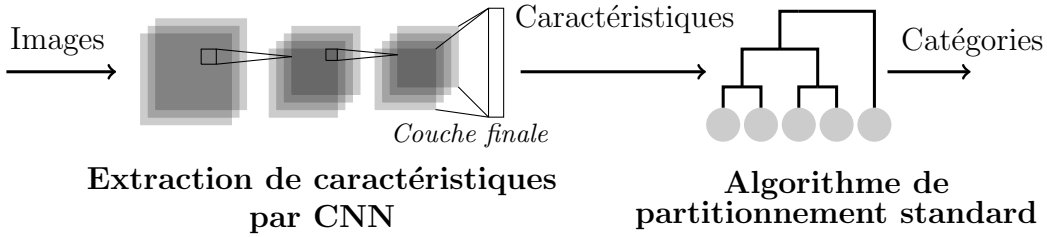
Figure G.17: Représentation schématique du module de prise de décision d'une application de tri robotique non-supervisée avec des objets réels.

G.5.2 Première mise en oeuvre et robustesse aux conditions d'éclairage et d'arrière-plan

Une première mise en oeuvre de tri robotique non supervisé est présentée sur la Figure G.18. La prise de données se fait avec une caméra montée sur l'effecteur d'un robot sériel, et pour toutes les prises de vue, la caméra est placée verticalement au dessus des différents objets. L'ensemble des images collectées est ensuite envoyé à un algorithme d'IC présenté précédemment.



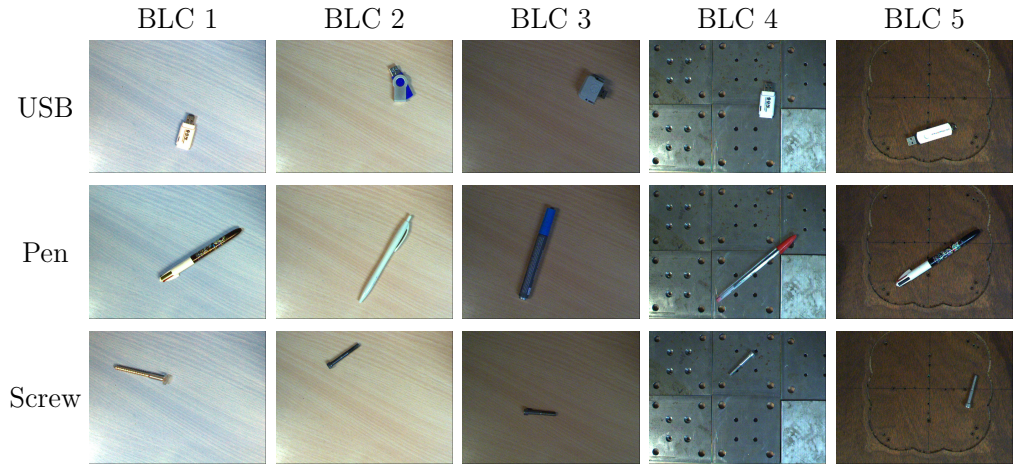
(a) Schéma de principe de l'application de tri robotique non-supervisé (URS).



(b) Description du module de vision.

Figure G.18: Tri robotique non-supervisé avec pose de caméra fixée.

Afin de tester la robustesse de la méthode de prise d'images par caméra verticale, un jeu d'images a été créé spécifiquement. Celui-ci comprend différents objets, placés sur différents supports et dans différentes conditions de lumière. Différentes instances de différentes classes sont utilisées afin de pouvoir réaliser un tri non-supervisé. Pour chaque objet, dans chaque condition, différentes positions de l'objet sont collectées afin de pouvoir également tester la robustesse à la pose de l'objet par la suite. Des exemples d'images extraites du jeu de données créé peuvent être vues sur la Figure G.19. Des modifications artificielles de la luminosité sont également proposées afin de tester la méthodologie sous des conditions extrêmes.



(a) Différentes poses, arrière-plans and conditions de lumière.

(b) Modification artificielle de la luminosité sur BLC 2.

Figure G.19: Exemples d'images utilisées pour l'évaluation de la robustesse aux perturbations non-maîtrisées.

Les différents résultats obtenus sont présentés sur les Tables G.1 et G.2. On y voit notamment que les résultats sont assez sensibles aux changements de luminosités importants ainsi qu'aux arrière-plans présentant des motifs.

Tableau G.1: Résultats de partitionnement pour différentes architectures de CNN et différents algorithmes de partitionnement sur le jeu de données de partitionnement d'outils.

		BLC1		BLC2		BLC3		BLC4		BLC5	
		NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*
Inception V3	Agg	0.82	*0.81*	0.82	*0.81*	0.80	*0.80*	0.65	*0.65*	0.79	*0.76*
	KMeans	0.80	*0.79*	0.79	*0.78*	0.76	*0.76*	0.63	*0.64*	0.75	*0.73*
Resnet50	Agg	0.81	*0.81*	0.74	*0.74*	0.74	*0.75*	0.62	*0.59*	0.72	*0.71*
	KMeans	0.77	*0.78*	0.71	*0.72*	0.71	*0.72*	0.58	*0.58*	0.70	*0.70*
VGG16	Agg	0.76	*0.75*	0.74	*0.73*	0.73	*0.72*	0.61	*0.60*	0.70	*0.69*
	KMeans	0.72	*0.72*	0.70	*0.70*	0.71	*0.70*	0.58	*0.57*	0.67	*0.67*
VGG19	Agg	0.76	*0.76*	0.77	*0.76*	0.71	*0.72*	0.59	*0.58*	0.71	*0.70*
	KMeans	0.73	*0.73*	0.73	*0.73*	0.69	*0.70*	0.56	*0.56*	0.67	*0.67*
Xception	Agg	0.86	*0.85*	0.90	*0.90*	0.84	*0.85*	0.69	*0.69*	0.83	*0.81*
	KMeans	0.84	*0.83*	0.87	*0.86*	0.82	*0.82*	0.66	*0.66*	0.80	*0.80*

Tableau G.2: Résultats de partitionnement pour différentes architectures de CNN et différents algorithmes de partitionnement sur l'ensemble d'images BLC2 avec différentes modifications artificielles de la luminosité.

		Very dark		Dark		Normal		Bright		Very bright	
		NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*	NMI	*PUR*
Inception V3	Agg	0.74	*0.72*	0.81	*0.79*	0.82	*0.81*	0.80	*0.80*	0.71	*0.70*
	KMeans	0.70	*0.70*	0.77	*0.75*	0.79	*0.78*	0.77	*0.77*	0.66	*0.67*
Resnet50	Agg	0.67	*0.67*	0.73	*0.73*	0.74	*0.74*	0.69	*0.68*	0.61	*0.61*
	KMeans	0.65	*0.66*	0.70	*0.71*	0.71	*0.72*	0.66	*0.66*	0.58	*0.59*
VGG16	Agg	0.66	*0.66*	0.73	*0.72*	0.74	*0.73*	0.68	*0.68*	0.61	*0.61*
	KMeans	0.62	*0.63*	0.69	*0.69*	0.70	*0.70*	0.65	*0.66*	0.57	*0.58*
VGG19	Agg	0.67	*0.67*	0.76	*0.75*	0.77	*0.76*	0.74	*0.72*	0.64	*0.65*
	KMeans	0.64	*0.65*	0.73	*0.73*	0.73	*0.73*	0.71	*0.70*	0.59	*0.62*
Xception	Agg	0.77	*0.77*	0.88	*0.89*	0.90	*0.90*	0.84	*0.84*	0.73	*0.74*
	KMeans	0.74	*0.74*	0.85	*0.86*	0.87	*0.86*	0.82	*0.82*	0.70	*0.71*

G.5.3 Influence de l'angle de prise de données

Pour essayer de pallier à ces problèmes et d'augmenter la robustesse du système global, une étude de l'influence de la prise de vue est proposée. Pour ce faire, nous comparons les résultats obtenus précédemment avec une méthodologie multi-vues, utilisant les images obtenues pour différentes positions ensemble, afin de procéder au tri. Le schéma de principe de la méthodologie employée peut être trouvé sur la Figure G.20.

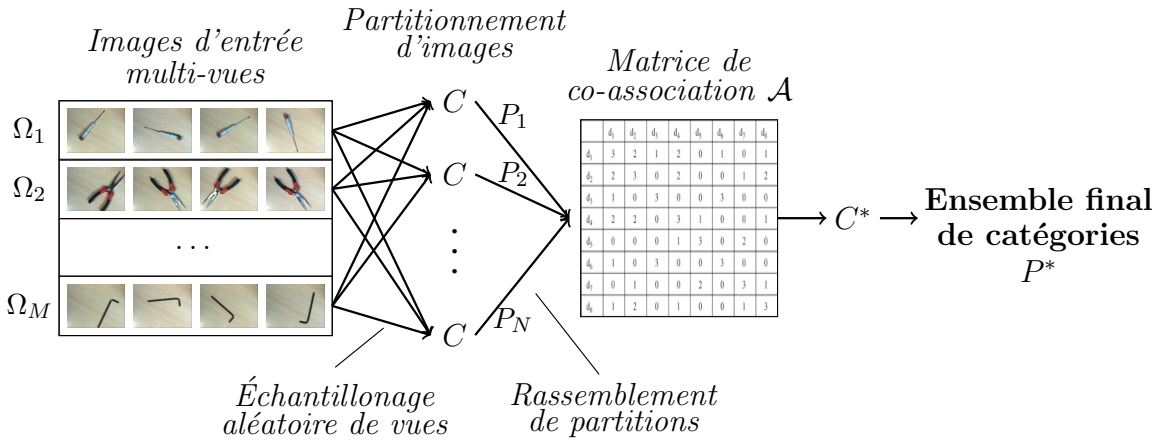


Figure G.20: Méthode MVEC, utilisée pour combiner plusieurs vues de chaque objet.

Les résultats obtenus par cette approche, présentés sur la Table G.3, montrent que l'utilisation simultanée de différents points de vue des objets permet d'améliorer la robustesse aux différents éléments non maîtrisables. L'étude menée dans ce chapitre souligne donc l'importance d'étudier la méthodologie de collecte des images de

manière plus fine que l'approche initiale proposée.

Tableau G.3: Résultats du partitionnement MVEC pour différentes BLC. Pour comparer, les résultats avec vue unique sont rappelés entre parenthèse.

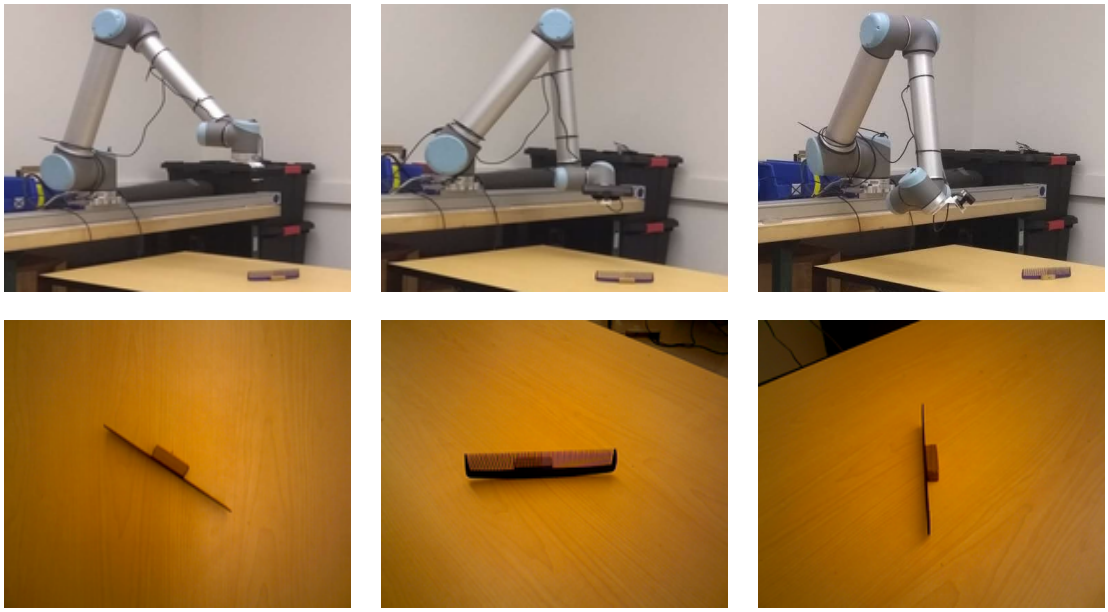
		NMI	Purity
	BLC1	0.95 *(0.86)*	0.96 *(0.85)*
BLC2	Very dark	0.91 *(0.77)*	0.93 *(0.77)*
	Dark	1.00 *(0.88)*	1.00 *(0.89)*
	Normal	1.00 *(0.90)*	1.00 *(0.90)*
	Bright	0.96 *(0.84)*	0.96 *(0.84)*
	Very bright	0.84 *(0.73)*	0.86 *(0.74)*
	BLC3	0.95 *(0.84)*	0.96 *(0.85)*
	BLC4	0.84 *(0.69)*	0.82 *(0.69)*
	BLC5	0.95 *(0.83)*	0.96 *(0.81)*

G.5.4 Conclusions

Dans ce chapitre, une première mise en oeuvre d'URS est proposée, dans laquelle des images sont collectées avec une caméra placée verticalement au-dessus des différents objets à trier. Cette approche fonctionne plutôt bien pour le cas pratique industriel testé. Pour tester de manière plus complète la robustesse de cette approche aux changements d'arrière-plan et de luminosité, un jeu d'images représentant différents objets dans différentes conditions a été créé. Ces données sont difficiles pour l'IC. Les résultats obtenus sur ces données montrent que la robustesse de l'approche diminue quand la luminosité change trop et quand l'arrière-plan comporte des motifs. Des expériences multi-vues sont également menées avec différentes images de chaque objet et montrent que l'angle de vue sous lequel sont observés les objets influe sur la robustesse du système global. Ainsi, comme il semble impossible de définir une méthodologie robuste de prise de données par caméra fixe, nous proposons de développer un schéma de pose caméra adaptative et autonome dans le prochain chapitre.

G.6 Sélection de vues sémantiquement riches

Comme démontré dans le chapitre précédent, une mise en oeuvre robuste d'URS requiert non seulement un bon algorithme d'IC mais également une bonne stratégie d'acquisition d'images. Dans ce chapitre, afin de pouvoir choisir des images adaptées pour le partitionnement, une méthode de sélection de vues à fort contenu sémantique est proposée. La sélection de vues sémantique est une direction de recherche peu explorée jusqu'à maintenant, mais qui peut avoir de nombreux impacts positifs pour la robotique. L'utilisation de la mobilité d'un robot pour améliorer la compréhension des objets qu'il manipule semble une piste d'étude importante pour de nombreuses applications. Pour mieux comprendre le rôle de la prise de vue dans la classification d'objets, on peut observer la Figure G.22, où seule l'image du milieu permet au robot de comprendre que l'objet étudié est un peigne. Dans ce chapitre, nous proposons d'entraîner un réseau de convolution pour optimiser la pose d'un robot sériel, équipé d'une caméra en bout de bras, afin de maximiser le contenu sémantique de l'image obtenue et de comprendre la nature des objets observés. Les résultats expérimentaux obtenus démontrent que les poses retenues par le réseau surpassent les poses fixes pour la tâche d'URS.



(a) Vue de dessus : $\theta = 90°$; $\varphi = 90°$

(b) Bonne vue : $\theta = 45°$; $\varphi = 45°$

(c) Mauvaise vue : $\theta = 135°$; $\varphi = 45°$

Figure G.21: Illustration du problème de sélection de vue sémantique. La paramètrisation angulaire est définie plus tard.

G.6.1 Méthodologie

L'objectif de ce chapitre est donc de proposer un modèle permettant de prédire si une pose caméra sera intéressante pour pouvoir comprendre la nature d'un objet. Cette prédiction doit être faite seulement à partir d'une vue de dessus perpendiculaire à l'objet. Pour réaliser cet objectif, un jeu de données multi-vues de différents objets est constitué. Après avoir rassemblé 144 objets, appartenant à 29 catégories différentes, un robot est utilisé afin de collecter des images de chacun des objets, prises depuis différentes poses de la caméra. Trois positionnements de chacun des objets sont choisis aléatoirement et pour chaque pose objet, une vingtaine d'images est collectée sous différents angles. En tout, 9112 images sont obtenues, comme le montre le Tableau G.4, qui résume les statistiques du jeu d'images. Des exemples d'images pour un objet dans une pose donnée peuvent être trouvés sur la Figure G.22.

Tableau G.4: Statistiques du jeu de données d'images multi-objets/multi-poses/multi-vues proposé.

# Classes	# Object/class (*total*)	# Poses/object (*total*)	# Images/pose (*total*)
29	4-6 (*144*)	3 (*432*)	17-22 (*9112*)

Ensuite, les images sont séparées en un jeu d'entraînement et un jeu de validation, cette séparation est faite par catégorie. Un indice de "partitionnabilité" est ensuite calculé pour toutes les images du jeu d'entraînement puis un CNN est entraîné à prédire cet indice à partir des informations de vue de dessus et du positionnement attendu de la caméra. Des images à haut (respectivement faible) score de partitionnabilité sont encadrées en vert (respectivement rouge) sur la Figure G.22. L'architecture de réseau retenue pour cet entraînement est présentée sur la Figure G.23. Des expli-

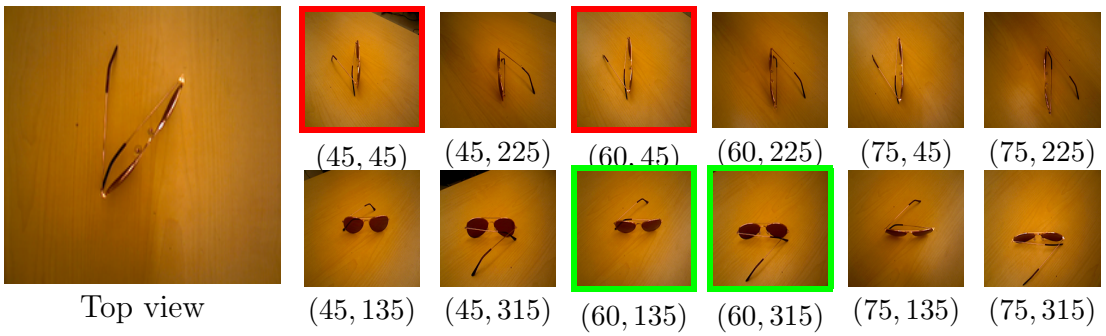


Figure G.22: *Meilleure visualisation en couleur.* Ensemble de vues pour un objet de la catégorie "lunettes de soleil" dans une pose donnée. Les deux images avec le plus haut (resp. bas) score FM individuel sont encadrées en vert (resp. rouge).

cations détaillées sur la méthodologie de création du jeu de données multi-vues ainsi que sur l'entraînement du modèle peuvent être trouvées dans le corps du texte.

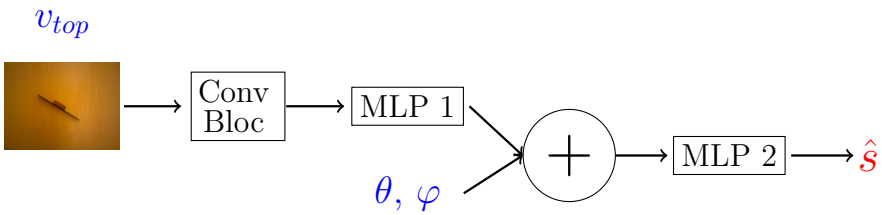


Figure G.23: Architecture SV-net proposée. *Les entrées sont en bleu et les sorties en rouge. Le "Conv Bloc" est la partie convolutionnelle de VGG19 (jusqu'à la couche "block4_pool").* Les MLPs sont présentés dans leurs versions intégrales dans le corps de la thèse.

G.6.2 Résultats

Pour analyser la qualité du réseau entraîné, nous comparons les vues qu'il prédit pour les objets du jeu de validation avec les vues de dessus fixes (TOP) et des vues aléatoires (RAND). Cette approche nous permet de valider que les prédictions du réseau sont meilleures que des vues fixes et aussi que la vue fixe choisie n'est pas particulièrement mauvaise à cette tâche. Les résultats sont comparés pour différents extracteurs de caractéristiques ainsi que différents algorithmes de partitionnement pour vérifier que le réseau n'a pas été sur-entraîné pour un modèle donné. Les résultats obtenus peuvent être vus sur le TableauG.5, où l'on voit que notre réseau est capable de proposer des vues intéressantes pour le problème de reconnaissance sémantique d'objets. Certaines des prédictions du réseau apparaissent sur la Figure G.24 et semblent correspondre à des angles de vues que des humains auraient pu choisir.

Tableau G.5: SV-net validation. Comparaison des résultats de partitionnement entre différentes méthodes de sélection de vue sur les données de test. *Pour chaque couple (c, m), le meilleur résultat est en gras.*

		FM	NMI	PUR
XCE_AGG	TOP	0.44	0.51	0.70
	RAND	0.48	0.56	0.74
	SV-net	**0.55**	**0.63**	**0.78**
XCE_KM	TOP	0.44	0.51	0.70
	RAND	0.48	0.55	0.73
	SV-net	**0.55**	**0.62**	**0.78**
VGG_AGG	TOP	0.46	0.53	0.71
	RAND	0.44	0.51	0.70
	SV-net	**0.48**	**0.55**	**0.73**

(a) Exemples de vues de dessus

(b) Sélections SV-net correspondantes

Figure G.24: Exemples de vues prédites par SV-net.

G.6.3 Conclusion

Dans ce chapitre, nous avons introduit un nouveau problème appelé sélection de vue sémantique. La SVS sert à choisir une bonne pose caméra pour améliorer la connaissance sémantique d'un objet physique observé. Pour résoudre ce problème, un jeu de données spécifique a été créé et une approche d'apprentissage profond a été proposée. La solution présentée a montré qu'il est possible d'inférer la qualité sémantique d'une pose caméra seulement à partir d'informations provenant d'une autre vue de l'objet. Ce résultat ouvre de nombreuses perspectives d'applications, telles que le tri robotique autonome, qui est en général résolu par des vues de dessus.

Jusqu'alors, les recherches présentées dans cette thèse se sont concentrées sur le module de prise de décision de l'URS. Les méthodologies proposées, tant pour la sélection de vues que pour l'IC, ont permis d'améliorer les résultats de tri d'objets non-supervisés. Cependant, un schéma d'URS complet doit proposer des solutions à d'autres problèmes importants tels que la segmentation de scènes, la prise d'objets, la localisation d'objets, etc. Ainsi, la prochaine partie de cette thèse se concentre sur deux de ces compétences : l'apprentissage de trajectoires et la localisation d'objets. Ces avancées représentent des étapes importantes vers la conception d'un robot de tri complètement autonome.

G.7 Apprentissage de trajectoires indépendant du modèle du système

G.7.1 Introduction

Dans les chapitres précédents, le problème de compréhension non supervisé d'objets physiques, qui est une compétence essentielle pour l'URS, a été étudié. Cependant, pour pouvoir mettre en oeuvre une application d'URS industrielle autonome et robuste, de nombreuses autres compétences sont requises. Dans les deux prochains chapitres, deux de ces compétences sont étudiées : l'apprentissage de trajectoires et la localisation 3D. Ainsi, dans ce chapitre, nous proposons d'adapter une méthode récente d'apprentissage de trajectoires afin qu'elle puisse fonctionner pour n'importe quelle fonction de coût mesurable. Cette solution est indépendante du modèle du robot, ce qui signifie que le schéma d'apprentissage ne nécessite pas de connaître l'intégralité des paramètre physiques du système étudié. Cette méthode d'apprentissage présente l'intérêt d'être plus intuitive et donc de rendre la programmation robotique plus accessible aux opérateurs dans l'industrie.

Pour valider notre solution, la tâche de positionnement cartésien d'un robot sériel est étudiée. Un robot est entraîné à apprendre comment atteindre un point de l'espace cartésien avec son effecteur en utilisant uniquement des information de capteurs de position. Le modèle est d'abord validé sur une tâche classique, où l'effecteur est attaché physiquement au robot, puis sur une tâche plus complexe, où le modèle direct ne peut pas être calculé.

G.7.2 Méthodologie

La solution proposée est construite à partir d'une méthode basée sur l'iLQG. L'algorithme iLQG a pour objectif d'optimiser des trajectoires, représentées numériquement par une alternance d'états du système étudié et de commandes envoyées aux actionneurs afin de changer cet état. Un schéma représentant les différents éléments d'une trajectoire est présenté sur la Figure G.25.

Sur la représentation de la Figure G.25, lorsque la dynamique est linéaire et la fonction de coût est quadratique, une série de commandes optimales peut être trouvée analytiquement grâce à la théorie des LQG. Cependant, lorsque ces fonctions sont non-linéaires et plus complexes, il est nécessaire de pouvoir en faire des développements limités locaux pour pouvoir optimiser des trajectoires. La littérature a aujourd'hui montré qu'il est possible d'adopter un schéma d'exploration et régression pour dériver

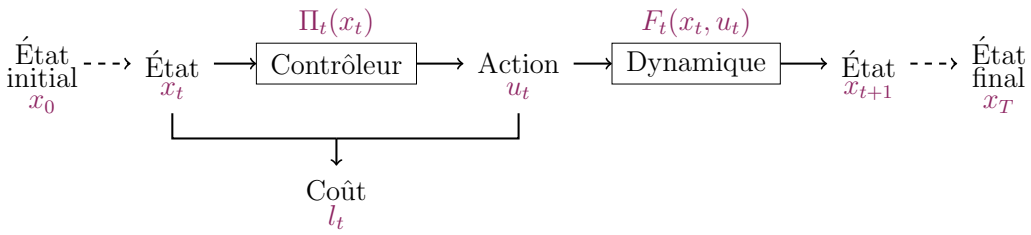
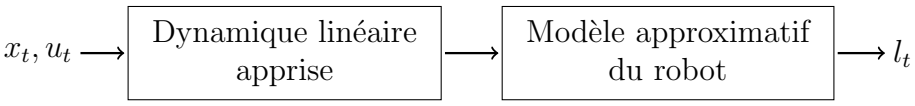
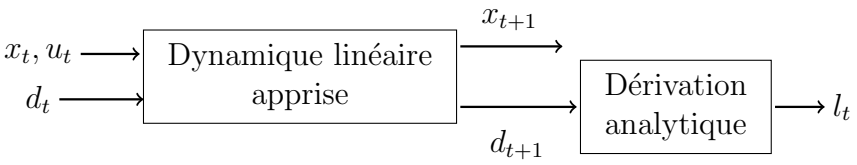


Figure G.25: Schéma bloc pour définir une trajectoire et résumer les notations.

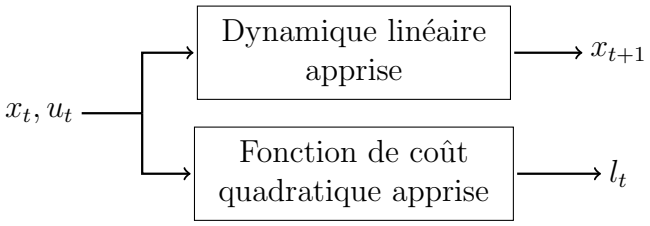
une approximation linéaire locale de la dynamique et obtenir de bons résultats d'optimisation de trajectoire. Cependant, pour ce qui est de la fonction de coût, la plupart des articles propose de la calculer à partir d'un modèle du robot ou encore d'ajouter des variables au vecteur d'état afin de pouvoir ensuite la calculer analytiquement. Dans ce chapitre, nous proposons d'étendre le processus d'exploration-régression à l'estimation de la fonction de coût. Cette méthode permet d'éviter de créer un estimateur quadratique calculé à partir d'une approximation linéaire. Les différentes méthodes possibles pour calculer le développement limité de deuxième ordre de la fonction de coût sont présentées sur la Figure G.26.



(a) Utiliser un modèle du robot.



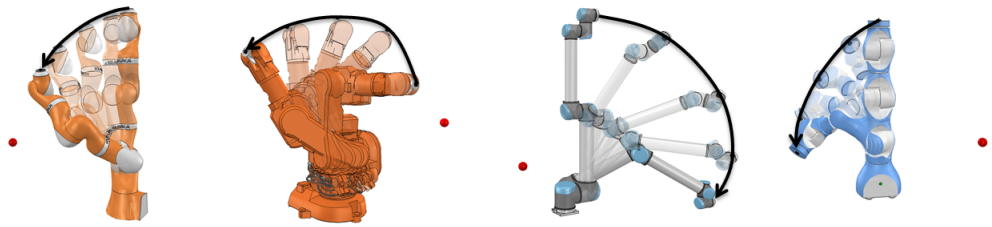
(b) Inclure la distance d_t dans la représentation d'état.



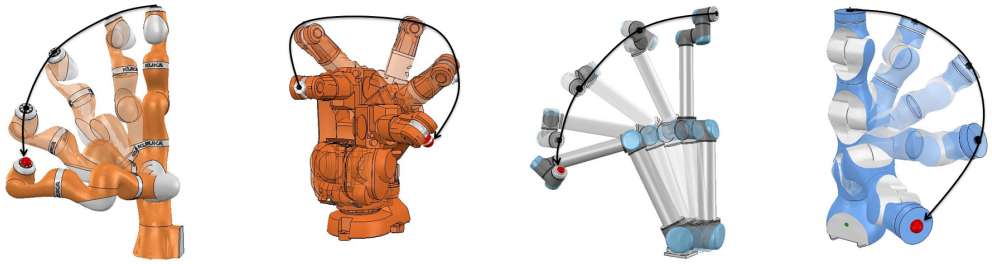
(c) Apprendre l'approximation quadratique du coût.

Figure G.26: Différentes options pour calculer le développement limité de second ordre de la fonction de coût cartésienne à partir de vecteurs d'état et d'un contrôle dans l'espace angulaire.

La méthodologie utilisée est détaillée dans le corps de la thèse. Nous noterons simplement que celle-ci requiert de prédéfinir plusieurs hyperparamètres et que, pour la tâche de positionnement cartésien étudiée, ceux-ci ont été choisis en simulation (Figure G.27) avant de mettre en oeuvre l'apprentissage sur système réel.



(a) Trajectoire nominale initiale, choisie aléatoirement



(b) Trajectoires finales apprises

Figure G.27: Trajectoires apprises en simulation pour une tâche de positionnement cartésien pour différents robots industriels (KUKA LBR iiwa / ABB IRB 140 / UR 10 / F&P P-ARM, de gauche à droite).

G.7.3 Résultats

Nous proposons de valider la méthode en étudiant la tâche de positionnement cartésien de l'effecteur d'un robot sériel, tout d'abord pour une tâche classique, où l'effecteur est lié physiquement au robot, puis dans une tâche plus complexe, qui ne peut fonctionner sans notre méthode du fait de l'impossibilité de calculer un modèle direct du robot. La première expérience est comparée avec la méthode (c) sur la Figure G.26. Les résultats sont présentés sur la Figure G.28, on y voit notamment que notre méthode est plus consistante quand le nombre d'échantillons est suffisant, et qu'elle converge plus rapidement vers des comportements de haute précision. L'autre expérience, qui représente une validation qualitative de l'indépendance du modèle du robot, voit le robot équipé d'un pointeur laser avec lequel il doit trouver comment viser le centre d'une cible en actionnant ces moteurs angulaires (Figure G.29). Avec notre méthode, la tâche peut être résolue en l'espace de quelques minutes.

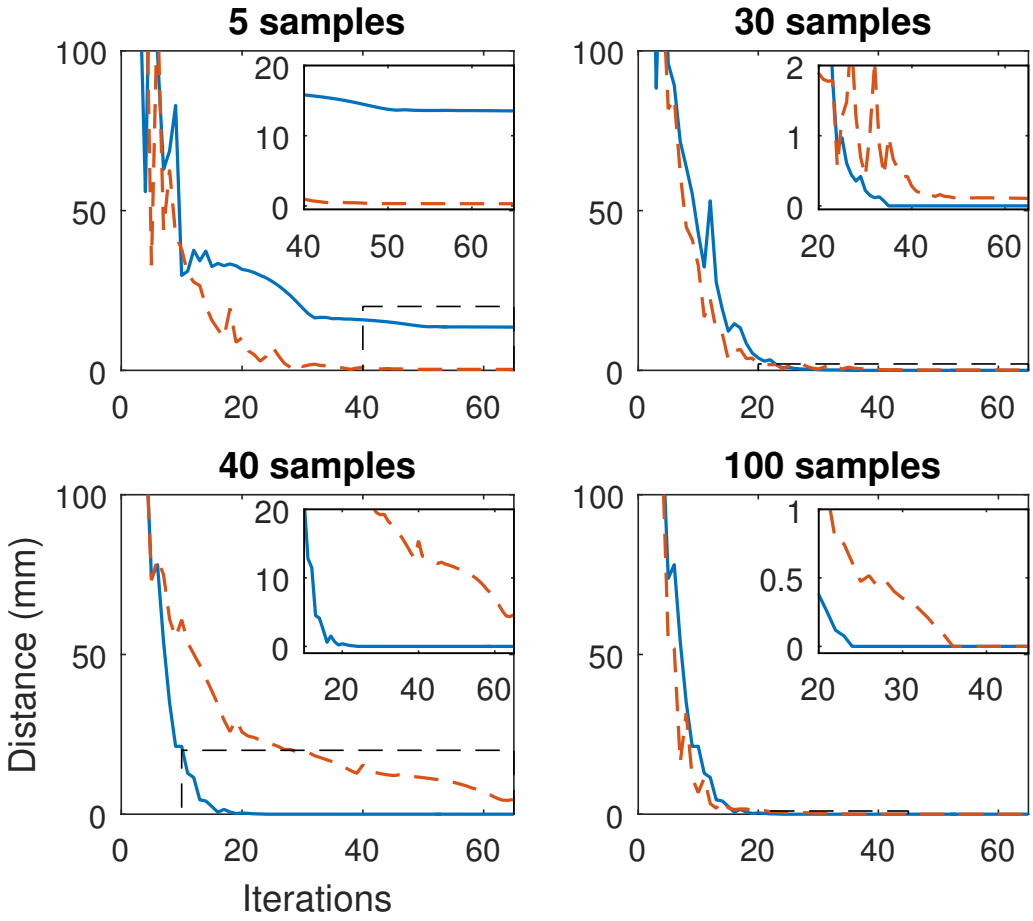
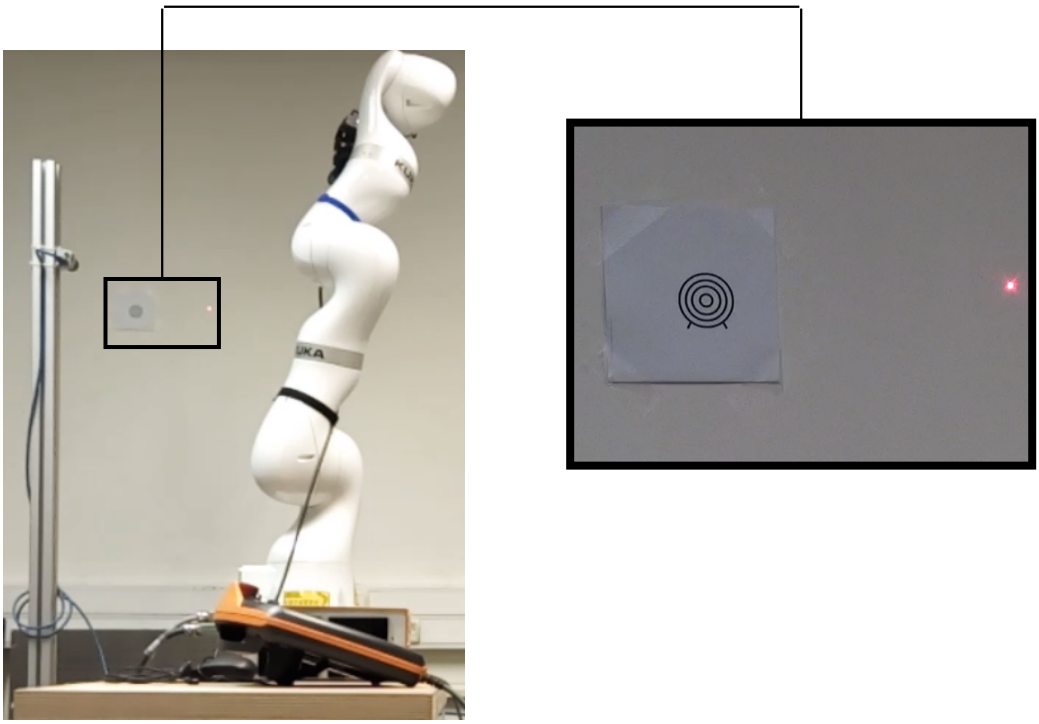


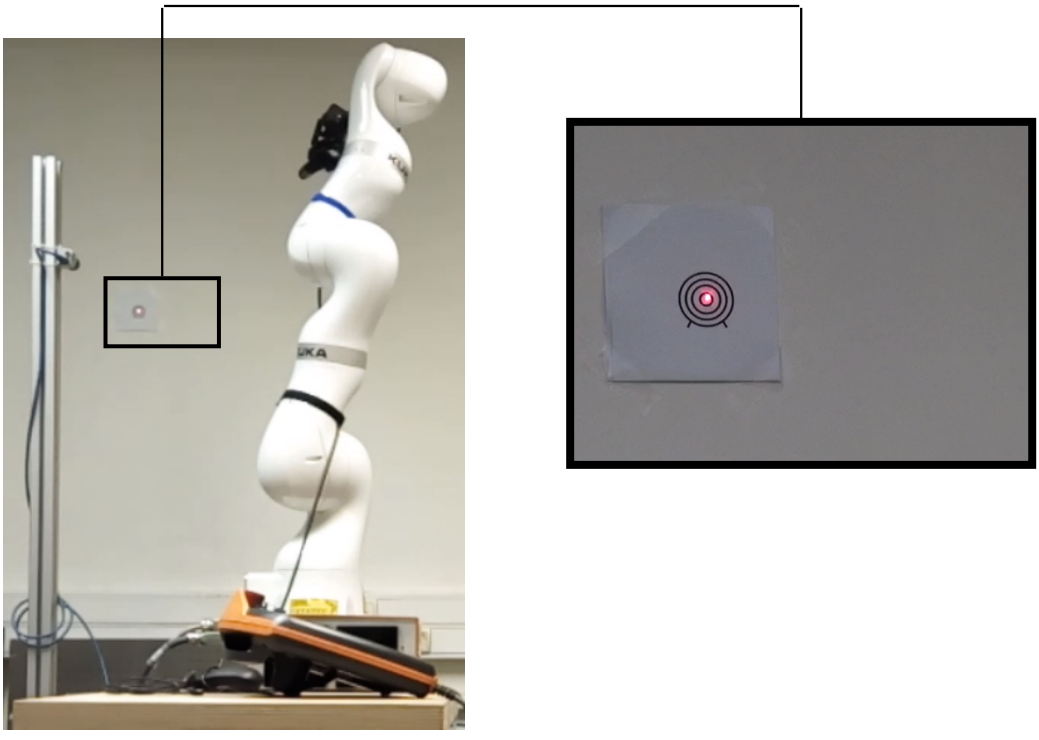
Figure G.28: Comparaison de deux méthodes : Régression quadratique (courbe pleine bleue) et modification du vecteur d'état (courbe pointillée orange). Les cadres en haut à droite de chaque figure sont des zooms sur les dernières itérations d'iLQG.

G.7.4 Conclusion

Ce chapitre introduit une version modifiée d'iLQG pour résoudre le problème de contrôle optimal local. Pour ce faire, le développement limité de second ordre de la fonction de coût est appris à partir de données mesurées. Comparée à d'autres méthodes de calcul de fonction de coût, notre approche présente l'avantage d'être indépendante du modèle de robot utilisé tout en ne nécessitant aucune cascade d'approximations. Cette approche est validée expérimentalement sur la tâche de positionnement cartésien d'un robot sériel sans utiliser de modèle du robot. Le système a seulement accès aux valeurs angulaires et à certains retours capteurs cartésiens. Les résultats obtenus sont plus stables et convergent plus rapidement vers des comportements très précis.



(a) Configuration initiale



(b) Pose finale apprise

Figure G.29: Validation qualitative : viser une cible avec un pointeur laser.

G.8 Construction automatique de jeux de données d'images réelles pour la localisation 3D d'objets en utilisant deux caméras

G.8.1 Introduction

Tout comme la compréhension d'objets physiques et l'apprentissage de trajectoires, la localisation 3D est une brique technologique importante pour rendre la robotique plus autonome. Dans ce chapitre, une méthode de création automatique de jeux de données stéréo pour la localisation d'objets est proposée. La localisation stéréo est une tâche complexe qui nécessite de nombreuses briques technologiques, présentées sur la Figure G.30, pouvant toutes être sources d'imprécision. L'entraînement d'un réseau de neurones résolvant la tâche bout-à-bout, sans passer par des briques intermédiaires, pourraient avoir un impact très bénéfique sur la qualité de la localisation. Un tel réseau correspond au cadre violet sur la Figure G.30. Ainsi, nous présentons une méthodologie pour générer automatiquement des données entrée-sortie pour entraîner un tel modèle. Nous proposons d'utiliser la grande précision et répétabilité d'un robot industriel pour construire ce type de jeu de données. Ce chapitre est un sujet de recherche en développement et doit encore être validé en utilisant les jeux de données créés pour entraîner des réseaux de localisation. Cependant, la méthodologie de construction proposée peut être utile à la communauté.

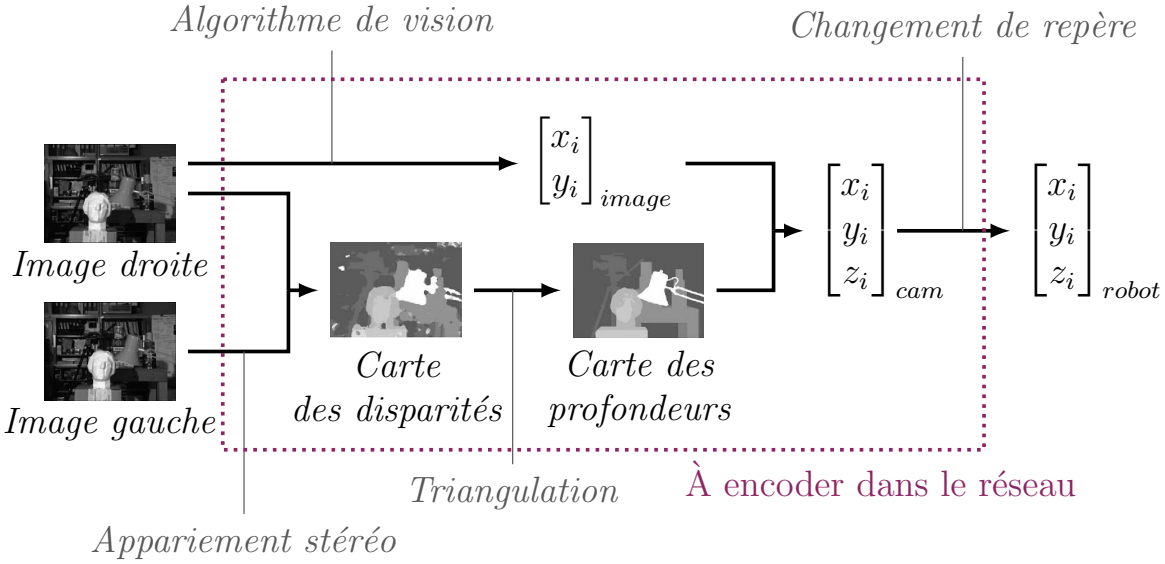


Figure G.30: Approche classique pour la localisation par vision stéréo.

G.8.2 Méthodologie

Pour générer un tel jeu de données, nous proposons de fixer l'objet à localiser sur l'effecteur d'un robot sériel dont on connaît le modèle direct. Ainsi, il est facile de calculer la position précise de l'objet dans le repère du robot. Deux caméras sont fixées par rapport au repère robot, de telle manière qu'elles observent la même zone dans l'espace de travail du robot. Ensuite, le robot déplace l'objet dans le champ des caméras et nous pouvons ainsi générer des couples d'images associés à une position 3D connue de l'objet. Ces données sont les entrées-sorties du réseau que l'on souhaite entraîner et ce procédé permet donc de générer un jeu de données supervisé pour la tâche de localisation stéréo. Des exemples d'images générées pour la tâche de localisation de tournevis peuvent être vues sur la Figure G.31.

Figure G.31: Ensemble d'images représentatif du jeu de données de localisation de tournevis.

Afin d'éviter d'obtenir des biais importants dans nos données, nous essayons de changer la luminosité et les arrières-plans des images lors de la création des jeux de données, comme nous pouvons le voir sur la Figure G.31. Afin d'éviter que nos réseaux apprennent simplement à localiser le robot puis appliquer une translation, il est également important que ce dernier ne soit pas présent dans l'intégralité des images. Pour ce faire, le robot est caché physiquement sur certaines des images (Figure G.31). Un algorithme pour supprimer le robot des images est également proposé, il se base sur l'acquisition de quatre images et est expliqué en détail dans le corps du texte. Un schéma explicatif peut être vu sur la Figure G.32.

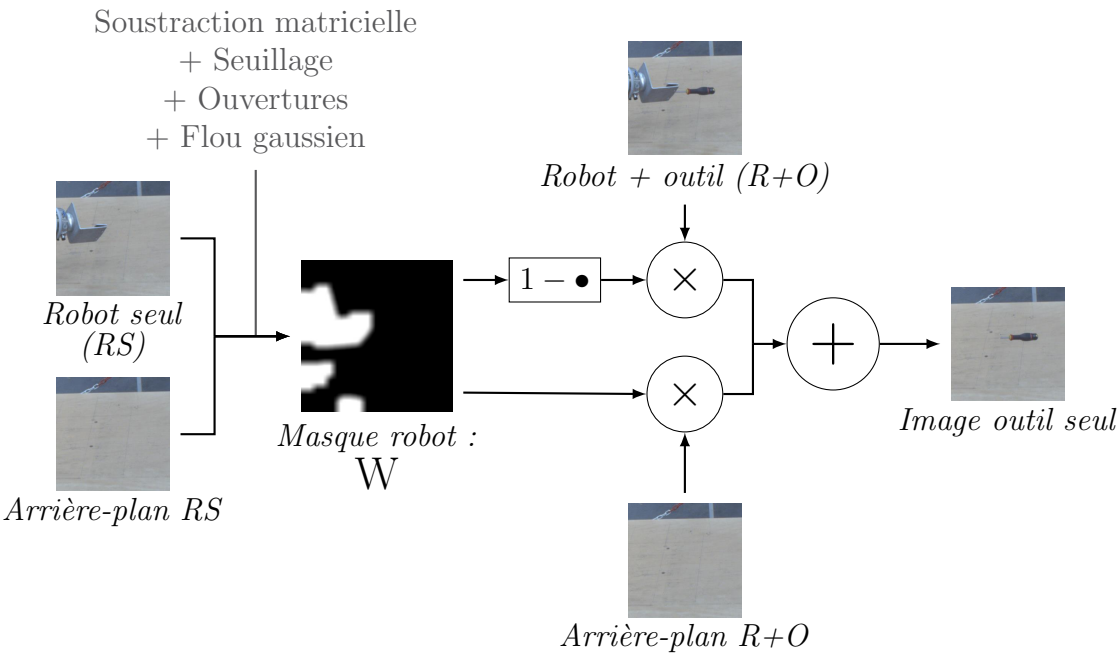


Figure G.32: Approche de vision par ordinateur utilisant quatre images pour générer une image contenant seulement l'objet étudié.

G.8.3 Conclusions

Dans ce chapitre, nous avons proposé une méthodologie de construction de jeux de données supervisés pour la localisation stéréo. Un jeu de données a été créé pour des tournevis et mis à la disposition de la communauté. Ces jeux de données peuvent ensuite servir à entraîner des CNNs siamois à résoudre la tâche de localisation.

G.9 Conclusions générales

Les robots industriels modernes sont des machines qui peuvent être reprogrammées pour résoudre de nombreuses tâches sans toucher à leur architecture physique. Ces caractéristiques leur permettent de résoudre une grande diversité de tâches. Cependant, leur utilisation actuelle est encore loin du niveau d'autonomie et d'adaptabilité espéré pour de tels systèmes complexes. Dans cette thèse, nous avons proposé diverses contributions d'apprentissage automatique afin de se rapprocher de l'autonomie complète. Ces contributions sont principalement centrées autour du tri robotique non-supervisé mais trouvent de nombreux autres domaines d'application dans la robotique.

Pour en savoir plus sur le travail réalisé ainsi que sur les perspectives futures de développement, nous invitons le lecteur à se référer au corps du texte de cette thèse, en anglais.

METHODES D'APPRENTISSAGE AUTOMATIQUE POUR DES APPLICATIONS ROBOTIQUES DANS UN CONTEXTE INDUSTRIEL :
ETUDE DE CAS DU TRI ROBOTISÉ

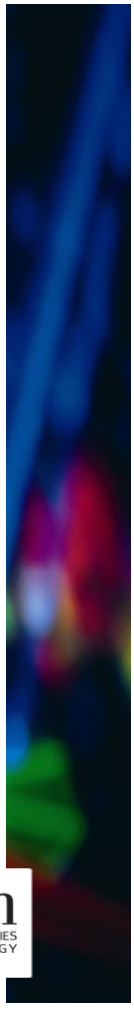
RESUME: L'architecture multi-axes des robots industriels permet de les programmer pour effectuer des tâches diverses. Cependant, malgré qu'ils soient équipés de nombreux capteurs - ce qui devrait leur permettre de s'adapter à des changements d'environnement - l'utilisation de robots dans l'industrie se limite souvent à des tâches très répétables et ne nécessitant que peu d'adaptabilité. Dans un contexte industriel, la programmation de robots capables de s'adapter automatiquement à diverses applications, et étant robustes sous différentes conditions de fonctionnement est une source de progrès importante. Ainsi, dans cette thèse, plusieurs contributions en apprentissage automatique sont proposées dans le but de concevoir des robots intelligents, ayant une plus grande gamme de fonctionnements. Les méthodes présentées dans ce mémoire sont centrées autour du tri autonome d'objets mais peuvent servir à implémenter de nombreuses autres applications robotiques. Afin de concevoir des applications plus polyvalentes, des solutions aux problèmes de tri d'images non supervisé, de choix de vue optimal, d'apprentissage de trajectoires et de localisation stéréoscopique ont été développées.

Mots clés : Tri robotique, Classification non supervisée, Transfert de connaissance, Choix de vue optimal, Apprentissage de trajectoires, Création automatique de bases de données

MACHINE LEARNING IMPROVEMENTS FOR ROBOTIC APPLICATIONS IN AN INDUSTRIAL CONTEXT:
CASE STUDY OF AUTONOMOUS SORTING

ABSTRACT: Thanks to their flexible mechanical design, modern industrial robots can be programmed for different tasks. However, despite the fact that they are highly instrumented – which should enable them to be responsive to their environment - the use of robots in industry is still often restricted to repeatable tasks with low level of adaptability. In an industrial context, it is essential to program robots that can autonomously adapt to different applications and are robust to changes in their working conditions. Hence, in this thesis, several machine learning contributions are presented, aiming at designing smarter robotic applications, with a broader operational range. The methods developed are centered on autonomous sorting, but may be useful to address problems in many other subfields of robotics. Throughout this thesis, new approaches are proposed to address image clustering, optimal view selection, trajectory learning and stereo localization, with the objective of designing more versatile robotic applications.

Keywords : Robotic sorting, Image clustering, Transfer learning, Optimal view selection, Trajectory learning, Autonomous dataset generation



ARTS ET MÉTIERS ParisTech

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY