UNIVERSITY OF ALICANTE

PHD THESIS

Deep Learning for 3D Perception: Computer Vision and Tactile Sensing

Author

Alberto GARCIA-GARCIA

Advisors

Jose GARCIA-RODRIGUEZ

Sergio ORTS-ESCOLANO

A thesis submitted in fulfilment of the requirements

for the degree of Doctor of Philosophy

in the

3D Perception Lab

Department of Computer Technology

May 20, 2019

This document was proudly typeset with LaTeXand Ti*k*Z.

Contact Details

Alberto García García

alberto.garcia.ua@gmail.com

www.albertogarciagarcia.com

"Will robots inherit the earth? Yes, but they will be our children."

Marvin Minsky

Abstract

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Resumen

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Acknowledgements

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus

semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Contents

List of Figures

List of Tables

List of Acronyms

1D one-dimensional

2D two-dimensional

2.5D two-and-a-half-dimensional

3D three-dimensional

AMT Amazon Mechanical Turk

BRIEF Binary Robust Independent Elementary Features

BRISK Binary Robust Invariant Scalable Keypoints

BVLC Berkeley Vision and Learning Center

CAD Computer Aided Design

CDBN Convolutional Deep Belief Network

CIFAR Canadian Institute for Advanced Research

CNN Convolutional Neural Network

ConvLSTM Convolutional LSTM

DoF Degrees of Freedom

ECC Edge-Conditioned Convolution

FREAK Fast Retina Keypoint

GCN Graph Convolutional Network

GNN Graph Neural Network

GPU Graphics Processing Unit

IR Infrarred

k-NN k-Nearest Neighbors

LIDAR Light Detection and Ranging

LSTM Long Short-Term Memory Network

MLP Multi-Layer Perceptron

MNIST Mixed National Institute of Standards and Technology

NELL Never-Ending Language Learning

OFF Object File Format

ORB Oriented FAST and Rotated BRIEF

PCD Point Cloud Data

PCL Point Cloud Library

POV Point of View

ReLU Rectified Linear Unit

RGB Red Green and Blue

RGB-D RGB-Depth

ROS Robot Operating System

SGD Stochastic Gradient Descent

SIFT Scale Invariant Feature Transform

SVM Support Vector Machine

SURF Speeded Up Robust Features

TSDF Truncated Signed Distance Function

Chapter 1

Introduction

1.1 Motivation

1.2 Approach

1.2.1 Machine Learning in Computer Vision

1.2.2 Machine Learning in Robotics

1.3 Contributions

As we already stated, this work concentrates on pushing forward three key aspects of robotic perception: object classification, semantic segmentation, and tactile sensing. In this regard, the contributions of this thesis stem from such areas and are as follows:

- We propose a Convolutional Neural Network architecture for 3D object classification which makes use of 3D representations such as point clouds or meshes by structuring them into a voxel grid. Futhermore, it is tested under difficult conditions such as noise and occlusion to gain insight about real-world situations. We also iterate over that initial architecture, creating a novel slice-based model which significantly improves over other approaches. We show the performance of these models and prove their suitability for real time object classification.

- We perform a comprehensive review of the state of the art of semantic segmentation for image and video using deep learning techniques. In such review, apart

from providing details about all existing methods and datasets, we also gather insight about weaknesses and future research. Following that train of thought, we introduce a novel large-scale dataset for various robotic perception tasks with special focus on 3D semantic segmentation.

- Finally, we show a novel Graph Neural Network architecture for tactile sensing which is able to classify the stability of robotic grasps using humanoid hands equipped with tactile sensors whose readings are interpreted as 3D graphs.

1.4 Co-Authored Papers

This thesis is the result of continuous effort throughout the last years. Such efforts have sometimes crystallized in form of journal publications, conference talks, and poster presentations. A significant part of this thesis consists of extracts from the following co-authored publications.

1.4.1 Chapter 2: 3D Object Classification

- Alberto Garcia-Garcia, Francisco Gomez-Donoso, Jose Garcia-Rodriguez, et al. "PointNet: A 3D Convolutional Neural Network for real-time object class recognition". In: *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24-29, 2016*. 2016, pp. 1578–1584. DOI: 10.1109/IJCNN.2016.7727386. URL: https://doi.org/10.1109/IJCNN.2016.7727386

- Alberto Garcia-Garcia, Jose Garcia-Rodriguez, Sergio Orts-Escolano, et al. "A study of the effect of noise and occlusion on the accuracy of convolutional neural networks applied to 3D object recognition". In: *Computer Vision and Image Understanding* 164 (2017), pp. 124–134. DOI: 10.1016/j.cviu.2017.06.006. URL: https://doi.org/10.1016/j.cviu.2017.06.006

- Francisco Gomez-Donoso, Alberto Garcia-Garcia, Jose Garcia-Rodriguez, et al. "LonchaNet: A Sliced-based CNN Architecture for Real-time 3D Object Recognition". In: *2017 International Joint Conference on Neural Networks, IJCNN 2017,*

Anchorage, Alaska, May 14-19, 2017. 2017. URL: https://ieeexplore.ieee.org/document/7965883/

1.4.2 Chapter 3: Semantic Segmentation

- Alberto Garcia-Garcia, Jose Garcia-Rodriguez, Sergio Orts-Escolano, et al. "A study of the effect of noise and occlusion on the accuracy of convolutional neural networks applied to 3D object recognition". In: *Computer Vision and Image Understanding* 164 (2017), pp. 124–134. DOI: 10.1016/j.cviu.2017.06.006. URL: https://doi.org/10.1016/j.cviu.2017.06.006

- Alberto Garcia-Garcia, Pablo Martinez-Gonzalez, Sergiu Oprea, et al. "The RobotriX: An eXtremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 6790–6797. URL: https://ieeexplore.ieee.org/abstract/document/8594495

- Pablo Martinez-Gonzalez, Sergiu Oprea, Alberto Garcia-Garcia, et al. "UnrealROX: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation". In: *CoRR* abs/1810.06936 (2018). arXiv: 1810.06936. URL: http://arxiv.org/abs/1810.06936

- Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, et al. "A Visually Plausible Grasping System for Object Manipulation and Interaction in Virtual Reality Environments". In: *CoRR* abs/1903.05238 (2019). arXiv: 1903.05238. URL: http://arxiv.org/abs/1903.05238

- Alberto Garcia-Garcia, Pablo Martinez-Gonzalez, Sergiu Oprea, et al. "The RobotriX: A Large-scale Dataset of Embodied Robots in Virtual Reality". In: *International Conference on Computer Vision and Pattern Recognition (CVPR). Workshop on 3D Scene Generation*. 2019

1.4.3 Chapter 4: Tactile Sensing

- Alberto Garcia-Garcia, Brayan Stiven Zapata-Impata, Sergio Orts-Escolano, et al. "TactileGCN: A Graph Convolutional Network for Predicting Grasp Stability with Tactile Sensors". In: *CoRR* abs/1901.06181 (2019). arXiv: 1901.06181. URL: http://arxiv.org/abs/1901.06181

- Brayan Stiven Zapata-Impata, Alberto Garcia-Garcia, Sergio Orts-Escolano, et al. "Tactile Graphs for Grasp Stability Prediction". In: *International Conference on Learning Representations (ICLR). Workshop on Representation Learning on Graphs and Manifolds*. 2019

1.4.4 Other

During the years spent working on the main topics of this thesis, several collaborations and side works were carried out that also were published either as journal papers, conference proceedings, or preprints. Those works, although not strictly related to the content of this thesis, helped in various ways: exchanging ideas that later inspired other concepts, sparking collaborations, and also expanding the knowledge of other interesting areas of research.

- Sergiu Oprea, Alberto Garcia-Garcia, Jose Garcia-Rodriguez, et al. "A Recurrent Neural Network based Schaeffer Gesture Recognition System". In: *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, Alaska, May 14-19, 2017*. 2017. URL: https://ieeexplore.ieee.org/document/7965885/

- Francisco Gomez-Donoso, Sergio Orts-Escolano, Alberto Garcia-Garcia, et al. "A robotic platform for customized and interactive rehabilitation of persons with disabilities". In: *Pattern Recognition Letters* 99 (2017), pp. 105–113. DOI: 10.1016/j.patrec.2017.05.027. URL: https://doi.org/10.1016/j.patrec.2017.05.027

- Sergiu Oprea, Alberto GarciaGarcia, Sergio OrtsEscolano, et al. "A long short-term memory based Schaeffer gesture recognition system". In: *Expert Systems* 0.0

(2017), e12247. DOI: 10.1111/exsy.12247. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.12247

- Alberto Garcia Garcia, Andreas Beckmann, and Ivo Kabadshow. "Accelerating an FMM-Based Coulomb Solver with GPUs". In: *Software for Exascale Computing-SPPEXA 2013-2015*. Springer, 2016, pp. 485–504. URL: https://link.springer.com/chapter/10.1007/978-3-319-40528-5_22

- Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, et al. "Multi-sensor 3D object dataset for object recognition with full pose estimation". In: *Neural Computing and Applications* 28 (2016), pp. 941–952. ISSN: 1433-3058. DOI: 10.1007/s00521-016-2224-9. URL: http://dx.doi.org/10.1007/s00521-016-2224-9

- Marcelo Saval-Calvo, Jorge Azorin-Lopez, Andres Fuster-Guillo, et al. "Evaluation of sampling method effects in 3D non-rigid registration". In: *Neural Computing and Applications* 28 (2016), pp. 953–967. ISSN: 1433-3058. DOI: 10.1007/s00521-016-2258-z. URL: http://dx.doi.org/10.1007/s00521-016-2258-z

- Sergio Orts-Escolano, Jose Garcia-Rodriguez, Miguel Cazorla, et al. "Bioinspired point cloud representation: 3D object tracking". In: *Neural Computing and Applications* 29 (2016), pp. 663–672. ISSN: 1433-3058. DOI: 10.1007/s00521-016-2585-0. URL: https://doi.org/10.1007/s00521-016-2585-0

- Alberto Garcia-Garcia, Sergio Orts-Escolano, Jose Garcia-Rodriguez, et al. "Interactive 3D object recognition pipeline on mobile GPGPU computing platforms using low-cost RGB-D sensors". In: *Journal of Real-Time Image Processing* 14 (2016), pp. 585–604. ISSN: 1861-8219. DOI: 10.1007/s11554-016-0607-x. URL: https://doi.org/10.1007/s11554-016-0607-x

- Higinio Mora, Jerónimo M Mora-Pascual, Alberto Garcia-Garcia, et al. "Computational analysis of distance operators for the iterative closest point algorithm". In: *PloS one* 11.10 (2016), e0164694. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0164694

- Sergio Orts-Escolano, Jose Garcia-Rodriguez, Vicente Morell, et al. "3D Surface Reconstruction of Noisy Point Clouds Using Growing Neural Gas: 3D Object/Scene Reconstruction". In: *Neural Processing Letters* 43 (2015), pp. 401–423. DOI: 10.1007/s11063-015-9421-x. URL: http://dx.doi.org/10.1007/s11063-015-9421-x

- Sergio Orts-Escolano, Jose Garcia-Rodriguez, Jose Antonio Serra-Perez, et al. "3D model reconstruction using neural gas accelerated on GPU". in: *Applied Soft Computing* 32 (2014), pp. 87–100. DOI: 10.1016/j.asoc.2015.03.042. URL: http://dx.doi.org/10.1016/j.asoc.2015.03.042

1.5 Thesis Structure

Chapter 2

Object Classification

Abstract

In this chapter, we address the problem of object classification. To approach this challenge, we rely on the geometric information provided by 3D object representations such as point clouds. Furthermore, we focus on learning-based methods to distinguish objects from different classes while capturing the variability of shape of different objects which belong to the same class. More specifically, we leverage deep learning for such task.

The chapter begins introducing and formulating the object classification task in Section 2.1 followed by a review of the most relevant literature and datasets in Sections 2.2 and 2.3. After that, we present our first proposal for 3D object classification, namely PointNet, in Section 2.4. Later, PointNet is improved and thoroughly tested in adverse conditions with noise and occlusion throughout the study in Section 2.5. Next, LonchaNet is introduced in Section 2.6 as the last iteration of our system that incorporates all the lessons learned by the previous work. Finally, Section 2.7 draws conclusions and sets future lines of research.

2.1 Introduction

Object classification is fundamental to computer vision and despite the progress achieved during the last years, it still remains a challenging area of research. Arguably, most of the interest in object classification is due to its usefulness for robotics.

In that regard, recognizing objects is one of the problems that must be solved to achieve total visual scene understanding. Such deeper and better knowledge of the environment eases and enables the execution of a wide variety of more complex tasks. For instance, accurately recognizing objects in a room can be extremely useful for any robotic system that navigates within indoor environments. Due to the unstructured nature of those environments, autonomous robots need to do reasoning grounded in the dynamic real world. In other words, they need to understand the information captured by their sensors to perform tasks such as grasping, navigation, mapping, or even providing humans with information about their surroundings. Identifying the classes to which objects belong is one key step to enhance the aforementioned capabilities.

Despite the easy intuitive interpretation of the problem, its inherent difficulty can be misleading. We humans recognize numerous objects in difficult settings (e.g., different points of view, occlusion, or clutter) with little to no effort. However, approaching that problem is not that easy for a computer and taking into account all the possible settings and combinations of external factors renders this task a difficult one to solve efficiently and with high precision (which is often required in numerous application scenarios).

From a formal point of view, the object classification task can be formulated as follows: given an image $\mathcal{I}^{H \times W}$ in which an object $\mathcal{O}$ appears, which can be either a gray-scale or RGB array of W pixels in width and H pixels in height, the goal is to predict the class of the object $\mathcal{L}_{\mathcal{O}}$ from a set of N predefined object classes $\mathcal{L} = \{\mathcal{L}_0, \mathcal{L}_1, ..., \mathcal{L}_{N-1}\}$.

Most of the classic literature of this topic tackled such problem by devising hand-crafted feature descriptors that are extracted on certain keypoints detected over the bidimensional image and later used either to compare them against pre-existing object descriptors in a database to match them to a certain class or either to feed them as input to a shallow machine learning architecture that learns to classify those descriptors to predict the class of the object that appears in the image. That paradigm shifted

recently due to the success of deep learning architectures that are able to exploit their feature learning capabilities to avoid the need of hand engineering descriptors while achieving unprecedented accuracy levels. Furthermore, the adoption and spread of depth sensors has also added a literally new dimension to learn from to boost performance. The approaches introduced in this thesis are part of that cutting-edge trend that takes advantage of the additional geometric information facilitated by commodity range scanners to perform learning over them using deep architectures. A more detailed review of the field, from the very beginning to the current trends using 3D data and deep neural networks, is performed in Section 2.2.

Apart from the methods, data also plays a key role in object classification. As methods have evolved, so have datasets. Due to the increasing needs imposed by data-driven approaches, datasets have grown larger, more varied, and richer. That progress has enabled the development of new ways of solving the problem, e.g., using three-dimensional data. Section 2.3 briefly reviews the evolution of such datasets and describes the data that will be used throughout this chapter.

After reviewing the literature and introducing the data we are going to use, we start describing our first approach to perform object classification using 3D data, namely PointNet, capable of learning object classes from point clouds discretized as occupancy grids with uniform voxel grids in the tridimensional space. Section 2.4 describes this architecture, its data representation, and also benchmarks it on a standard 3D object classification dataset to validate it.

Following that, Section 2.5 analyzes how noise and occlusion impact such 3D deep learning architecture and the importance of the data representation when dealing with such adverse conditions that commonly appear in the real world. In that study, we also propose minor changes to the architecture and the representation themselves that significantly boost accuracy with regard to the originally proposed PointNet.

At last, Section 2.6 takes all the lessons learned from the initial PointNet proposal and the extensive study to introduce a novel slice-based architecture to tackle the 3D object class recognition problem, LonchaNet, which achieved state of the art results in a standard benchmark.

In the end, Section 2.7 concludes this chapter by summarizing the insights gathered while discussing the proposed approaches. Furthermore, we also our main flaws and how to improve upon them besides from proposing new proposing future lines of research for 3D object classification.

2.2 Related Works

Since the very beginning of computer vision, a considerable amount of effort has been directed towards achieving robust object recognition systems [21]. This was mainly due to the fact that recognizing objects is a key capability required by robots to operate autonomously in unstructured, real-world environments. That continuous endeavor configured object classification as an ever-evolving area which has followed the general trend of computer vision, i.e., moving from hand-crafted features to trainable feature extractors, and which has also benefited from the improvements on imaging hardware, e.g., depth information from range scanners. In this section, we briefly review that evolution in order to put our proposal in context.

2.2.1 Traditional Approaches

Object recognition has been traditionally dominated by feature-based methods. This approach relies on extracting features, i.e., pieces of information which describe simple but significant properties of the objects. Those features are encoded into *descriptors* such as Scale Invariant Feature Transform (SIFT)[22], Speeded Up Robust Features (SURF)[23], Binary Robust Independent Elementary Features (BRIEF)[24], Binary Robust Invariant Scalable Keypoints (BRISK)[25], Oriented FAST and Rotated BRIEF (ORB)[26], or Fast Retina Keypoint (FREAK)[27] to name a few. After extracting those descriptors, machine learning techniques are applied to train a system with them so that it becomes able to classify features extracted from unknown instances. Based on the types of features, these methods can be divided into two categories: global or local feature-based methods. Global ones are characterized by dealing with the object as a whole; they define a set of features which completely encompass the object and

describe it effectively. On the other hand, local methods describe local patches of the object, those regions are located around highly distinctive spots named *keypoints*.

Real-world scenes tend to be unstructured environments. This implies that object recognition systems must not be affected by clutter or partial occlusions. In addition, they should be invariant to illumination, transforms, and object variations. Those are the main reasons why local surface feature-based methods have been popular and successful during the last years – since they do not need the whole object to describe it properly, they are able to cope with cluttered environments and occlusions [28].

2.2.2 3D Object Recognition

Traditionally, those object recognition systems made use of 2D images with intensity or color information, i.e., Red Green and Blue (RGB) images. However, technological advances made during the last years have caused a huge increase in the usage of 3D information. The field of computer vision in general, and object recognition research in particular, have been slowly but surely moving towards including this richer information into their algorithms.

Nowadays, the use of 3D information for this task is in a state of continuous evolution, still far behind, in terms of maturity, from the systems that make use of 2D images. Nevertheless, the use of 2D information exhibits a handful of problems which hinder the development of robust object recognition systems. Oppositely, the use of range images or point clouds, which provide 2.5D or 3D information respectively, presents many significant benefits over traditional 2D-based systems. Some of the main advantages are the following ones [29]: (1) they provide geometrical information thus removing surface ambiguities, (2) many of the features that can be extracted are not affected by illumination or even scale changes, (3) pose estimation is more accurate due to the increased amount of surface information. Therefore, the use of 3D data has become a solid choice to overcome the inherent hurdles of traditional 2D methods.

However, despite all the advantageous assets of 3D data, researchers had to overcome certain difficulties or drawbacks. On the one hand, sensors capable of providing 3D were expensive, limited, and performed poorly in many cases. The advent of

low-cost 3D acquisition systems, e.g., Microsoft Kinect, enabled a widespread adoption of these kind of sensors thanks to their accessibility and affordability. On the other hand, 3D object recognition systems are computationally intensive due to the increased dimensionality. In this regard, advances in computing devices like GPUs provided enough computational horsepower to run those algorithms in an efficient manner. In addition, the availability of low-power GPU computing devices like NVIDIA's Jetson has supposed a significant step towards deploying robust and powerful object recognition systems in mobile robotic platforms.

The combination of those three factors (the advantages of 3D data, low-cost sensors, and parallel computing devices) transformed the field of computer vision in general, and object recognition in particular. As we can see in Figure 2.1, there has been a significant dominance of 3D over 2D research in computer vision since the year 2000.

Therefore, creating a robust 3D object recognition system, which is also able to work in real time, became one of the main goals towards for computer vision researchers [30]. There exist many reviews about 3D object recognition in the literature, including the seminal works of Besl and Rain [31], Brady et al. [32], Arman et al. [33], Campbell and Flynn [34], and Mamic and Bennamoun [35]. All of them perform a general review of the 3D object recognition problem with varying levels of detail and different points of view. The work of Guo et al. [29] is characterized by its comprehensive analysis of different local surface feature-based 3D object recognition methods which were

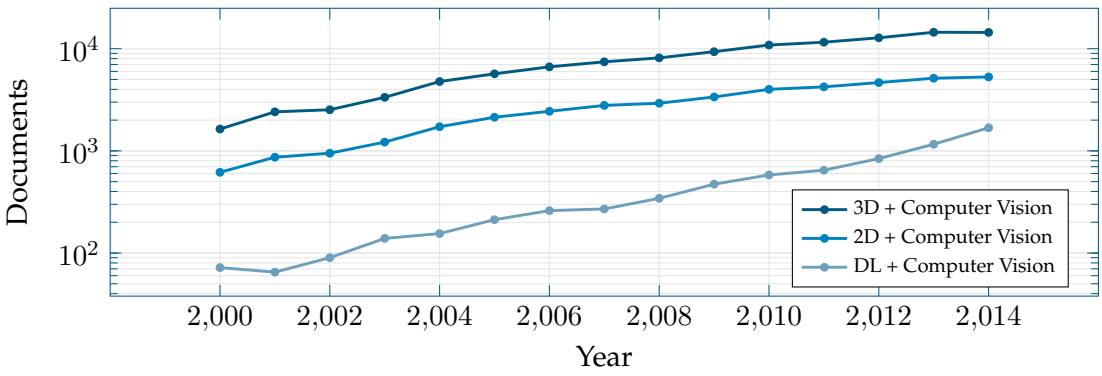


Figure 2.1: Evolution of the number of academic documents containing the terms 2D, 3D, and Deep Learning together with *Computer Vision*. Search terms statistics obtained from scopus.com.

published between the years 1992 and 2013. In that review, they explain the main advantages and drawbacks of each one of them. They also provide an in-depth survey of various techniques used in each phase of a 3D object recognition pipeline, from the keypoint extraction stage to the surface matching one, including the extraction of local surface descriptors. The review is specially remarkable due to its freshness and level of detail. It is important to remark that all the described methods make use of carefully designed feature descriptors by experts in the field.

2.2.3 Deep Learning

From the earliest days of computer vision, the aim of researchers has been to replace hand-crafted feature descriptors, which require domain expertise and engineering skills, with multilayer networks able to learn them automatically by using a general-purpose training algorithm [36]. The solution for this problem was discovered during the 1970s and 1980s by different research groups independently [37][38][39]. This gave birth to a whole new branch of machine learning named deep learning.

Deep learning architectures usually consist of a multilayer stack of hierarchical learning modules which compute non-linear input-output mappings. Those modules are just functions of the input with a set of internal weights. The input of each layer in the stack is transformed, using the functions defined by the modules, to increase the selectivity and invariance of the representation. The backpropagation procedure is used to train those multilayer architectures by propagating gradients through all the modules. In the end, deep learning applications use feedforward neural network architectures which learn to map a fixed-size input, e.g., an image, to a fixed-size output, typically a vector containing a probability for each one of the possible categories [36].

Figure 2.2 shows some sample filter modules automatically learned by training one of the most successful deep learning architectures: the deep convolutional neural network proposed by Krizhevsky et al. [40] to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 [42] contest into 1000 different classes.

In spite of the fact that these kind of architectures showed a huge potential for solving many computer vision problems, they were ignored by the computer vision community. In the latter years, certain breakthrough works revived the interest in deep

learning architectures [36]. Recent studies proved that local minima are not an issue with large neural networks. Following a set of seminal works for the field on training deep learning networks [43][44], a group of researchers from the Canadian Institute for Advanced Research (CIFAR) introduced unsupervised learning procedures to create layers of feature detectors without labelled data, they also pre-trained several layers and added a final layer of output units; the system was tuned using backpropagation and achieved a remarkable performance when applied to the handwritten digit recognition or pedestrian detection problems [45]. In addition, the advent of GPUs, which

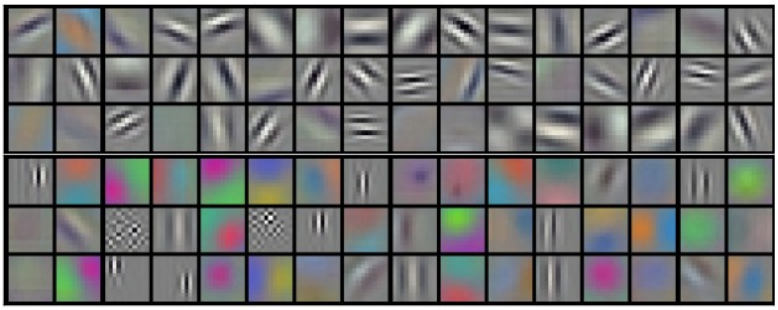


Figure 2.2: Filters learned by the network proposed by Krizhevsky et al. [40]. Each of the 96 filters shown is of size $11 \times 11 \times 3$. The filters have clearly learned to detect edges of various orientations and they resemble Gabor filters. Image analysis using that kind of filters is thought to be similar to perception in the human visual system [41].

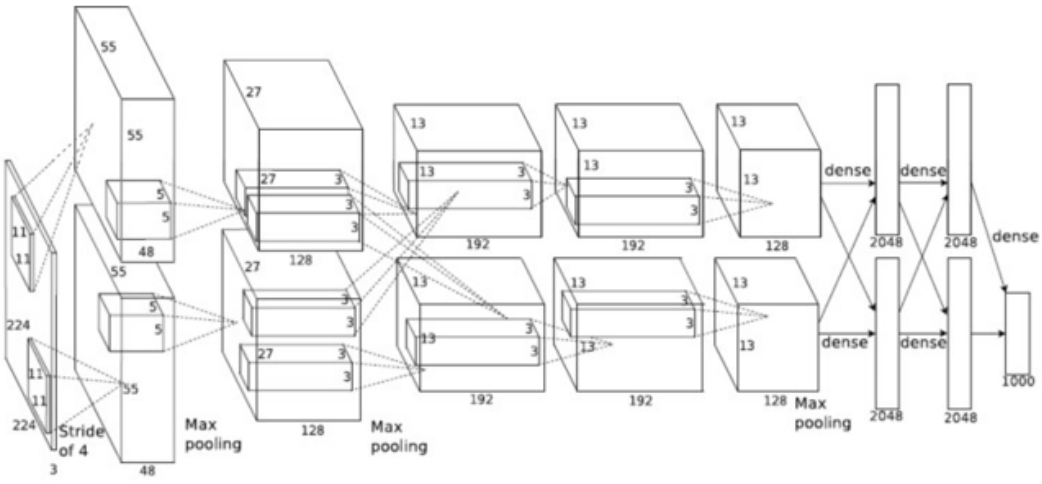


Figure 2.3: Illustration of the architecture of the aforementioned CNN proposed by Krizhevsky et al.[40] for the ImageNet challenge. Besides the normal components, e.g., convolutional, pooling, and fully connected layers, this network features two different paths. One GPU runs the layers at the top while the other runs the layers at the bottom.

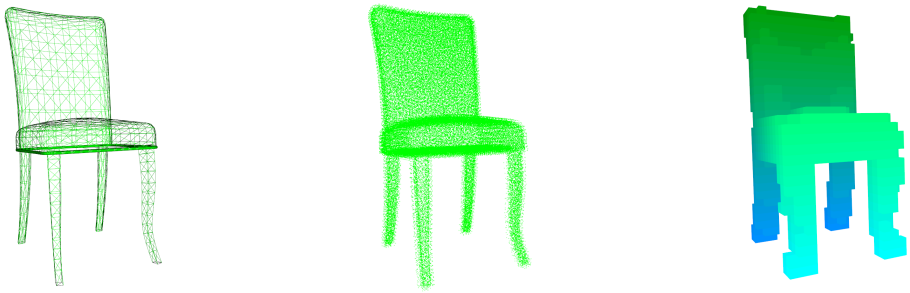
were easily programmable and extremely efficient for parallel problems, made possible the training of huge networks in acceptable time spans [46].

All those contributions to the field led to the birth of probably the most important milestone regarding deep learning: the Convolutional Neural Network (CNN). This special kind of deep network was designed to process data in form of multiple arrays and gained popularity because of its many practical successes. This was due to the fact that they were easier to train and generalized far better than previous models. The architecture of a typical CNN is composed by many stages of convolutional layers followed by pooling ones and non-linearity Rectified Linear Unit (ReLU) filters; in the end, convolutional and fully connected layers are stacked. The key idea behind using this stack of layers is to exploit the property that many natural signals are compositional hierarchies, in which higher-level features are obtained by composing lower-level ones. Figure 2.3 shows a typical architecture of a CNN.

2.2.4 Volumetric Representations

Most of the literature focuses on how CNNs can learn filters and recognize objects using 2D images as input. In this chapter, we will explore the usage of 3D information to feed the network. For this purpose, we need volumetric representations for such information. Arguably, the most popular representations for volumetric data are 3D meshes or point clouds, shown in Figure 2.4a and 2.4b respectively. A mesh consists of a collection of vertices (points in a three-dimensional (3D) coordinate system), edges (connections between those vertices), and faces (closed sets of edges, usually triangles) that defines the shape of an object. A point cloud is just a set of points defined by x, y, and z coordinates in a three-dimensional coordinate system that model the surface of an object. However, those representations are unbounded and barely structured since they contain an arbitrary number of components, e.g., vertices or points, and no particular ordering is enforced for those entities.

This fact poses a problem since CNNs require a fixed-size representation for the input data. In order to overcome this limitation, alternative volumetric representations must be used to provide samples to the network for both training and testing. The

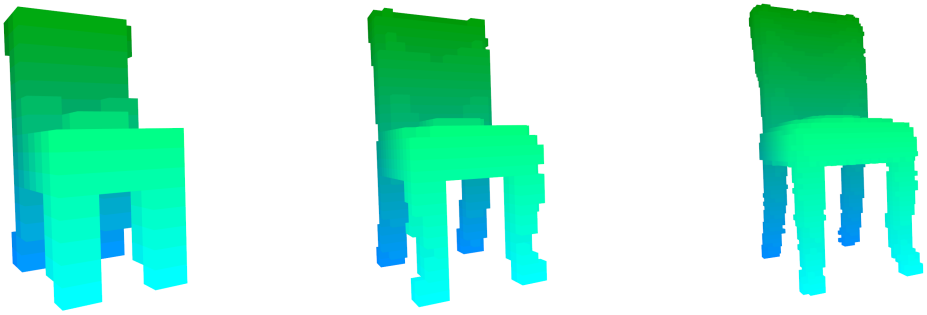


(a) Mesh **(b)** Point cloud **(c)** Voxel grid

Figure 2.4: Common volumetric representations: polygonal mesh (a), point cloud (b), and voxel grid (c) of a chair model (which is color coded by height and depth).

most common volumetric representation which allows a structured and bounded definition of an object shape is the voxel grid. A voxel (word contraction of *volume element* or *volumetric pixel*) is the 3D equivalent to a 2D pixel, i.e., it is the minimal unit of a three-dimensional matrix. A volumetric object can be represented as a 3D matrix of voxels, whose positions are relative to other voxels while points and polygons must be represented explicitly by 3D coordinates. In this regard, voxels are able to efficiently represent regularly sampled 3D spaces that are also non-homogeneously filled, while meshes and point clouds are good for representing 3D shapes with empty or homogeneously filled space. It is important to notice that a voxel is just a data point in a three-dimensional grid, so its value may represent many different properties. The most popular and simple voxel grid type is the binary one (see Figure 2.4c) in which each voxel contains a binary value depending on whether the object's surface intersects or is partially contained in the voxel's volume.

Despite the fact that a binary voxel grid representation allows us to feed a CNN with volumetric data coming from different sources (point clouds provided by range sensors or polygonal meshes from 3D models can be easily converted into voxel grids) a significant amount of information from the original representation is lost. This loss depends on the resolution of the grid, i.e., the voxel size which is usually referred as the *leaf size*. Figure 2.5 shows how the resolution of the voxel grid can be tuned to obtain more accurate or more compact volumetric representations. Although an arbitrary precision can be obtained by changing the leaf size, it is hard to determine a specific



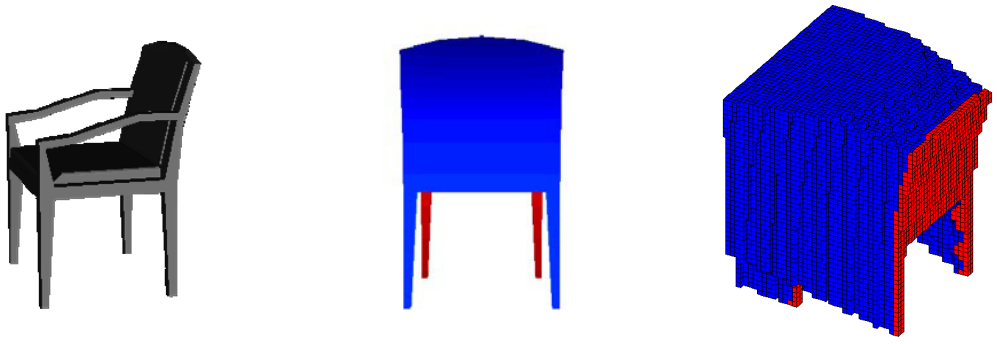
(a) $15 \times 15 \times 15$ **(b)** $30 \times 30 \times 30$ **(c)** $60 \times 60 \times 60$

Figure 2.5: Effect of the leaf size on binary voxel grids. All grids have the same cubic size: $300 \times 300 \times 300$ units. Leaf sizes vary from 5, 10, and 20 units, resulting in binary grids of $15 \times 15 \times 15$ (a), $30 \times 30 \times 30$ (b), and $60 \times 60 \times 60$ voxels (c) respectively.

size which describes with enough detail all the possible inputs for the CNN without sacrificing the compactness of the grid.

Apart from those basic representations, others which maintain the properties of the voxel grid, but include additional information in the values of the cells can be used as well. Here we briefly review the most popular and successful volumetric representations for 3D data that have been used to feed CNNs for object recognition purposes.

The first step was taken by Wu *et al.* [47]; their work *3DShapeNets* was the first to apply CNNs to pure 3D representations. Their proposal (shown in Figure 2.6) represents



(a) Object model **(b)** Depth map **(c)** Voxel grid

Figure 2.6: *3DShapeNets* representation proposed by Wu *et al.* as shown in their paper [47]. An object (a) is captured from a certain point of view and a depth map is generated (b) which is in turn used to generate a point cloud that will be represented as a voxel grid (c) with empty voxels (in white, not represented), unknown voxels (in blue), and surface or occupied voxels (red).

(a) Object **(b)** Point cloud **(c)** TSDF grid

Figure 2.7: TSDF representation proposed by Song and Xiao as shown in their paper [48]. An object (a) is captured by a range sensor as a point cloud (b) and then a TSDF grid is generated (red indicates the voxel is in front of surfaces and blue indicates the voxel is behind the surface; the intensity of the color represents the TSDF value).

3D shapes, from captured depth maps that are later transformed into point clouds, as 3D voxel grids of size $30 \times 30 \times 30$ voxels – $24 \times 24 \times 24$ data voxels plus 3 extra ones of padding in both directions to reduce convolution artifacts – which can represent free space, occupied space (the shape itself), and unknown or occluded space depending on the point of view. Neither the grid generation process, nor the leaf size is described but the voxel grid relies on prior object segmentation.

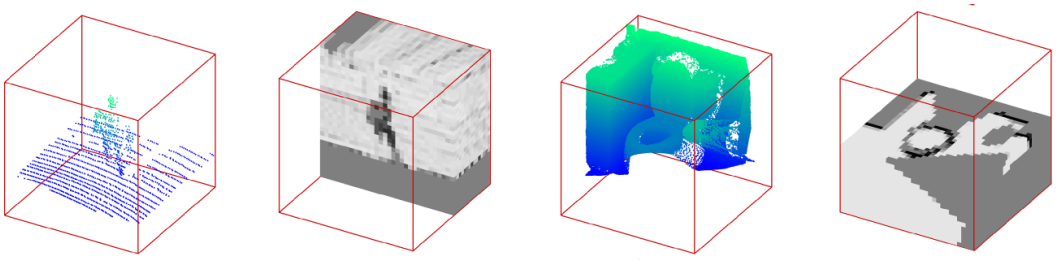
Song and Xiao [48] proposed to adopt a directional TSDF encoding which takes a depth map as input and outputs a volumetric representation. They divide a 3D space using an equally spaced voxel grid in which each cell holds a three-dimensional vector that records the shortest distance between the voxel center and the three-dimensional surface in three directions. In addition, the value is clipped by 2δ, being δ the grid size in each dimension. A $30 \times 30 \times 30$ voxels grid is fitted to a previously segmented object candidate. Figure 2.7 shows a graphical representation of this approach.

Maturana and Scherer [49] use occupancy grids in *VoxNet* to maintain a probabilistic estimate of the occupancy of each voxel to represent a 3D shape. This estimate is a function of the sensor data and prior knowledge. They propose three different occupancy models: binary, density, and hit. The binary and density models make use of raytracing to compute the number of hits and pass-throughs for each voxel. The former one assumes that each voxel has a binary state, occupied or unoccupied. The latter one assumes that each voxel has a continuous density, based on the probability it will block

a sensor beam. The hit grid ignores the difference between unknown and free space, only considering hits; it discards information but does not require the use of raytracing so it is highly efficient in comparison with the other methods. They also propose two different grids for Light Detection and Ranging (LIDAR) and RGB-D sensor data. For the RGB-D case, they use a fixed occupancy grid of $32 \times 32 \times 32$ voxels, making the object of interest – obtained by a segmentation algorithm or given by a sliding box – occupy a subvolume of $24 \times 24 \times 24$ voxels. The z axis of the grid is aligned with the direction of gravity. Figure 2.8 shows the occupancy grids used by *VoxNet*.

2.2.5 Our Proposal in Context

Our proposal builds upon the successes in the literature which suggested that applying deep learning techniques to 3D information to solve the object class recognition problem exhibits potential to raise the bar in terms of performance. Firstly, we introduce a novel way for representing the 3D input data, which is based on point density occupancy grids, and we integrate it into a CNN architecture. The focus of this first iteration is to prove that simple representations and architectures for 3D data can perform reasonably well while keeping computational cost at bay to enable real-time class recognition. Secondly, we carry out an in-depth study of the effect of adverse conditions that characterize real-world scenarios – such as noise caused by the sensor and occlusions due to the positions of the objects in the scene – on the performance of CNNs



(a) LIDAR data **(b)** Voxnet grid **(c)** RGBD data **(d)** Voxnet grid

Figure 2.8: Volumetric occupancy grid representation used by *VoxNet* as shown in their paper [49]. For LIDAR data (a) a voxel size of 0.1m^3 is used to create a $32 \times 32 \times 32$ grid (b). For RGB-D data (**??**), the resolution is chosen so the object occupies a subvolume of $24 \times 24 \times 24$ voxels in a $32 \times 32 \times 32$ grid (d).

applied to 3D object class recognition. In this case, our target is the assessment of iterative improvements applied to the previous architecture and representations to show that not only they can perform real-time recognition at reasonable accuracy but also maintain it under adverse circumstances. At last, we take a sideways step to propose a novel approach that uses multiple two-dimensional (2D) cross-section views of 3D models for 3D object class recognition.

2.3 Datasets

Deep neural network architectures are usually composed by many layers which in turn mean many weights to be learned. Because of that, there is a strong need of large-scale datasets to train those networks in order to avoid overfitting the model to the input data. Nowadays, large-scale databases of real-world 3D objects are scarce, some of them do not have that high number of objects [50][51][52], or were incomplete by the time this work was performed [53]. A possible workaround to this problem consists of using Computer Aided Design (CAD) model databases – which are virtually unlimited – and processing those models to simulate real-world data.

The *Princeton ModelNet* project is one of the most popular large-scale 3D object dataset. Its goal, as their authors state, is to provide researchers with a comprehensive clean collection of 3D CAD models for objects, which were obtained via online search engines. Employees from the Amazon Mechanical Turk (AMT) service were hired to classify over 150 000 models into 662 different categories.

At the moment, there are two versions of this dataset publicly available for download[1]: *ModelNet-10* and *ModelNet-40*. Those are subsets of the original dataset which only provide the 10 and 40 most popular object categories respectively. These subsets are specially clean versions of the complete dataset.

On the one hand, ModelNet-10 is composed of a collection of over 5000 models classified into 10 categories and divided into training and test splits. In addition, the orientation of all CAD models of the dataset was manually aligned. On the other hand, ModelNet-40 features over 9800 models classified into 40 categories, also including training and test sets. However, the orientations of its models are not aligned as they

[1] http://modelnet.cs.princeton.edu/

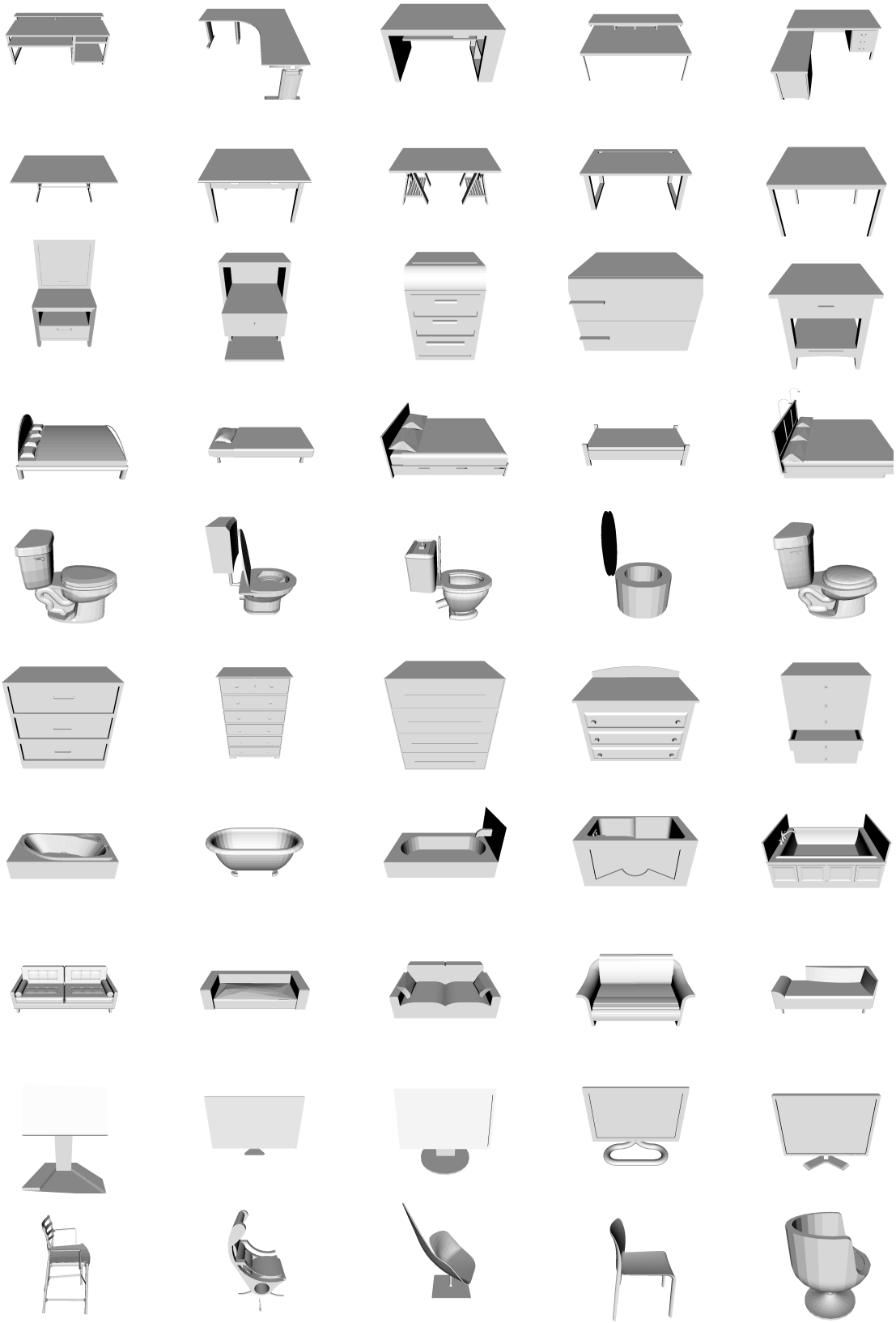


Figure 2.9: ModelNet10 samples.

are in ModelNet-10. Figure 2.9 shows some model examples from ModelNet-10. Figure 2.10 and Table 2.1 show the model distribution per each class of both subsets taking into account the training and test splits.

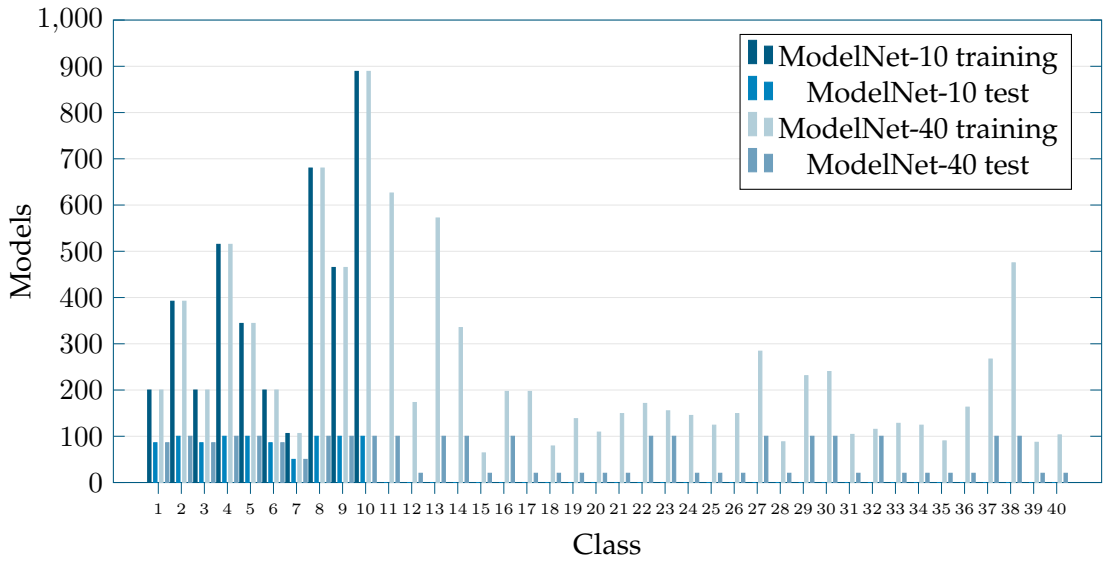


Figure 2.10: Model distribution per object class or category for both ModelNet-10 and ModelNet-40 training and test splits.

Since the final goal of the CNNs is to provide means to recognize objects onboard a mobile robotic platform which features RGB-Depth (RGB-D) sensors, it is logical to transform the full mesh representation provided by the dataset into the representation that our networks will deal with. RGB-D cameras output depth maps which can be used to generate 3D point clouds of the scene viewed by the camera. In this regard, we will transform each CAD object of the dataset into partial point clouds from different Point of Views (POVs).

For this purpose, we converted the OFF models into Point Cloud Data (PCD) clouds using a raytracing-based process. The object is placed in the center of a 3D sphere, which is tessellated to a certain level, and a virtual camera pointing to the center of the sphere is placed in each vertex of that truncated icosahedron. Then those partial views are rendered and their z-buffer data, which contains the depth information, is used to generate point clouds from each POV. In the end, those views are translated and rotated, depending on their POV, and merged into a cloud for the full object.

Figure 2.11 shows a diagram of the aforementioned process. For the conversion, we

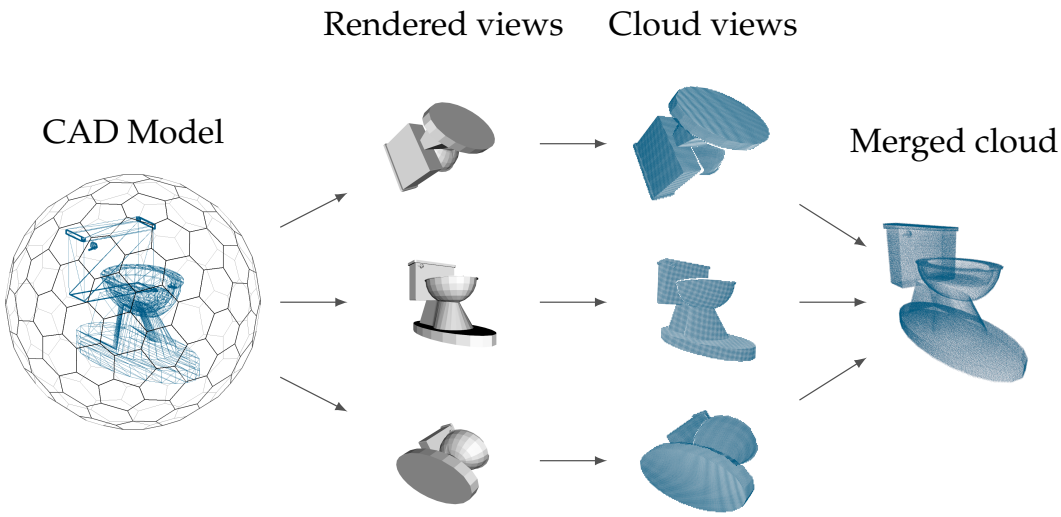


Figure 2.11: From CAD models to point clouds. The object is placed in the center of a tessellated sphere, views are rendered placing a virtual camera in each vertex of the icosahedron, the z-buffer data of those views is used to generate point clouds, and the point clouds are transformed and merged at last.

used the first tessellation level of the sphere, which generates 42 vertices or POVs. A resolution of 256×256 pixels was used for rendering the views. A voxel grid filter with a leaf size of $0.7 \times 0.7 \times 0.7$ units is applied to the merged cloud to equalize the point density, which is higher in certain zones due to view overlapping.

2.4 PointNet

In this first iteration of our efforts towards an object class recognition solution, we propose a system that takes a point cloud of an object as input and predicts its class label by leveraging two novel components: (1) a point density volumetric grid to estimate spatial occupancy inside each voxel, and (2) a 3D-CNN which is trained to predict object classes from those grids. The occupancy grid – inspired by VoxNet's [49] occupancy models that rely on probabilistic estimates – provides a compact representation of the object's 3D information originally present in the point cloud that can be fed to the CNN architecture, which in turn computes a label for that sample, i.e., predicts the class of the object. The components of this approach, namely *PointNet*, will be described throughout this section. Firstly, the proposed data representation is introduced

		ModelNet-10		*ModelNet-40*	
ID	**Category**	**Training Set**	**Test set**	**Training Set**	**Test set**
1	Desk	200	86	200	86
2	Table	392	100	392	100
3	Nighstand	200	86	200	86
4	Bed	515	100	515	100
5	Toilet	344	100	344	100
6	Dresser	200	86	200	86
7	Bathtub	106	50	106	50
8	Sofa	680	100	680	100
9	Monitor	465	100	465	100
10	Chair	889	100	889	100
11	Airplane	-	-	626	100
12	Bench	-	-	173	20
13	Bookshelf	-	-	572	100
14	Bottle	-	-	335	100
15	Bowl	-	-	64	20
16	Car	-	-	197	100
17	Cone	-	-	167	20
18	Cup	-	-	79	20
19	Curtain	-	-	138	20
20	Door	-	-	109	20
21	Flower Pot	-	-	149	20
22	Glass Box	-	-	171	100
23	Guitar	-	-	155	100
24	Keyboard	-	-	145	20
25	Lamp	-	-	124	20
26	Laptop	-	-	149	20
27	Mantel	-	-	284	100
28	Person	-	-	88	20
29	Piano	-	-	231	100
30	Plant	-	-	240	100
31	Radio	-	-	104	20
32	Range Hood	-	-	115	100
33	Sink	-	-	128	20
34	Stairs	-	-	124	20
35	Stool	-	-	90	20
36	Tent	-	-	163	20
37	TV Stand	-	-	267	100
38	Vase	-	-	475	100
39	Wardrobe	-	-	87	20
40	X-Box	-	-	103	20

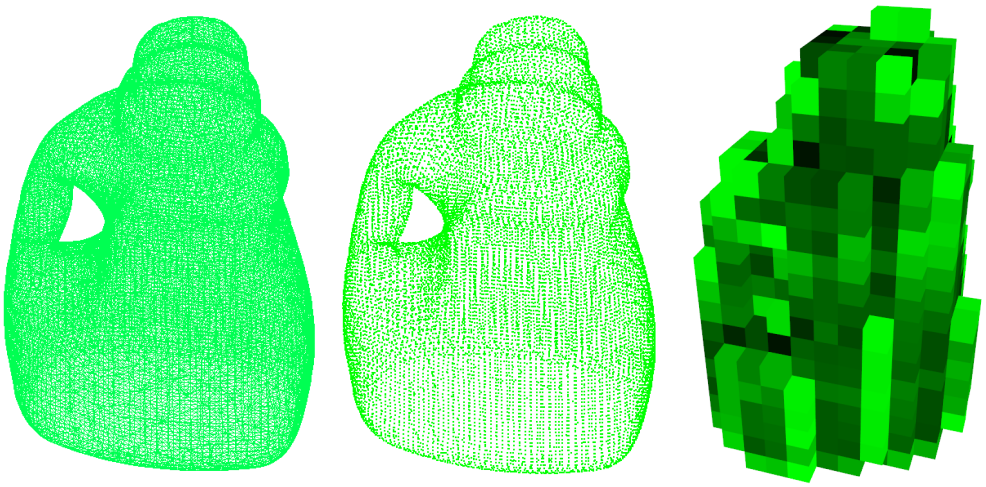
Table 2.1: ModelNet-10 and ModelNet-40 distributions.

in Section 2.4.1. Secondly, the network itself is presented in Section 2.4.2. Next, we carry out a set of experiments to validate the system in Section 2.4.3. At last, in Section 2.4.4, we draw conclusions about this work and set the stage for the next iteration.

2.4.1 Data Representation

As we mentioned before, our proposed architecture takes a point cloud of an object as input to recognize it. However, point clouds are unstructured representations that cannot be easily handled by common CNN architectures due to the lack of a matrix-like organization. The most straightforward way to apply formal convolutions to that unstructured space is to impose a certain organization into it. Occupancy grids expose a compact representation of the volumetric space; they stand between meshes or clouds, which offer rich but unstructured information, and voxelized representations, with packed but poor information. At that midpoint, occupancy grids provide important shape cues to perform learning while enabling an efficient processing of that information thanks to their matrix-like organization.

As we previously reviewed in Section 2.2, certain 3D deep learning architectures make use of occupancy grids as a representation for the input data. For instance, 3D ShapeNets [47] is a Convolutional Deep Belief Network (CDBN) which represents a 3D shape as a $30 \times 30 \times 30$ binary tensor in which a one indicates that a voxel intersects the mesh surface, and a zero represents empty space. VoxNet [49] introduces three different occupancy grids that employ 3D ray tracing to compute the number of beams hitting or passing each voxel and then use that information to compute the value of each voxel depending on the chosen model: a binary occupancy grid using probabilistic estimates, a density grid in which each voxel holds a value corresponding to the probability that it will block a sensor beam, and a hit grid that only considers hits thus ignoring empty or unknown space. The binary and density grids proposed by Maturana *et al.* [49] differentiate unknown and empty space, whilst the hit grid and the binary tensor do not. VoxNet's occupancy grid outperforms 3D ShapeNets in terms of accuracy in the ModelNet challenge for the 3D-centric approaches described above. However, ray tracing grids considerably harmed performance in terms of execution time so that other approaches must be considered for a real-time implementation.



(a) Mesh **(b)** Cloud **(c)** Grid

Figure 2.12: Various 3D representations for an object. A mesh (a) is transformed into a point cloud (b), and that cloud is processed to obtain a voxelized occupancy grid (c). The occupancy grid shown in this figure is a cube of $30 \times 30 \times 30$ voxels. Each voxel of that cube holds the point density inside its volume. In this case, dark voxels indicate high density whilst bright ones are low density volumes.

With PointNet, we propose an occupancy grid inspired by the aforementioned successes but aiming to maintain a reasonable accuracy while enabling a real-time implementation. In our volumetric representation, each point of a cloud is mapped to a voxel of a fixed-size occupancy grid. Before performing that mapping, the object cloud is scaled to fit the grid. Each voxel will hold a value representing the number of points mapped to itself. At last, the values held by each cell are normalized. Figure 2.12 shows the derivation of the proposed occupancy grid representation from other typical tridimensional representations of a sample object.

2.4.2 Network Architecture

As we have previously stated, CNNs have proven to be very useful for recognizing and classifying objects in 2D images. A convolutional layer can recognize basic patterns such as corners or planes and if we stack several of them they can learn a topology of hierarchical filters. Furthermore, the composition of several of these regions can define a feature of a more complex object. By doing that, a combination of various filters is able to recognize a full object. We apply this approach used in 2D images to 3D recognition. The deep architecture featured by *PointNet* is represented in Figure 2.13.

2.4.3 Experiments

In order to evaluate the performance of our proposed representation and network, we carried out a set of experiments using the previously described ModelNet-10 dataset. Those experiments aim to prove that the network is able to achieve reasonable accuracy while still being able to execute in real time. Furthermore, we also study ways to increase our performance by analyzing our data, e.g., the balance between classes in the dataset and the possible sources of confusion that make learning harder. To that end, we first describe our data generation process as well as the experimentation methodology (implementation details and test setup); then we show our experiments together with a discussion for each one of them to keep on building and experimenting in an incremental way.

Data Generation

We generated point clouds to feed PointNet following the procedure described in Section 2.3. Figure 2.14 illustrates the result of the aforementioned process. After that, the resulting point clouds are used to train, randomizing the order of the models, and test the system taking into account the corresponding splits provided by ModelNet-10.

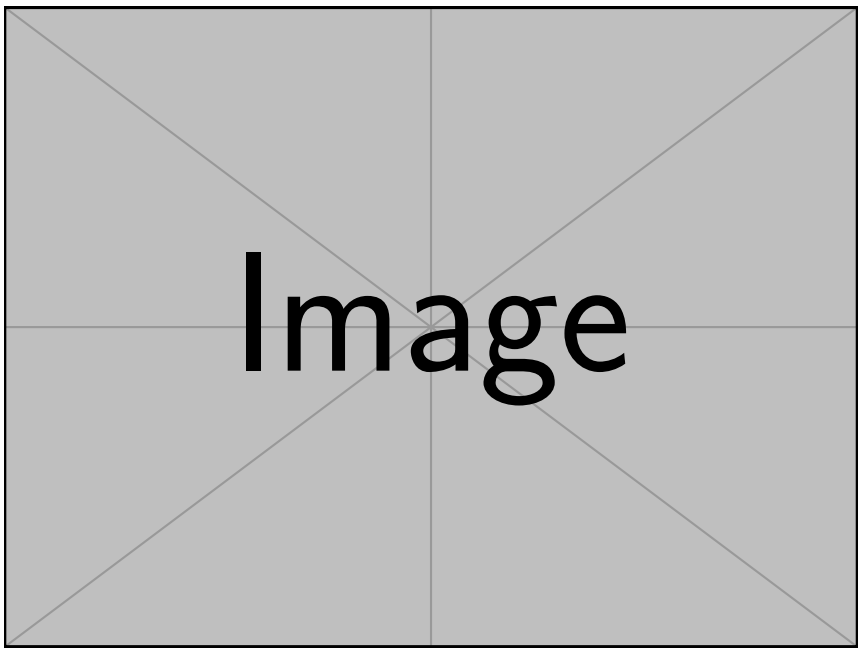
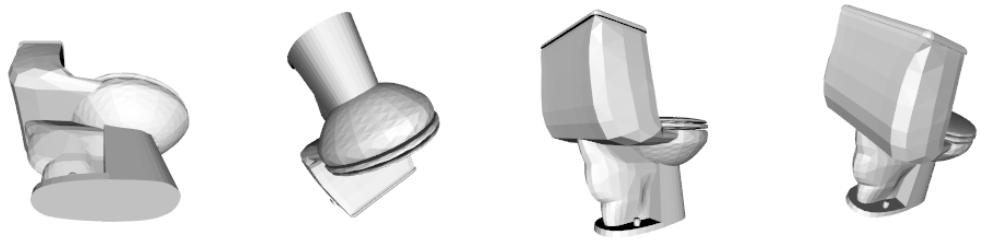


Figure 2.13: PointNet's 3D CNN architecture. [MISSINGDETAILS]

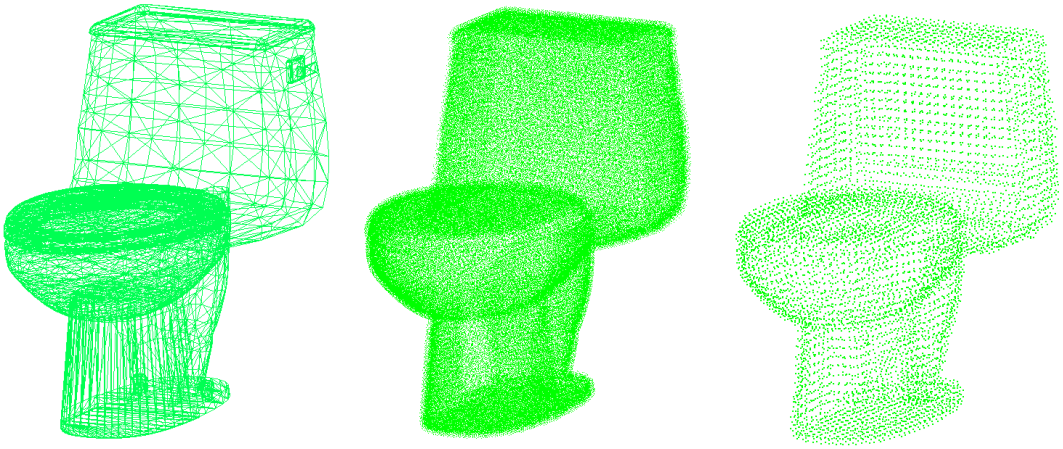
Implementation and Setup

This architecture was implemented using the Point Cloud Library (PCL) [54][55] which provides state-of-the-art algorithm implementations for 3D point cloud processing and Caffe [56], a deep learning framework developed and maintained by the Berkeley Vision and Learning Center (BVLC) and an active community of contributors on GitHub. This BSD-licensed C++ library enables researchers to design, train, and deploy CNN architectures efficiently, mainly thanks to its drop-in integration of NVIDIA cuDNN [57] to take advantage of GPU acceleration.

All the timings and results were obtained by performing the experiments in the following test setup: Intel Core i5-3570 with 8 GB of 1600 MHz DD3 RAM on an ASUS P8H77-M PRO motherboard (Intel H77 chipset). Additionally, the system includes an



(a) Rendered views.



(b) Mesh. **(c)** Cloud. **(d)** Downsampled.

Figure 2.14: Dataset model processing example to generate the point clouds for PointNet. Some rendered views of a toilet model are shown in (a). The original OFF mesh is shown in (b). The generated point cloud after merging all points of view is shown in (c), and (d) shows the downsampled cloud using a voxel grid filter with a leaf size of $0.7 \times 0.7 \times 0.7$.

NVIDIA Tesla K20 GPU, and a Seagate Barracuda 7200.14 secondary storage. Caffe RC2 was run over ElementaryOS Freya 0.3.1, an Ubuntu-based Linux distribution. It was compiled using CMake 2.8.7, g++ 4.8.2, CUDA 7.0, and cuDNN v3.

Results and Discussion

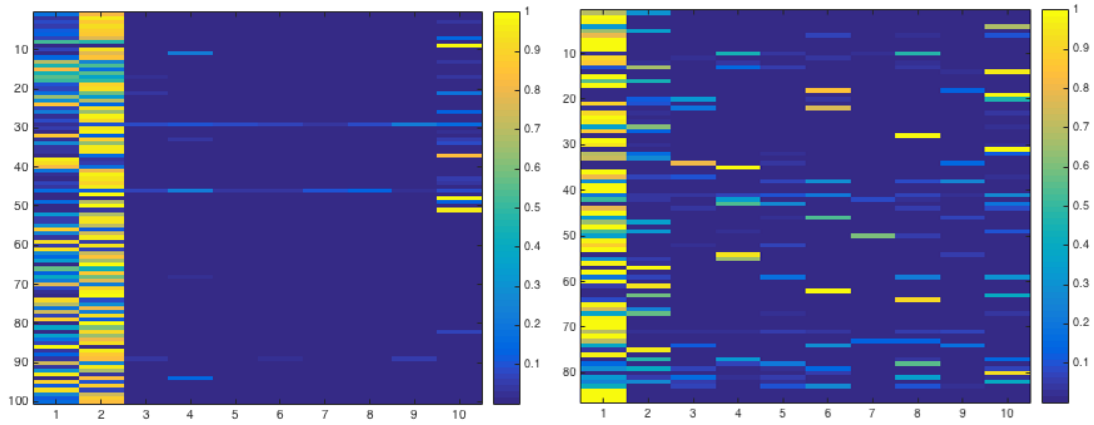
The first experiment just consisted of training the vanilla network to establish a baseline upon which we could build future improments. To do so, we trained *PointNet* using Stochastic Gradient Descent (SGD) with a learning rate of $1e^{-4}$ and a momentum of $9e^{-1}$ during 200 iterations with early stopping. This first experimental iteration achieved and accuracy rate of 77.6% in the test set. As shown in Table 2.2, the confusion matrix reveals the stability of the system, mainly confusing items that look alike such as desk and table. Because of the nature of CNNs, which heavily rely on detecting combinations of features, these kind of errors are common. As we can observe in Figure 2.15, the visual features that define a desk and a table are almost the same, making it hard to distinguish between both classes. Figure 2.16 shows the neuron activations for the output layer of the architecture, proving that *Desk* and *Table* are consistently confused in the test set. In light of those experiments it is conceivable to think that a deeper network would provide better results by being able to model the subtleties of such small differences.

Table 2.2: Confusion matrix for the first set of experiments making use of the vanilla *PointNet* architecture. Notice the high amount of confusion in the Desk-Table pair.

Desk	Table	Nstand	Bed	Toilet	Dresser	Bathtub	Sofa	Monitor	Chair
52	9	1	4	0	5	1	5	0	9
25	69	0	1	0	0	0	0	1	4
1	2	60	1	4	8	0	0	2	8
4	0	0	80	0	0	3	11	1	1
1	0	3	1	84	0	1	3	2	5
3	0	14	0	0	61	0	1	6	1
0	1	0	3	0	0	34	8	3	1
1	0	1	4	1	2	0	88	1	2
1	1	1	1	0	5	1	1	87	2
1	2	1	2	1	1	0	1	1	90

(a) **(b)**

Figure 2.15: Similarity between two objects of different classes: Table and Desk. The point cloud shown in (a) represents an object of the Table class, whilst the point cloud in (b) represents an object whose class is Desk but it is misclassified as a Table due to their resemblance.



(a) **(b)**

Figure 2.16: Neuron activations for the output layer of the architecture when classifying all the test samples for both *Desk*(b) and *Table*(a) classes. Each row represents an activation vector for a specific sample, so each column is a position of the vector: the activation to that particular class. The first column corresponds to the *Desk* class, while the second one is the *Table*. The activation shows the clear confusion between *Desk* and *Table*. Although the latter one is much less confused with other classes, many *Tables* are misclassified as *Desks* thus lowering the accuracy for this class.

For the deeper network experiment we added several layers to the vanilla *PointNet* architecture. One more convolutional layer was added since these layers are coupled to the detection of the features of the objects, so the more layers induce a more complex or expressive model. Another fully connected or inner product layer was also added. Since these layers make the classification possible, making that classifier deeper would theoretically make us able to provide better classification results. Despite all those

theoretical improvements, one must also consider that we are also introducing more parameters in the network thus making it harder to train and more prone to overfitting the train data.

This architecture was trained during 1.000 iterations to account for that increase in training difficulty. The best result was provided at 800 iterations with an accuracy of 76.7% in the test set. Those models that kept on training dropped performance to 75.9% due to overfitting. Surprisingly, increasing the depth of the network did not help rising test accuracy. Because of that, we decided to investigate our data to check if any factor was hindering the training process.

Training using an unbalanced dataset tends to harm those classes with the least number of examples and to benefit those with the most [58]. Having this in mind, and knowing that ModelNet-10 is highly unbalanced as shown in Table 2.1, the dataset was balanced by limiting the number of examples of each class to 400 samples using random undersampling. This does not fully solve the problem but makes the difference between classes with the least number of examples and those with the most least acute. The network was trained and tested again with this more balanced dataset and the best performer achieved a test accuracy of 72.9%. Again, this experiment did not result into any gain but rather made our network perform worse. The fact is that balancing the training set makes the accuracy of the classes with less examples higher, but it harms the success rate on classes with more instances as seen in Figure 2.17. The improvement on those less represented classes is not enough to make up for the impact on the most represented ones.

After analyzing the results, it can be stated that neither a deeper network nor balancing the dataset increase accuracy. In fact, the experiments of the original architecture with the unbalanced ModelNet-10 exhibited better performance with a 77.6% success rate. In addition, *PointNet* takes an average time of 24.6 milliseconds to classify an example (in comparison with *VoxNet*, which can take up to half a second for its raytracing-based implementation). These results prove the system as a fast and accurate 3D object class recognition tool.

2.4.4 Conclusion

PointNet is our first approach towards devising an efficient and accurate CNN for 3D object class recognition. It was inspired by other contemporary works such *VoxNet* and

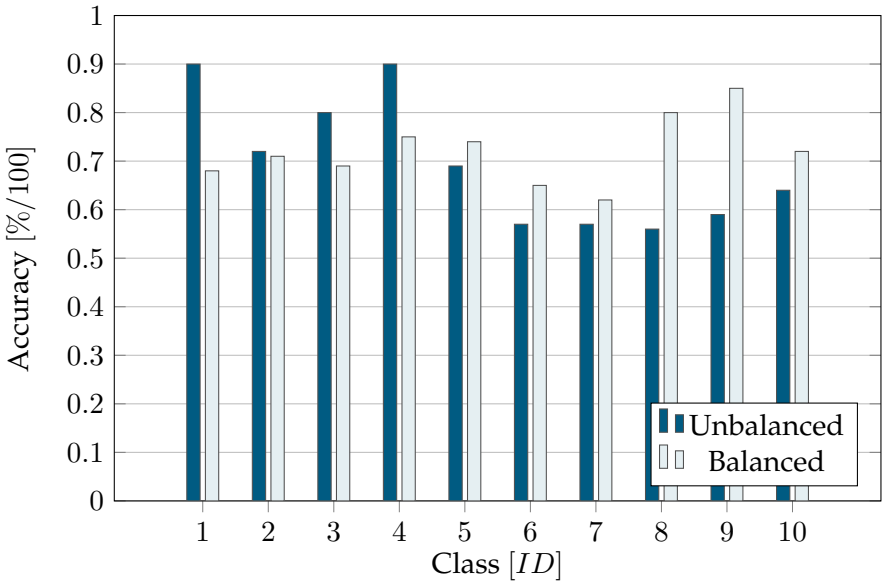


Figure 2.17: Comparison of accuracy per class using an unbalanced dataset and a balanced one with a maximum of 400 models per class via random undersampling. Accuracy is harmed in the classes in which models are removed but gained otherwise.

3D *ShapeNets*, in the sense that it also handles tridimensional data for object recognition. Its contributions lies in the fact that the architecture itself and the representation too were simplified and the whole pipeline was implemented in Caffe thus providing a faster method than the state of art ones yet obtaining a high accuracy rate in the ModelNet-10 dataset.

2.5 Noise and Occlusion

Although *PointNet* was a step forward at the time it was released, there were still a set of open questions regarding its effectiveness. Such questions are strongly related to common problems that arise when recognizing objects in real-world scenes: how sensitive is the network to the noise introduced by the sensor? And most importantly, how robust is the network against clutter and occlusions? In this second iteration of our object recognition system we aim to throw some light into such unknowns. To that end, we carry out a study in which we test various data representations and an improved network iterated over *PointNet*. The section is organized as follows. First off, we introduce our proposals for robust data representations in Section 2.5.1. In the second

place, we discuss the improved network architecture in Section 2.5.2. After that, the main body of this iteration is presented in Section 2.5.3: an extensive experimentation to determine the effect of noise and occlusion. In the end, in Section 2.5.4 we discuss the insights gained thanks to the experimentation and we also set the scene for the last iteration of the object recognition system.

2.5.1 Data Representation

As we already stated previously, a volumetric representation to be fed to a CNN must encode the 3D shape of an object as a 3D tensor of binary or real values. This is due to the fact that raw 3D data is sparse, i.e., a 3D shape is only defined on its surface, and CNNs are not engineered for this kind of data. However simple this statement might seem, there are many nuances and subtleties in the way such tensors are created and how their values are computed. Those detials may have an important impact on performance in unexpected ways.

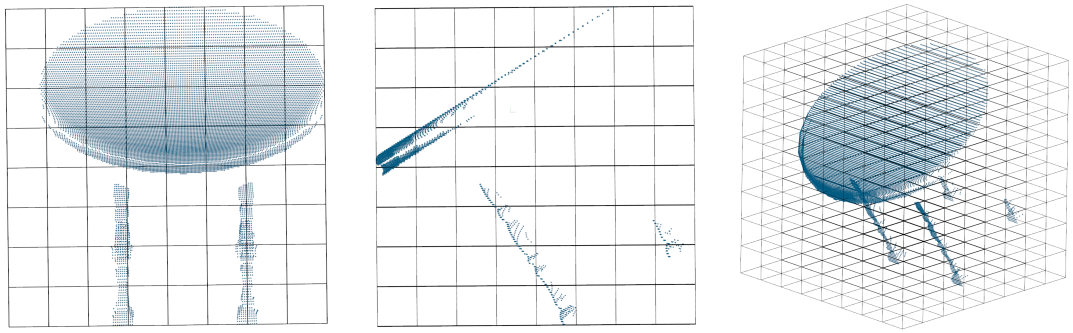
In this regard, our proposal for the study is twofold. First, we implemented two different ways of generating the structure of the tensor – position, grid size, and leaf size – using a fixed grid and an adaptive one. Second, we developed two possible occupancy measures for the volumetric elements of the tensor.

Tensor Generation

Providing that the input to our network consists of point clouds generated from the information provided by RGB-D sensors, we need to generate a discretized representation of the unbounded 3D data to feed the network. Each cloud will be represented as a 3D tensor. For that purpose, we need to spawn a grid to subdivide the space occupied by the point clouds. Two types are proposed: one with fixed leaf and grid sizes, and another one which will adapt those sizes to fit the data.

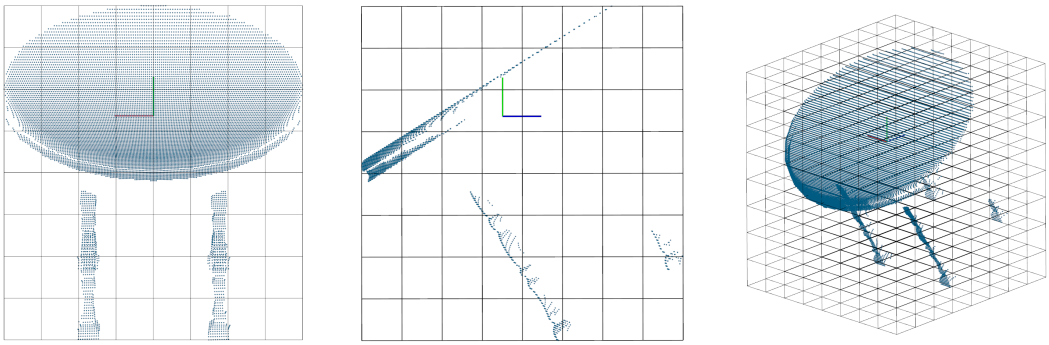
Fixed This kind of grid sets its origin at the minimum x, y, and z values of the point cloud. Then the grid is spawned, with fixed and predefined sizes for both grid and voxels. After that, the cloud is scaled up or down to fit the grid. The scale factor is computed with respect to the dimension of maximum difference between the cloud

and the grid. The cloud is scaled with that factor in all axes to maintain the original ratios. As a result, a cubic grid is generated as shown in Figure 2.18.



(a) Front **(b)** Side **(c)** Perspective

Figure 2.18: A fixed occupancy grid ($8 \times 8 \times 8$ voxels) with 40 units leaf size and 320 units grid size in all dimensions. The grid origin is placed at the minimum x, y, and z values of the point cloud. Front (a), side (b), and perspective (c) views of the grid over a partial view of a segmented table object are shown.



(a) Front **(b)** Side **(c)** Perspective

Figure 2.19: An adaptive occupancy grid ($8 \times 8 \times 8$ voxels) with adapted leaf and grid sizes in all dimensions to fit the data. The grid origin is placed at the minimum x, y, and z values of the point cloud. Front (a), side (b), and perspective (c) views of the grid over a partial view of a segmented table object are shown. Notice that the point clouds for the three views are exactly the same for this figure and Figure 2.18, but the grids do change. There is a noticeable difference in the front view. In Figure 2.18, using fixed grids, all voxels are cubic and the point cloud does not fit the grid completely (leftmost column in Figure 2.18a), whilst in this figure, with adaptive grids, the grid is fitted to the cloud.

Adaptive The adaptive grid also sets its origin at the minimum x, y, and z values of the point cloud. Next, the grid size is adapted to the cloud dimensions. The leaf size

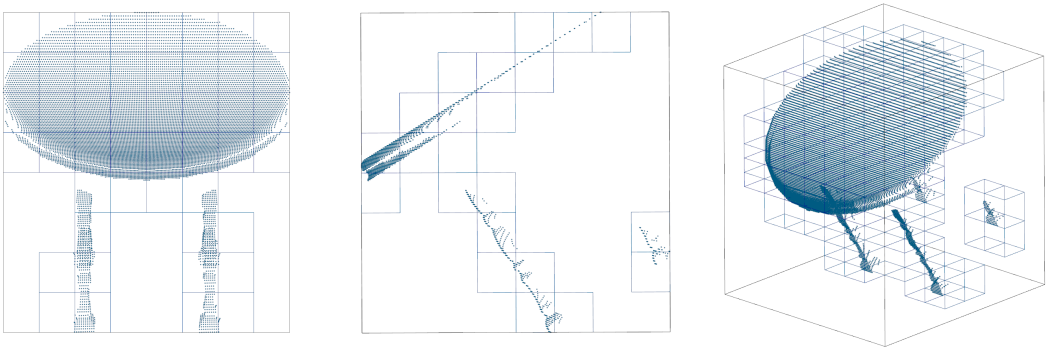
is also computed in function of the grid size. Knowing both parameters, the grid is spawned, fitting the point cloud data. As a result, a non-cubic grid is generated. As shown in Figure 2.19, all voxels have the same size, but they are not necessarily cubic.

It is important to remark that, in both cases (fixed and adaptive), the number of voxels in the grid is fixed. Figures 2.18 and 2.19 show examples for both types using $8 \times 8 \times 8$ voxels for the sake of a better visualization.

It is also important to notice that each representation serves a purpose. The fixed grid will not always fit the data perfectly so it might end up having sparse zones with no information at all (as seen in Figure 2.18a on the first column). However, it can be used right away for sliding box detection. On the contrary, the adaptive grid fits the data to achieve a better representation. Nonetheless, it relies on a proper segmentation of the object to spawn the grid.

Occupancy Computation

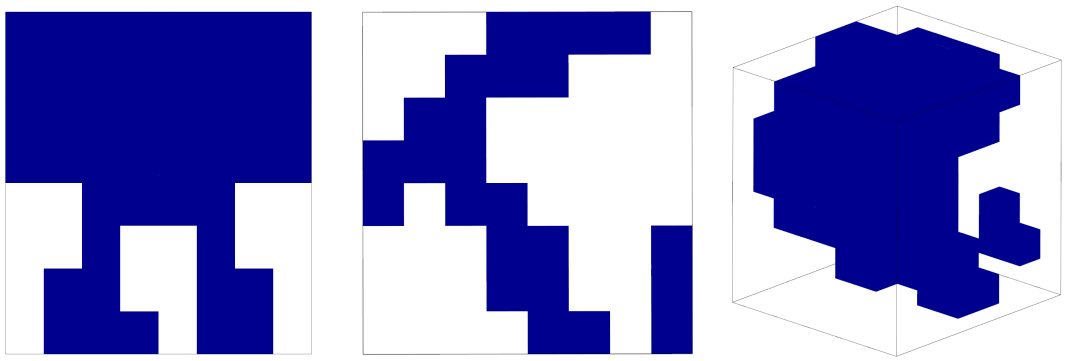
After spawning the grid to generate a discrete space, we need to determine the values for each cell or voxel of the 3D tensor. In order to do that, we must encode the geometric information of the point cloud into each occupied cell (see Figure 2.20). In other words, we have to summarize as a single value, the information of all points which lie inside a certain voxel. One way to do that is using occupancy measures. For that purpose, we propose two different alternatives: binary occupancy, normalized density.



(a) Front **(b)** Side **(c)** Perspective

Figure 2.20: Occupied voxels in an adaptive $8 \times 8 \times 8$ grid generated over a partial view point cloud. Those voxels with points inside are shown in a wireframe representation. Empty voxels are omitted. Occupied voxels must be filled with values which represent the contained shape.

Binary The binary tensor is the simplest representation that can be conceived to encode the shape. Voxels will hold binary values, they will be considered occupied if at least a point lies inside, and empty otherwise. Figure 2.21 shows an example of this tensor.



(a) Front **(b)** Side **(c)** Perspective

Figure 2.21: Binary tensor computed over a point cloud of a partial view of an object (shown in Figure 2.20). Occupied voxels are shown in blue, empty voxels are omitted for the sake of simplicity.

Normalized Density Binary representations are simple and require low computational power. However, complex shapes may get oversimplified so useful shape information gets lost. This representation can be improved by taking into account more shape information. A possible alternative consists of computing the point density inside each voxel, i.e., counting the number of points that fall within each cell.

It is important to notice that point density directly depends on the cloud resolution which in turn depends on many factors involving the camera and the scene, e.g., it is common for RGB-D to generate denser shapes in closer surfaces. To alleviate this problem, we can normalize the density inside each voxel dividing each value by the maximum density over the whole tensor. An example of normalized density tensor is shown in Figure 2.22.

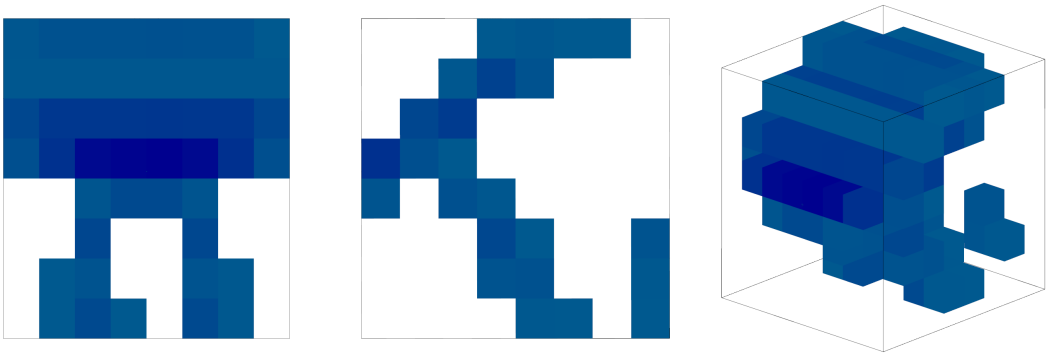
2.5.2 Network Architecture

In this section, we will describe the main layers that compose the CNN that will be used for the study. Figure 2.23 shows a diagram of the chosen architecture. It is highly

inspired by *Voxnet* [49] and *PointNet* [17]. The network was implemented using Caffe. It features 2D convolutions and takes full 3D object model point clouds as input.

The input layer is a custom data layer implemented in Caffe which takes object point clouds as inputs and generates the corresponding discrete volumetric representation as discussed in the previous section.

Next, we can find a *convolution layer* or $C(m, n, d)$. This layer applies m filters of size $n \times n$ and a stride of $d \times d$ voxels. In our case, this first convolution layer learns 48



(a) Front **(b)** Side **(c)** Perspective

Figure 2.22: Normalized density tensor over a point cloud of a partial view of an object (shown in Figure 2.20). Denser voxels are darker and sparse ones are shown in light blue. Empty voxels were removed for visualization purposes.

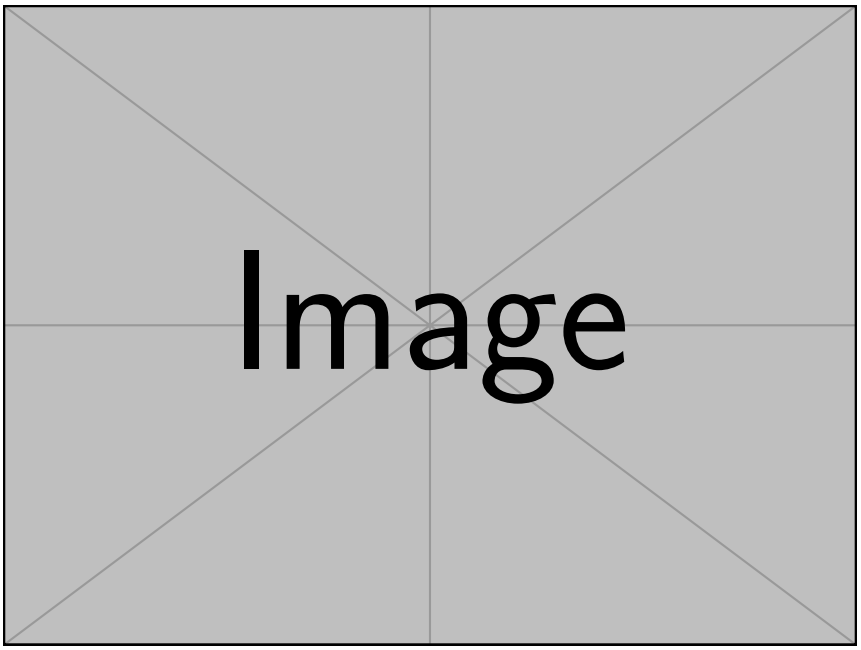


Figure 2.23: CVIU's architecture. [MISSINGDETAILS]

3×3 filters using a stride of 1×1 voxels. This convolution layer is followed by a ReLU activation to introduce non-linearities to the model.

After that, another convolution layer is found. In this case, it will learn 128 5×5 filters with a stride of 1×1 voxels again. This layer is also followed by a ReLU activation one.

A *pooling layer* or $P(n, d)$ takes place after those blocks. It performs a max-pooling process to summarize the input data, taking the maximum value of a fixed local spatial region of $n \times n$ which is slided across the input volume using a stride of $d \times d$ voxels. In this case, a pooling region of 2×2 voxels with the same stride was chosen.

At last, we can find an *inner product layer* or $IP(n)$. It is just a fully connected layer, a traditional neural network architecture which consists of n neurons (1024 in this case). It is followed by a ReLU activation and a *dropout layer* **Srivastava2014** or $DP(r)$. The function of the dropout layer is to avoid overfitting, randomly dropping connections with a probability r (0.5 in our case). In the end, another fully connected layer represents the output of the network, with as many output neurons as classes has our classification problem. Since our dataset has 10 classes (see Section **??**) this layer has 10 neurons.

We use the term 2.5D to refer to this network due to the fact that it processes 3D data using 2D convolutions. This means that, in the end, its convolutions do not fully take into account the depth spatial dimension of the input as if we were using pure 3D convolution filters. It is intuitive to think that a 3D CNN would yield better results due to that extra spatial dimension. However, a 3D CNN has some disadvantages that made us consider using a 2.5D CNN instead for the experimentation: (1) higher computational cost, (2) memory footprint is also much higher, (3) more parameters thus harder training. For those reasons, the main body of the experiments were carried out using the 2.5D approach.

2.5.3 Experiments

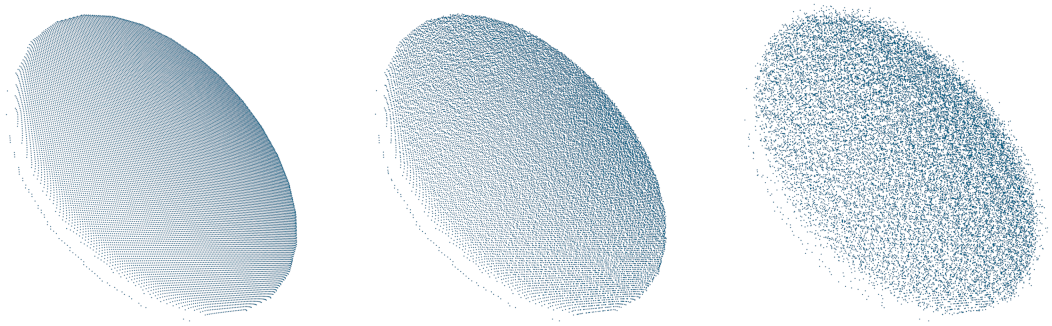
In order to assess the performance of the proposed model-based CNN we carried out an extensive experimentation to determine the accuracy of the model and its robustness against occlusions and noise – situations that often occur in real-world scenes. For that

purpose we started using the normalized density grids since they offer a good balance between efficiency and representation. We also investigated the effect of both fixed and adaptive grids using different sizes. Further experimentation was performed to compare the normalized density grids with the binary ones. We also carried out a brief experiment using a 3D CNN to compare its performance with the 2.5D counterpart.

Data Generation

Data was generated in a similar fashion as we did for PointNet. The meshes from the ModelNet-10 dataset were converted to point clouds following the procedure described in Section 2.3. However, this time we applied a set of transformations to simulate noise and occlusions in our data.

Noise Simulation The partial views generated using the previously described process are not a good simulation of the result that we would obtain by using a low-cost RGB-D sensor. Those systems are noisy, so the point clouds produced by them are not a perfect representation of the real-world objects.



(a) $\sigma = 0$ **(b)** $\sigma = 0.1$ **(c)** $\sigma = 1$

Figure 2.24: Different levels of noise ($\sigma = 0$ (a), $\sigma = 0.1$ (b), and $\sigma = 1$ (c)) applied to the z-axis of every point of a table partial view.

In order to properly simulate the behavior of a sensor, a model is needed. In our case, we are dealing with low-cost RGB-D sensors such as Microsoft Kinect and Primesense Carmine. A complete noise model for those sensors, specifically for the Kinect device, must take into account occlusion boundaries due to distance between the Infrarred (IR) projector and the IR camera, 8-bit quantization, 9×9 pixel correlation window smoothing, and z-axis or depth Gaussian noise [59].

We will make use of a simplification of this model, only taking into account the Gaussian noise since it is the most significant one for the generated partial views. In this regard, the synthetic views are augmented by adding Gaussian noise to the z dimension of the point clouds with mean $\mu = 0$ and different values for the standard deviation σ to quantify the noise magnitude. Figure 2.24 shows the effect of this noise over a synthetic partial view of one object of the dataset.

Occlusion Simulation In addition to modelling the sensor to improve our synthetic data, it is important to also take the environment into account. In a real-world scenario, objects are not usually perfectly isolated and easily segmented; in fact, it is common for them to be occluded by other elements of the scene.

The occlusion simulation process consists of picking a random point of the cloud with a uniform probability distribution. Then, a number of closest neighbors to that point are picked. At last, both the neighbors and the point are considered occluded surface and removed from the point cloud. The number of neighbors to pick depends on the amount of occlusion ψ we want to simulate. For instance, for an occlusion $\psi = 25\%$ we will remove neighbors until the rest of the cloud contains a 75% of the original amount of points, i.e., we will remove a 25% of the original cloud. Figure **??** shows the effect of the random occlusion process with different occlusion factors ψ over a synthetic partial view of a table object of the dataset.

It is important to notice the randomness of the occlusion process. This means that even with a high ψ it is possible not to remove any important surface information and vice versa. In other words, it is possible for some objects to remove a 50% of their points and still be recognizable because the removed region was not significant at all, e.g., a completely flat surface. However it is possible to render an object unrecognizable by removing a small portion of its points if the randomly picked surface is significant for its geometry. This remark is specially important when testing the robustness of the system. In order to guarantee that an appropriate measure of the robustness against missing information is obtained, a significant amount of testing sets must be generated and their results averaged so that it is highly probable to test against objects which have been occluded all over their surface across the whole testing set.

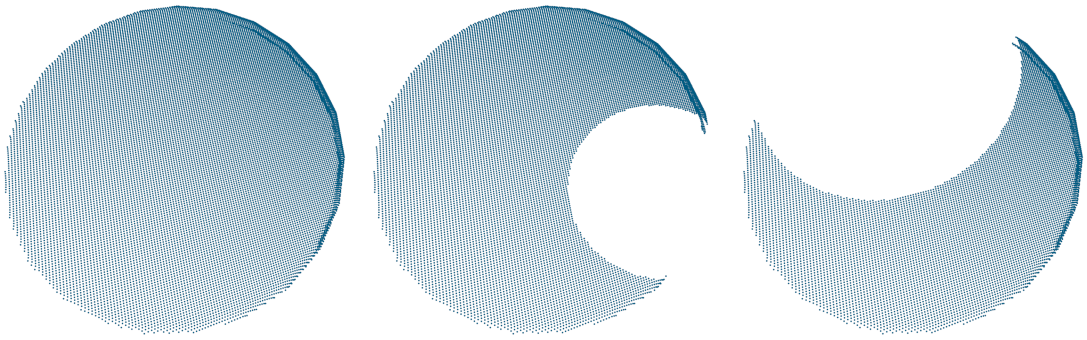
Implementation and Setup

Results were obtained using the following test setup: Intel Core i7-5820K with 32 GiB of Kingston HyperX 2666MHz and CL13 DDR4 RAM on an Asus X99-A motherboard (Intel X99 chipset). Additionally, the system included an NVIDIA Tesla K40c GPU used for training and inference. The framework of choice was Caffe RC2 running on Ubuntu 14.04.02. It was compiled using CMake 2.8.7, g++ 4.8.2, CUDA 7.5, and cuDNN v3.

Results and Discussion

The networks were trained for a maximum of 5000 iterations – weights were snapshotted every 100 iterations so in the end we selected the best sets of them as if we were early stopping – using Adadelta as optimizer with $\delta = 1 \cdot 10^{-8}$. The regularization term or weight decay in Caffe was set to $5 \cdot 10^{-3}$. A batch size of 32 training samples was chosen.

After describing the experimentation setup, the dataset that was used to train and test the networks, and the ways of simulating noise and occlusion for the test sets, we will present and discuss the results of the experiments. Firstly, the normalized density tensor results – using the 2.5D CNN – will be presented. After that, we will proceed with the binary tensor ones. Furthermore, we will report the experiments which produced the best results with a pure 3D CNN with fully 3D convolutions. At last, we will perform a comparison with the state of the art.



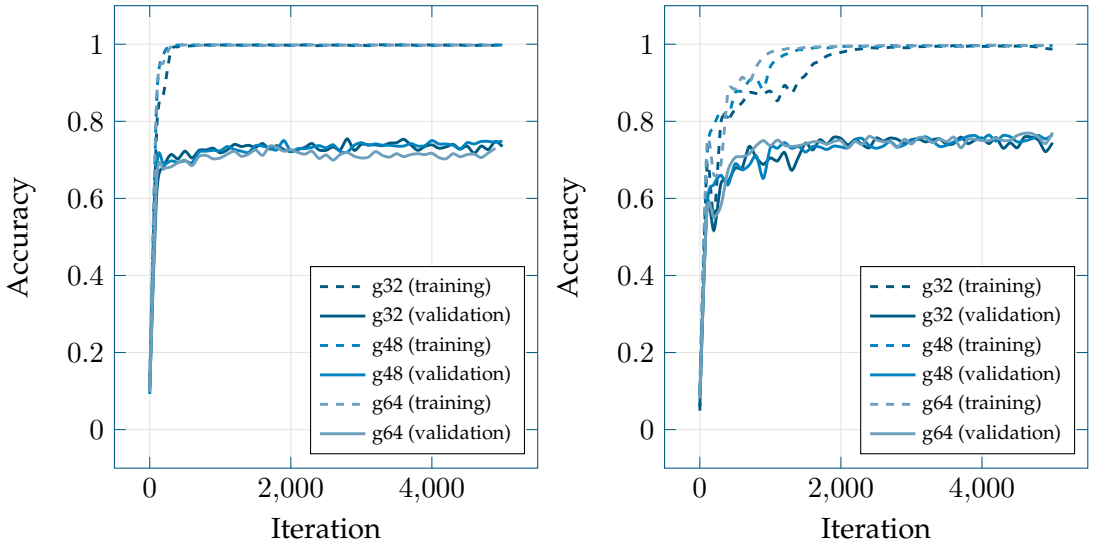
(a) $\psi = 0\%$ **(b)** $\psi = 25\%$ **(c)** $\psi = 50\%$

Figure 2.25: Different levels of occlusion ($\psi = 0\%$ (a), $\psi = 25\%$ (b), and $\psi = 50\%$ (c)) applied randomly to a table partial view.

Density Tensor Figure 2.26 shows the accuracy results of the network for both grid types and increasing sizes. The peak accuracies for the fixed grids are ≈ 0.75, ≈ 0.76, and ≈ 0.73 for sizes 32, 48, and 64 respectively. In the case of the adaptive one, the peak accuracies are ≈ 0.77, ≈ 0.78, and ≈ 0.79 for the sizes 32, 48, and 64 respectively.

Taking those facts into account, we can extract two conclusions. First, the adaptive grid is able to achieve a slightly better peak accuracy in all cases; however, the fixed grid takes less iterations to reach accuracy values close to the peak in all cases. Second, there is no significant difference in using a bigger grid size of 64 voxels instead of a smaller one of 32.

The most important fact that can be observed in the aforementioned figures is that there is a considerable gap between training and validation accuracy in all situations. As we can observe, all networks reach maximum accuracy for the training set whilst the validation one hits a glass ceiling at approximately 0.80. We hypothesize that the network suffers overfitting even when we thoroughly applied measures to avoid that. The most probable cause for that problem is the reduced number of training examples. In the case of ModelNet10 the training set consists of only 3991 models. Considering the complexity of the CNN, it is reasonable to think that the lack of a richer training set



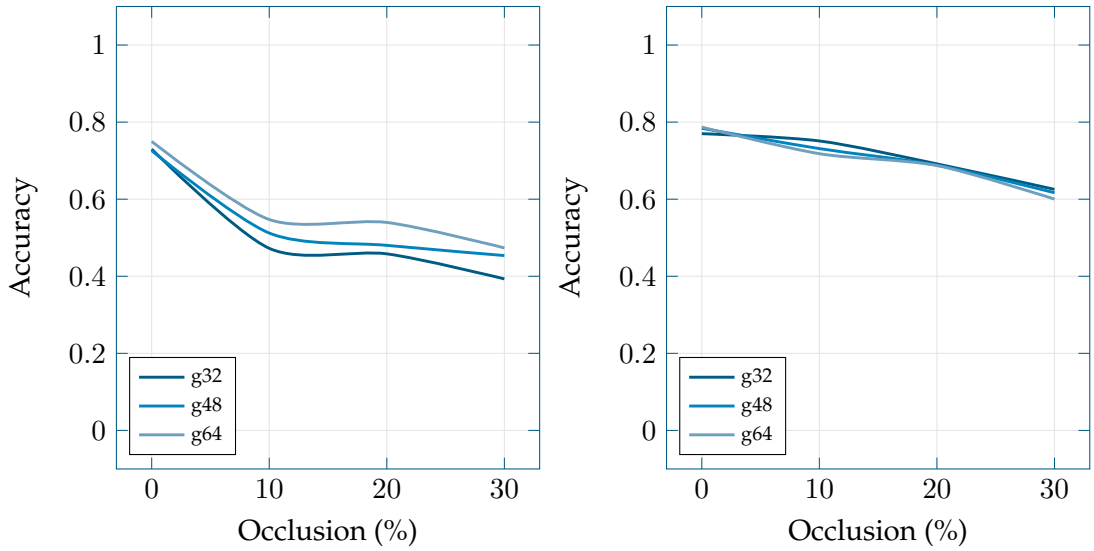
(a) Fixed Grid **(b)** Adaptive Grid

Figure 2.26: Evolution of training and validation accuracy of the model-based CNN using both fixed (a) and adaptive (b) normalized density grids. Different grid sizes ($32, 48,$ and 64) were tested.

is causing overfitting.

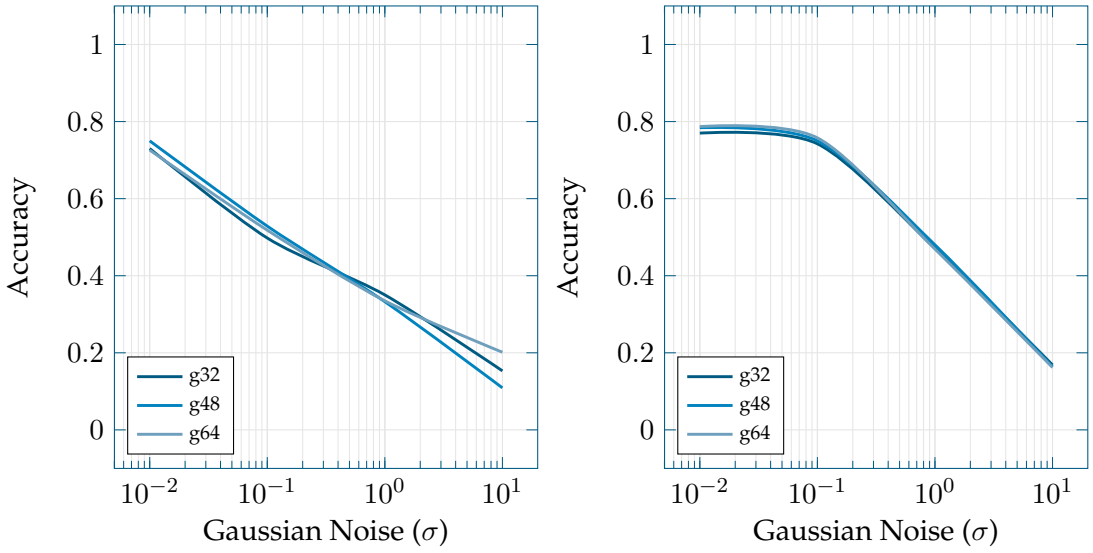
Concerning the robustness against occlusion, we took the best networks after training and tested them using the same validation sets as before but introducing occlusions in them (up to a 30%). Figure 2.27 shows the accuracy of both grid types with different sizes as the amount of occlusion in the validation model increases. As we can observe, occlusion has a significant and negative impact on the fixed grid – bigger grid sizes are less affected – going down from ≈ 0.75 accuracy to $0.40 - 0.50$ approximately in the worst and best case respectively when a 30% of the model is occluded. On the contrary, the adaptive grid does not suffer that much – it goes down from ≈ 0.78 to ≈ 0.60 in the worst case – and there is no significant difference between grid sizes. In conclusion, the adaptive grid is considerably more robust to occlusion than the fixed one.

Regarding the resilience to noise, we also tested the best networks obtained from the aforementioned training process using validation sets with different levels of noise (ranging from $\sigma = 1 \cdot 10^{-2}$ to $\sigma = 1 \cdot 10^{1}$). Figure 2.28b shows the results of those experiments. It can be observed that adding noise has a significant impact on the fixed grid, even small quantities, reducing the accuracy from ≈ 0.75 to ≈ 0.60, ≈ 0.4, and



(a) Fixed Grid **(b)** Adaptive Grid

Figure 2.27: Evolution of validation accuracy of the model-based CNN using both fixed (a) and adaptive (b) normalized density grids as the amount of occlusion in the validation models increases from 0% to 30%. Three grid sizes were tested ($32, 48,$ and 64).



(a) Fixed Grid **(b)** Adaptive Grid

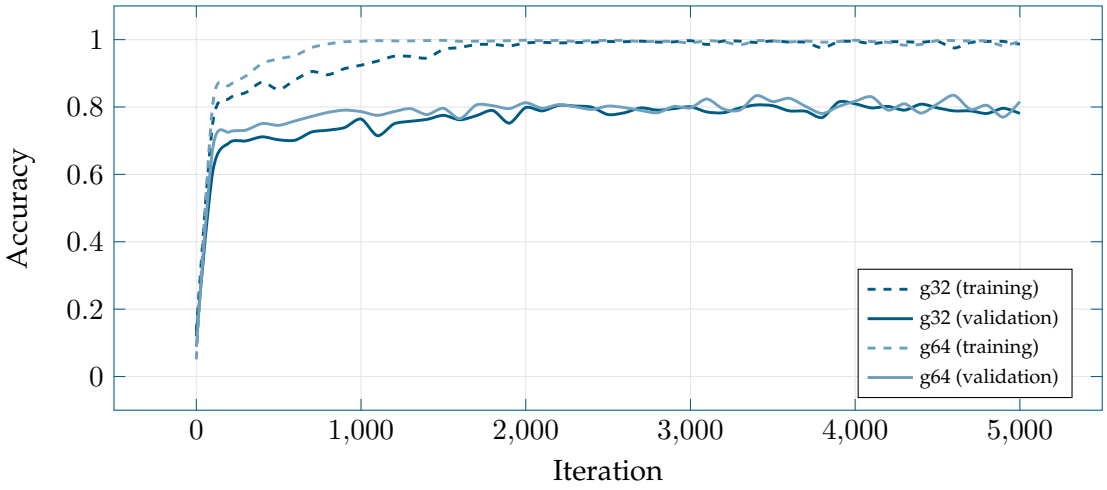
Figure 2.28: Evolution of validation accuracy of the model-based CNN using both fixed (a) and adaptive (b) normalized density grids as the standard deviation of the Gaussian noise introduced in the z-axis of the views increases from 0.001 to 10. The common grid sizes were tested ($32, 48,$ and 64).

≈ 0.2 for $\sigma = 1 \cdot 10^{-1}, \sigma = 1 \cdot 10^{0}$, and $\sigma = 1 \cdot 10^{1}$ respectively. On the other hand, the adaptive one shows remarkable robustness against low levels of noise (up to $\sigma = 1 \cdot 10^{-1}$), barely diminishing its accuracy.

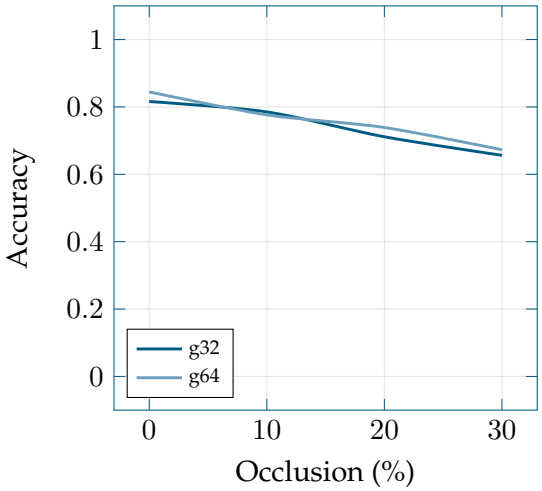
In the end, both grids suffer huge penalties in accuracy when noise levels higher than $\sigma = 1 \cdot 10^{-}1$ are introduced, being the adaptive one less affected. The grid size has little to no effect in both cases, only in the fixed grid bigger sizes are slightly more robust when intermediate to high levels of noise are introduced. In conclusion, the adaptive grid is significantly more resilient to low levels of noise, and slightly outperforms the fixed one when dealing with intermediate to high ones.

Binary Tensor After testing the performance of the normalized density grid, we also trained and assessed the accuracy of the binary one in the same scenarios. This test intended to show whether there is any gain in using representations which include more information about the shape – at a small penalty to execution time.

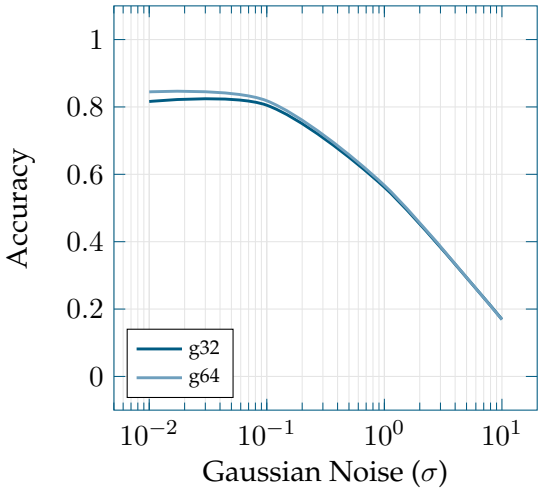
For this experimentation we picked the best performer in the previous sections: the adaptive grid. We also discarded the intermediate size (48 voxels) since there was no



(a) Training



(b) Occlusion



(c) Noise

Figure 2.29: Evolution of training and validation accuracy of the model-based CNN using adaptive binary grids (a). Evolution of validation accuracy for the best network weights after training as the amount of occlusion in the validation set increases (b) and different levels of noise are introduced (c).

significant difference between it and the others. Figure **??** shows the accuracy results of the network trained using binary grids. As we can observe, there is no significant difference between grid sizes neither. However, using this representation we achieved a peak accuracy of approximately 0.85, using 64 voxels grids, which is better to some extent than the normalized density one shown in Figure 2.26.

Occlusion and noise tolerance (shown in Figures 2.29b and 2.29c respectively) is

mostly similar to the robustness shown by the normalized density adaptive grid (see Figures 2.27b and 2.28b) except from a small offset caused by the higher accuracy of the binary grid network.

In conclusion, the less-is-better effect applies in this situation and turns out that the simplification introduced by the binary representation helps the network during the learning process. It is pending to check if this statement is still valid if the validation accuracy is not bounded by network overfitting.

3D CNN At last, we tested the best configuration – binary adaptive grids – with a 3D CNN architecture with pure 3D convolutions. We kept the same architecture we introduced in Section **??**, but extended its convolution and pooling layers to three dimensions. We then trained the network using adaptive binary grids as the volumetric representation of choice and monitored validation and training errors. Due to memory limitations on the GPU we could only experiment with grids of $32 \times 32 \times 32$ voxels.

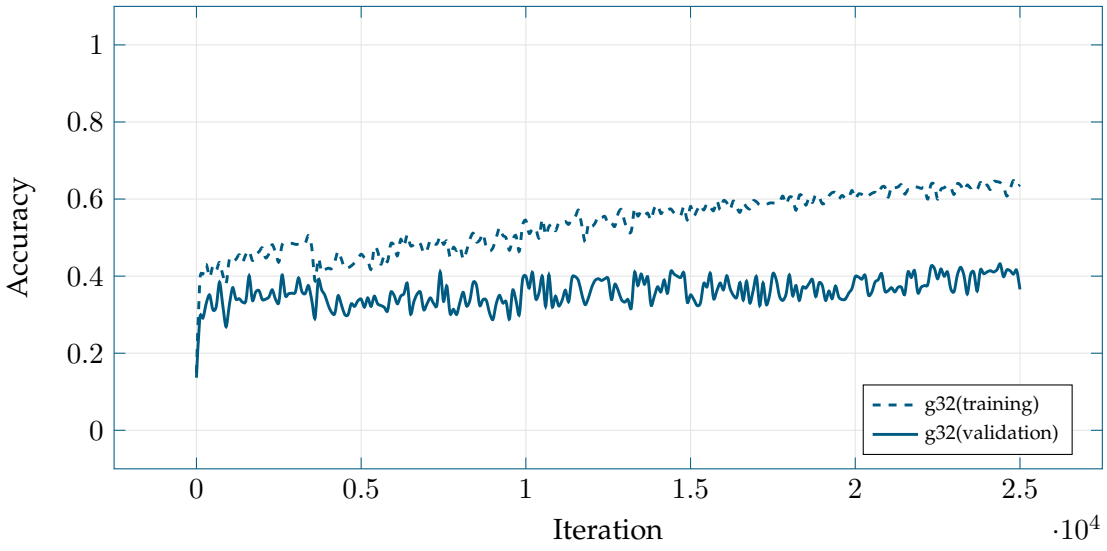


Figure 2.30: Evolution of training and validation accuracy of the 3D CNN using adaptive binary grids with size $32 \times 32 \times 32$.

Figure 2.30 shows the results of this experiment. As we can observe, we trained the network for five times more iterations than before and even then we couldn't achieve a proper convergence. The training set accuracy kept increasing slowly up to approximately 0.65 whilst the validation one got stuck around 0.40 for the whole experiment.

In conclusion, porting the 2.5D network directly to 3D just by extending its convolution and pooling layers to slide along the depth axis did not produce good results using the same dataset and setup that produced a significantly good outcome with the 2.5D architecture. We hypothesize various causes for this problem.

On the one hand, the data representation might not be adequate for such fine-grained convolutions. It is presumable that bigger grids, e.g., $64 \times 64 \times 64$, would yield better results. However, given the size of the model, they could not be tested in the available GPU.

On the other hand, the complexity of the network increased considerably after including that extra dimension in convolution and pooling layers. This means that the number of parameters of the network gets increased significantly, making it harder to train with so few samples due to overfitting. This hypothesis is backed up by the fact that training accuracy kept increasing slowly while validation one got stuck. This would eventually lead to a perfect fit on the training set but low accuracy on the validation split.

Discussion To sum up, we determined that the adaptive grid slightly outperforms the fixed one in normal conditions. The same happens with the grid size, obtaining marginally better results with bigger sizes. However, when it comes down to noise and occlusion robustness, the adaptive grid exceeds the accuracy of the fixed grid by a large margin for low levels of occlusion and noise, whilst for intermediate and high levels the impact on both grids is somewhat similar. In other words, the adaptive grid is better than the fixed one and it is preferable to use a bigger grid size if the performance impact can be afforded.

It is important to remark that the binary occupancy measure performed better than the normalized density one, both using adaptive grids, while maintaining similar resilience against noise and occlusions. The best network trained with normalized density grids reached a peak accuracy of approximately 0.79 while the best binary one achieved approximately a 0.85 accuracy on the validation set.

Another remarkable fact was that all networks exhibited a considerable amount of overfitting, i.e., training accuracy was almost perfect whilst validation was far away

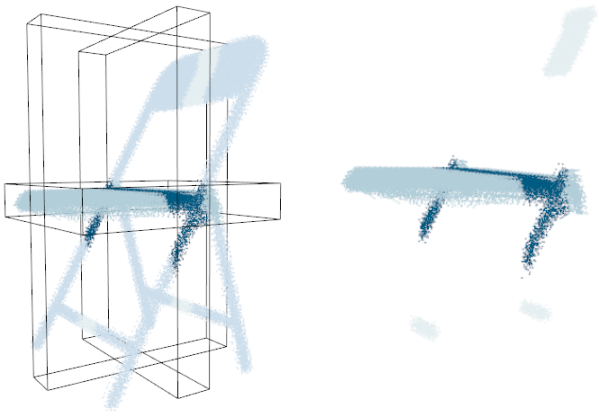
from it by a considerable margin. We hypothesize that this was due to the fact that the dataset has few training examples considering the complexity of the network. Besides, we also inspected the confusion matrix shown in Table **??** to gain insight about the behavior of our network. As we can observe, there are many misclassified samples of classes that are similar. If we take a closer look at some of the misclassified samples (see Figures **??**, **??**, and **??**) it is reasonable to think that the network is not able to classify them properly because they are extremely similar. In this regard, the dataset must be augmented introducing noise, translations, rotations, and variations of the models to avoid overfitting and learn better those models that can be easily misclassified.

2.5.4 Conclusion

2.6 LonchaNet

In this work, we present a novel approach that uses multiple 2D views from 3D models for 3D object recognition. The proposed 2D renderings are based on cross sections of 3D models. The proposed method outperforms most existing approaches that participated in the ModelNet challenge. Currently, we are in the second position of the leaderboard, obtaining 94.37% classification accuracy. Our proposal is focused on learning a discriminative representation that is able to distinguish among most categories of the ModelNet dataset. We also contribute with a new architecture that uses the existing GoogLeNet network **Szegedy2015**. We use three independent GoogLeNet networks for learning specific features of each cross-section or slice from the 3D model.

As stated before, we propose a 3D object recognition method using deep learning. First, for every example in the dataset, we took three sections from an object, one for each 3D axis, and project the 3D points to a plane, so we obtain three images from every example. Each of these three images that shape a single example are then fed to a deep Convolutional Neural Network. Our novel deep architecture features three GoogLeNets, one for each image, joined in a layer prior to the classification layer. This provides us with great expressiveness and a high success rate.



(a) Sliced example **(b)** Points of every slice

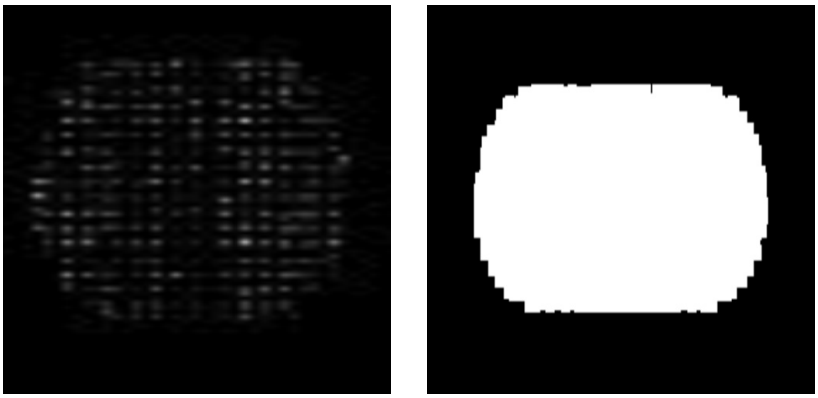
(c) Resultant projections of the three slices

Figure 2.31: Extracting the slices from a point cloud example [MISSINGDETAILS]

2.6.1 Data Representation

LonchaNet takes point clouds as input, but the neural architecture itself uses three images corresponding to three slices, so first we need to extract the slices from the 3D point cloud. To that end, we load the point clouds and calculate the central point for each axis. Then we make a slice across the XY, XZ, and YZ planes with a thickness of 5% of the model size. Those points that fall inside these sections are isolated and projected in their planes to generate an image of 500×500 pixels. These images are binary maps in which the background is black and the projected points are white. This process is shown in Figure 2.31.

Due to the inconsistent point density produced by the sampling process, there are



(a) Model slice with scattered pixels

(b) Model slice with dilated pixels

Figure 2.32: A comparison of a slice before and after the dilation process. The dilated image provides a more accurate representation the object.

some slices in which the points are very scattered, so the projection does not represent the object faithfully. To deal with this problem, a post-process is carried out for each projection in which we apply dilations of 10 pixels using a square as structuring element. This post-processing step fills the gap between the scattered points and produces a more suitable representation of the object, as shown in Figure 2.32. This process is performed for each example present in the dataset, so for each point cloud there are three corresponding images, one per slice.

This sliced representation of the object allows us to train and test a 3D recognition system in a 2D fashion, that provides the high success rate and the fast execution time that characterizes deep image recognition networks yet preserving and taking advantage of the 3D information implicitly materialized in the slicing method.

2.6.2 Network Architecture

The main architecture of LonchaNet is composed of three isolated GoogLeNet that are joined at the bottom in a Concatenation Layer prior a Fully Connected Layer which is the final classifier, as shown in Figure **??**.

As stated before, GoogLeNet is the state-of-the-art deep Convolutional Neural Network for image recognition tasks, providing the highest accuracy in several challenges, so we chose it over the other topologies.

In this architecture, all convolutions, including those inside the inception modules, use ReLU activation. The size of the receptive field in this network is 224×224 taking RGB channels with mean subtraction, although in LonchaNet ensemble we use binary maps with no mean normalization. GoogLeNet network consists on 22 layers deep when considering only layers with parameters (or 27 layers if we also consider pooling ones). The overall number of layers (independent building blocks) used for the construction of the network is about 100. However, this number depends on the machine learning infrastructure system used. The use of average pooling step prior the classifier is based on **Lin2013**, although this implementation differs in the use of an extra linear layer. This enables adapting and fine-tuning the network for other datasets.

LonchaNet features three independent GoogLeNet (we will reference them as "branches") learning features that define an object for each slice, so we force each branch to specialize the filters in particular features of every slice. Finally, the responses of each branch are concatenated in a single output that is fed to a fully connected layer that acts as a classifier.

2.6.3 Experiments

Data Generation

Methodology and Setup

All timings and results were obtained by conducting the experiments in the following test setup: Intel Core i7-5820K with 32 GiB of Kingston HyperX 2666 MHz and CL13 DDR4 RAM on an Asus X99-A motherboard (Intel X99 chipset). Secondary storage was provided by a Samsung 850 EVO SSD. Additionally, the system included two NVIDIA Tesla K40c GPUs used for training and inference.

The framework of choice was Caffe RC2 running on Ubuntu 14.04.02. It was compiled using CMake 2.8.7, g++ 4.8.2, CUDA 7.5, and cuDNN v3.

Results and Discussion

We trained from scratch and tested LonchaNet with the ModelNet-10 dataset, as described earlier in the subsection **??**. It is remarkable that no transfer learning or fine

tuning over a pre-trained set of parameters was used. It is remarkable that the training and test splits are defined by the dataset itself and also it is worth saying that the number of examples per class are not balanced, as seen in Table **??**. This fact could harm the accuracy of the system, biasing the learning and the classification to the classes with more number of examples, as stated by **He2009**.

Regarding the parameters that affects to the learning process, we trained the architecture with a base learning rate of 0.00001, multiplying the current learning rate by 0.75 every 10000 iterations. To compute the weights update we use the ADAM **KingmaB14** solver with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The training process executed for 20000 iterations, but the best weights set was produced on the iteration $\#18300$ which threw a test accuracy of 94.3709%, the second best score in the leaderboard of the ModelNet10 challenge.

It is also worth noticing the fast computations of our architecture. One training iteration with a batch size of 30 examples takes an average of 2.25 seconds. Also, classify a new examples only takes 0.0896 seconds.

Figure 2.33 shows the success rate per class. It can be seen that the classes with the less success rates are the very same classes with the less number of examples, namely

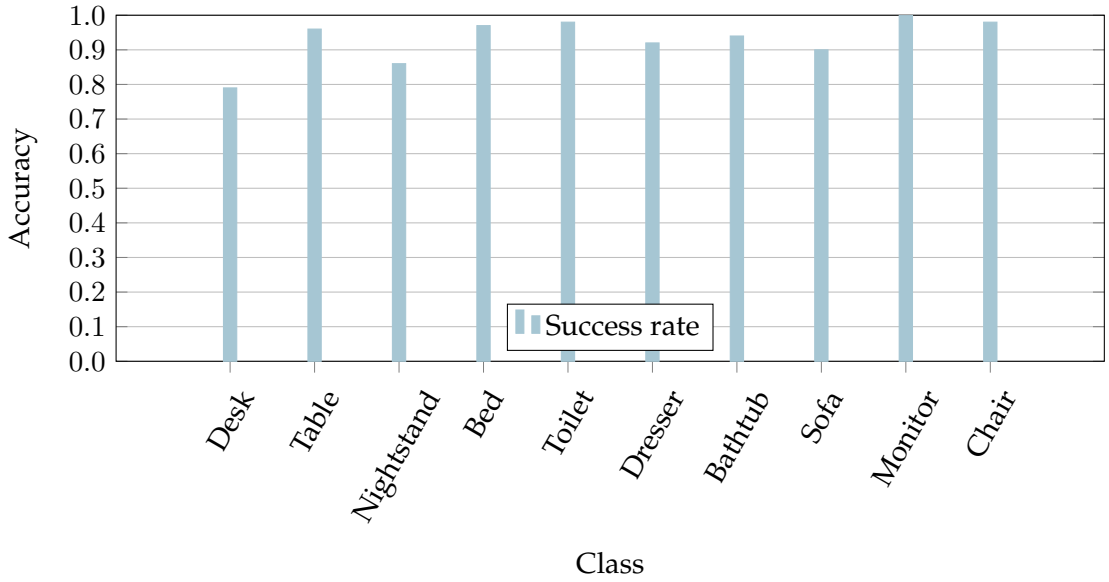


Figure 2.33: Success rate per class for the test split of the ModelNet-10 dataset achieved by LonchaNet after 18300 training iterations using the *ModelNet-10* dataset (solver type is ADAM, learning rate is 0.00001, β_1 is 0.9 and β_2 is 0.999).

the classes desk and nightstand. In light of this fact we can expect that if we could have a balanced dataset, the error rate of these classes would decrease significantly.

In Figure **??**, we can observe the confusion matrix of the classification accuracy for the test split of ModelNet-10. This table, alongside the precision-recall curves presented in Figure **??**, confirms the stability and reliability of the system, which confuses only objects of classes that heavily look alike from a visual perspective.

That is the case of the classes desk and table which the system confuses, yet not the other way around. This could be possibly caused by the fact that a desk is a type of table, meaning the desks have minor visual features that go unnoticed in some examples, making those examples hard to distinguish from the table ones, as shown in Figures **??** and **??**. In addition, in the Table **??**, it could be seen that the class desk have a reduced number of examples compared with other classes such as sofa or chair.

Also, there is a confusion problem among the classes nightstand and dresser. This kind of confusion is very common in CNN architectures because, again, they rely their classification capabilities on visual features of an object and, as seen in Figures **??** and **??**, these two classes are visually similar.

Figure **??** remarks the ambiguity of the visual features between the classes desk an table, and nightstand an dresser and point up the difficulty of the problem.

Using our current architecture, we reached the second place in the leaderboard of the challenge, with an accuracy of 94.37% in the ModelNet-10 Accuracy task. The state of the leaderboard (as of January 2017) is shown in Table **??**.

2.6.4 Conclusion

In this last iteration, we introduced a novel architecture for 3D object recognition, LonchaNet. Our system takes three slices of the input point cloud (one per 3D axis), projects the points to a plane, generating three images, and sends them to the neural network. The architecture consist in three independent GoogLeNet branches which activations are concatenated and fed to a fully connected layer. Each of this branches learns particular features of a slice.

This method allows us to take advantage of the fast 2D computation whilst preserving the 3D information. LonchaNet posits as the second place in the ModelNet

challenge with a succes rate of 94.37% in the ModelNet-10 accuracy test, yet providing a extremely fast computation: once the model is trained, classifying a 3D object only takes 0.0896 seconds.

2.7 Conclusion

Method	ModelNet10 Accuracy	ModelNet40 Accuracy
VRN Ensemble [60]	97.14%	95.54%
LonchaNet	**94.37**%	N/A
ORION [61]	93.80%	N/A
FusionNet [62]	93.11%	90.80%
Pairwise [63]	92.80%	90.70%
GIFT [64]	92.35%	83.10%
VoxNet [49]	92.00%	83.00%
3D-GAN [65]	91.00%	83.30%
DeepPano [66]	85.45%	77.63%
3DShapeNets [47]	83.50%	77.00%
MVCNN [67]	N/A	90.10%

Table 2.3: ModelNet leaderboard as of January, 2017.

Chapter 3

Semantic Segmentation

3.1 Introduction

3.2 Related Works

3.3 The RobotriX

3.4 UnrealROX

3.5 2D-3D-SeGCN

Chapter 4

Tactile Sensing

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

4.1 Introduction

When we humans grasp objects, we know whether the grip is stable or not before lifting the object. It is not necessary to raise our hands in order to check such state of the grasp. Using our tactile sense, along with our vision and other senses, we can accurately predict the stability of the grasp. This skill is desirable for any robotic manipulator since it favors the early detection of grasp failures so the robot can react in consequence: for example, a re-stocking robot working in a store would recognize when an object could slip from its hand and, therefore, avoid breaking it.

The problem of predicting the stability of a grasp is a task under research in the field of robotic grasping. In order to approach a solution to it, tactile sensors are being used as the main source of data since they provide valuable information (e.g. temperature, pressure) about the acting forces during the interaction of the robotic hand with the objects [68]. As for the stability prediction, two states are usually distinguished: stable, meaning that the object is firmly grasped; or slippery, meaning that the object could slide from the hand.

Previous works found in the literature approach this problem following the next methodology: grasp the object, read the tactile sensors equipped in the fingers and/or palm of the hand, calculate custom features that try to characterize these two stability states and learn them in order to make future predictions [69]–[72]. These proposals treat the tactile readings as classic signals: they pre-process them as if they were arrays, calculate features and learn their characteristics using probabilistic methods. As a consequence, their performance highly depends on the selected characteristics. Moreover, the spatial distribution inherent to the tactile sensor is lost due to the fact of squeezing the data into a one dimensional array.

In this chapter, we propose the use of Graph Neural Networks (GNNs) for predicting grasp stability. Since these are deep learning models, there is no need to hand-engineer features because the algorithm is designed for learning them by itself. Moreover, graphs can reflect more accurately the real distribution of the electrodes in the sensor as well as their spatial relationships, which should be of great value for learning tactile features. The main contributions of this work can be summarized as follows:

- We process tactile readings using a novel perspective: instead of considering them as 1D arrays or 2D images, we build a 3D graph connecting the multiple sensing points (taxels) of the tactile sensor.
- We introduce a novel way of processing such information using Graph Neural Networks (GNNs).
- We quantitatively check the performance of this new methodology in the real world using a set of tactile sensors installed in a robotic hand.
- We release an extension that effectively doubles the size of an already existing dataset [73] for grasp stability prediction and includes a whole new split for testing.

4.2 Related Works

In this section, we review the state of the art of the two main fields related to our work. On the one hand, we describe previous approaches for predicting grasp stability. On the other hand, we explain the most recent and relevant advances in neural networks for graph processing.

4.2.1 Grasp Stability Prediction

In the last years, deep learning models are being applied to the problem of grasp stability prediction using tactile sensors as input. Meier *et al.* [74] processed tactile readings using Fourier-related transforms and the resulting vectors were vertically stacked in order to create a matrix. Then, a CNN trained with these matrices learnt to predict stability. Although this approach used modern machine learning models, it still had to hand-engineer features.

In contrast, Cockbum *et al.* [75] proposed to use autoencoders to autonomously calculate the relevant characteristics for the task. Afterwards, a dictionary of basis features was built using a sparse encoding algorithm. Finally, the authors trained a Support Vector Machine (SVM) in order to predict grasp stability using the dictionary. Similarly, Kwiatkowski *et al.* [76] built a composite image by placing the readings of

two matrix-like sensors side by side. Then, they used this tactile image as input for a CNN along with the proprioceptive data from the robot. As a result, the proposed method calculated by itself the features needed for predicting grasp stability.

A more recent trend suggests the interpretation of tactile sensors as images in order to exploit the potential of CNN as feature learners. In some cases, vision-based sensors are used for this purpose. Calandra *et al.* [77] used a tactile sensor that contained an internal camera, which recorded the deformation of the gel inside of the sensor throughout its contact with a surface. Then, the recorded tactile images were learnt using a CNN in order to predict the grasp outcome. In some other cases, the tactile sensor is not naturally arranged in an array or it does not contain a camera, so a pre-processing is necessary in order to get a tactile image. For example, Zapata-Impata *et al.* [73] studied how the readings from a non-matrix like sensor should be arranged in a matrix in order to train a CNN for grasp stability prediction. Although such approach showed promising results, the spatial distribution of the real sensor was not accurately reflected because it reduced the 3D locations of the taxels into 2D coordinates of a tactile image.

Recently, CNNs are being combined with Long Short-Term Memory Networks (LSTMs) for grasp stability prediction. Li *et al.* [78] in their work learnt visual features from a camera-based tactile sensor, similar to the one used by Calandra *et al.* [77], and an external camera pointing to the scene. These features were calculated using a pre-trained CNN. Then, both cameras features were concatenated and passed in time sequences to a LSTM, which was in charge of detecting slippage. Similarly, Zhang *et al.* [79] used another camera-based tactile sensor for grasp stability detection but in this work the authors trained a Convolutional LSTM (ConvLSTM) and they only passed the sensor images to the network.

4.2.2 Graph Neural Networks

Lately, GNNs have emerged as a solid alternative to process irregular data which can be structured as graphs. Their original focus was tasks whose data can be expressed as graphs holding locality, stationarity, and composionality principles in general. In the literature, various works have successfully made use of this kind of architecture to deal

with unstructured 3D representations mainly in classification tasks. Most of them have proposed extensions to the well-known CNN architecture to process graph-structured data. That generalization is not trivial since various problems must be addressed when applying convolution filters in domains in which there is no regular structure. In that regard, there are two dominant ways to convolve a graph signal with a learned filter: spatial or spectral.

Spectral methods are characterized by providing a spectral graph theoretical formulation of CNNs on graphs using Graph Signal Processing (GSP) theory [80]. The fundamentals of this kind of methods rely on decomposing the graph Laplacian to form a Fourier basis via an eigendecomposition of the graph matrix, i.e., a spectral decomposition. By doing that, a convolution in the graph domain can be expressed as a multiplication in the spectral one. This kind of methods usually faces three challenges: the design of compactly supported filters, the definition of parameter sharing schemes among different graphs, and the aggregation of multi-scale information. Arguably, the most common and limiting drawback is the first challenge: filters are not directly transferable to different graphs. Since filters are learned in the context of the spectrum of the graph Laplacian, a global graph structure must be assumed. In other words, only the signals on the vertices may change, the structure of the graph must remain the same.

Spatial methods constitute the straightforward generalization of convolutions to graph, just by sliding a filter on the vertices as a traditional CNN does with any other structured data representation. Despite its simplicity, the direct application of the definition of a convolution to graphs poses two difficulties: the definition of neighborhoods, and the ordering of the nodes to form receptive fields. Because of that, one common problem of spatial methods is the difficulty to generate a weight sharing schema across graph locations due to the fact that local neighborhoods can be completely different, i.e., the number of nodes adjacent to another one varies and there is no well-defined ordering for them.

Here we briefly review the most relevant GNNs that have been successfully applied to similar problems to the one at hand.

The pioneer spectral formulation of a CNN to operate over irregular domains modeled as graphs was introduced by Bruna et al. [81]. In that work, they exploited the

global structure of the graph with the spectrum of its graph-Laplacian to extend the convolution operator. This method was applied to hand-written digit classification using the Mixed National Institute of Standards and Technology (MNIST) dataset.

Defferrard *et al.* [82] proposed strictly localized filters, which are provable to be localized in a ball of a certain radius, i.e., hops from a specific vertex. That enhancement has some other collateral effects such as improved computational complexity for the filters (linear w.r.t. the supports size and the number of edges). They also introduced an efficient pooling strategy based on a rearrangement of the vertices as a binary tree. Their approach, namely *Chebyshev Spectral Graph Convolutional Operator* or just *ChebConv*, was successfully applied and performed similarly to classical CNNs in digits classification problems such as MNIST.

Kipf and Welling [83] introduced a set of simplifications to Bruna's [81] and Defferrard's [82] formulations to improve performance and scalability in large-scale networks. They proved the efficacy of their work on transductive node classification on very large scale networks for various problems such as semi-supervised document classification in citation networks (CiteSeer, Cora and PubMed datasets) and semi-supervised entity classification in a knowledge graph (NELL dataset). As its main feature, their *GCNConv* operator takes advantage of fast localized first-order features to achieve linear scaling in the number of graph edges.

Simonovsky and Komodakis [84], inspired by the idea from Jia *et al.* [3] about dynamic filter networks, took a similar approach for solving the weight sharing problem suffered by spatial methods. They introduced Edge-Conditioned Convolutions (ECCs) in which filter weights are conditioned on edge features and generated by a generator network. That generator, usually implemented as a Multi-Layer Perceptron (MLP), outputs specific weights for each edge in the neighborhood. That method was successfully tested on point cloud classification problems (Sydney urban objects and ModelNet), a standard graph classification benchmarks, and also on MNIST.

Velickovic *et al.* [85] introduced a *Graph Attention* operator, namely *GATConv*, that leverages masked self-attentional layers to compute the hidden representations of each node in the graph, by attending over its neighbors, following a self-attention strategy. This approach addressed many of the key challenges of spectral-based methods and

achieved or surpassed state of the art methods in the aforementioned citation network datasets as well as protein interaction ones.

Fey *et al.* [86] proposed the *Spline-based Convolutional Operator*, a continuous and spatial kernel that leverages B-spline bases's properties to efficiently filter graph data of arbitrary dimensionality. They prove this method to be successful in digit image graph classification problems using MNIST and graph node classification using the Cora dataset.

Following this success in those similar domains, we intend to use a GNN to process tactile sensor readings and predict grasp stability. By doing so, we expect that such architecture is able to better capture the spatial locality and relationships of the tactile sensor readings expressed as graphs instead of other non-spatial (1D arrays) or discrete (images) representations.

4.3 TactileGCN

In this section, we describe our full approach for predicting grasp stability using tactile sensors. The whole pipeline comprises three main components:

1. A robotic setup which consists of a Shadow hand and BioTac Sp sensors, all operated by Robot Operating System (ROS).

2. A tactile graph generator which takes the sensor readings and generates a proper graph representation for the network.

3. A GNN architecture to process such graphs and predict graph stability.

4.3.1 Robotic Set Up

In this work, we use the BioTac SP tactile sensors developed by Syntouch [87]. The sensor provides three different sensory modalities: force, pressure, and temperature. In more detail, this biomimetic sensor counts with 24 electrodes, also named taxels, integrated in just a single phalanx. These electrodes record signals from four emitters in the internal core of the sensor and, therefore, they measure the impedance in the fluid located between the internal core and the external elastic skin of the sensor. The fluid is

displaced when the sensor makes contact with a surface, affecting that impedance read by the electrodes. Thus, the sensor can approximate how much pressure is being experienced at each electrode. In addition, the sensor features a hydro-acoustic pressure sensor in order to estimate a general pressure value and it also counts with a thermistor, which is used to detect vibrations and heat flows. The sensor is presented in Figure 4.1.

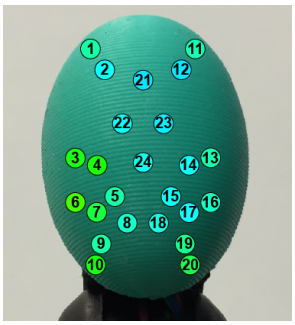


Figure 4.1: BioTac SP tactile sensor with its 24 electrodes approximated position.

For our work, we use a setup of three BioTac SP sensors in the tip of the index, middle finger, and thumb of a Shadow Dexterous robotic hand developed by the Shadow Robot Company [88]. The Shadow hand is an anthropomorphic hand with five fingers and 20 Degrees of Freedom (DoF) in total. Those features allow the robot to reach a wide range of configurations that are comparable to those of a human hand. Its integration with the BioTac SP sensors is seamless since the sensor readings can be directly obtained using the ROS [89] framework, in which the Shadow hand works.

4.3.2 Tactile Graphs

In order to feed our Graph Neural Network, we expressed the aforementioned sensor readings in a novel graph representation, namely tactile graphs. Such graphs are triplet $G = (N, E, Y)$ where N is a set of 24 nodes $n_0, ..., n_{23}$ (one for each electrode or taxel in the sensor), E is a set of ordered pair of vertices called edges, and Y is the label or class of the graph (in our case, stable or unstable).

Each node n in the graph G represents a taxel and as such, they are characterized by a 3D position $p_n = (x_n, y_n, z_n)$ and a feature vector $f_n = (f_{n_0}, ..., f_{n_F})$ of arbitrary length F.

Node positions p_n are accurately mapped to the physical taxel (X, Y, Z) coordinates within the sensor. Such positions are specified in Table 4.1. Edges or connections are generated following two different approaches: manual or k-NN. For the first approach, we manually specified undirected connections following proximity and symmetry criteria. For the second one, we generated directed edges towards each k-Nearest Neighbors for each node. Figure 4.2 shows a 3D graph representation of a tactile graph.

Table 4.1: Actual position of the taxels inside the BioTac SP sensor expressed in Cartesian coordinates (X, Y, Z) in inches.

Taxel	X (inches)	Y (inches)	Z (inches)
1	0.386434851	-0.108966104	0.156871012
2	0.318945051	-0.205042252	0.120706090
3	0.087372680	-0.128562247	0.281981384
4	0.083895199	-0.235924865	0.201566857
5	-0.018624877	-0.300117050	0.094918748
6	-0.091886816	-0.120436080	0.284956139
7	-0.136659500	-0.237549685	0.187122746
8	-0.223451775	-0.270674659	0.071536904
9	-0.320752549	-0.199498368	0.127771244
10	-0.396931929	-0.100043884	0.151565706
11	0.386434851	-0.108966104	-0.156871012
12	0.318945051	-0.205042252	-0.120706090
13	0.087372680	-0.128562247	-0.281981384
14	0.083895199	-0.235924865	-0.201566857
15	-0.018624877	-0.300117050	-0.094918748
16	-0.091886816	-0.120436080	-0.284956139
17	-0.136659500	-0.237549685	-0.187122746
18	-0.223451775	-0.270674659	-0.071536904
19	-0.320752549	-0.199498368	-0.127771244
20	-0.396931929	-0.100043884	-0.151565706
21	0.258753050	-0.252337663	0.000000000
22	0.170153841	-0.274427927	0.072909607
23	0.170153841	-0.274427927	-0.072909607
24	0.075325086	-0.298071391	0.000000000

Node features f_n correspond to the taxel pressure readings. In the case of the most basic tactile graph, each node has three features, i.e., the pressure reading for each finger: index f_{n_0}, middle f_{n_1}, and thumb f_{n_2}. Figure 4.3 shows visualizations of the three components of the feature vector for sample graphs generated with various values of $k = 0$, $k = 2$, $k = 4$, $k = 8$.

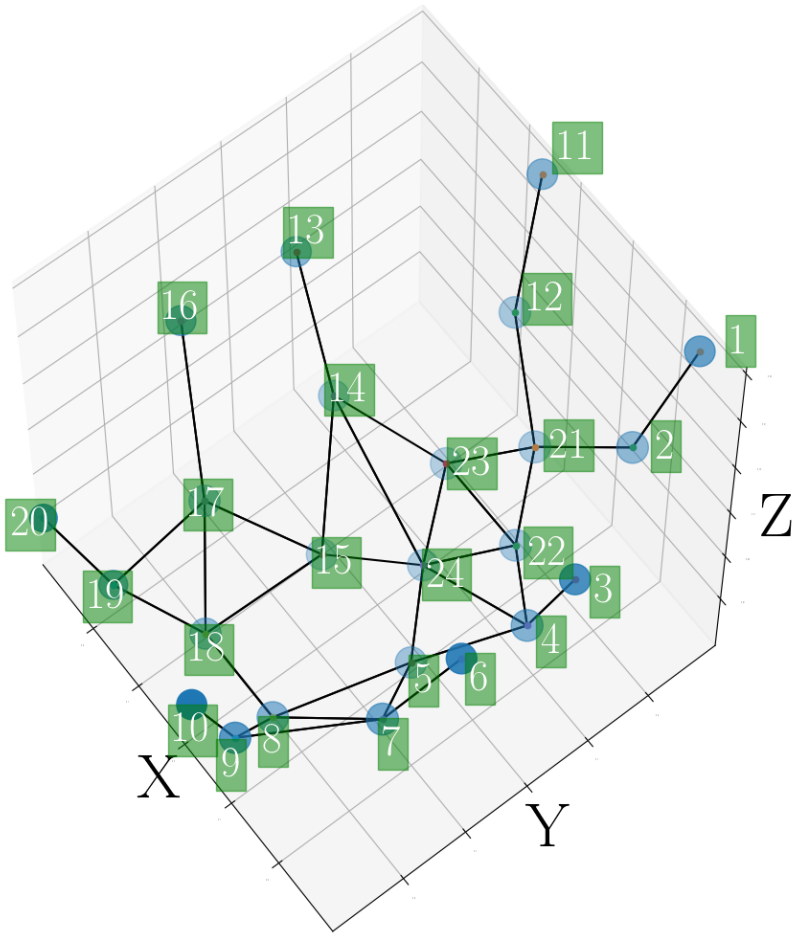


Figure 4.2: 3D visualization of the tactile graph layout using the accurate spatial arrangement from the actual BioTac SP sensor. Graph edges correspond to the manually defined connections.

4.3.3 Graph Neural Network

Our Graph Neural Network (GNN) of choice is based on the Graph Convolutional Network (GCN) model by Kipf and Welling [83]. Such model is arguably one of the most successful, yet simple, approaches to date to generalize a well-established model such as the CNN to arbitrarily structured graphs [90][91]. Their proposal, which is somewhat similar to Defferard's *et al.*, introduce a set of simplifications into a framework of spectral graph convolutions to make them train significantly faster and achieve state-of-the-art levels of accuracy across various classification tasks [82].

The goal of such models is to learn features on a graph $G = (N, E, Y)$ by taking as input a feature matrix X ($N \times F$ with a feature vector f_n for each node n) and a description of the graph structure in the shape of an adjacency matrix A (computed

from the set of edges E in the graph). The output is another feature matrix Z ($N \times F'$ with node-level feature vectors f'_n with a predefined number of output features F').

Each GCN layer $H^(l)$ in a network with L layers can be expressed as a non-linear function $H^{(l+1)} = f(H^{(l)}, A)$. The first layer takes the input feature matrix ($H^{(0)} = X$) and the final layer generates the output node-level feature matrix ($Z = H^{(L)}$). Each intermediate layer generates a node-level feature matrix $Z^{(l)}$ which is fed to the next layer. In the case of Kipf and Welling [83], the graph-convolution layer $f(H^{(l),A)}$ is defined, in the most basic instantiation, as $\sigma(AH^{(l)}W^{(l)})$, where σ is an activation function of choice and $W^{(l)}$ is the weight matrix for the l layer.

This basic framework was heavily extended to overcome two limitations: (1) unless there are explicitly defined self-loops in the graph, the multiplication of A only sums up the feature vectors of all the neighboring nodes but not the node itself, and (2) since A is not normalized by default, the multiplication of A has a huge impact on the scale of the feature vectors. Overcoming those two limitations is crucial to improve the model's convergence.

In order to fix those two limitations, they first enforced self-loops in the graph by adding the identity matrix to A so the new adjacency matrix is $\hat{A} = A + I$. Secondly, they normalized that adjacency matrix in a row-like fashion by leveraging a symmetric normalization with the diagonal node degree matrix $\hat{D}$ of $\hat{A}$. Those two improvements combined form the layer propagation rule proposed by Kipf and Welling [83]: $f(H^{(l)}, A) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$. This is the *GCNConv* operator that we used to build our GNN.

However, it is important to remark again that this model produces a feature matrix with node-level feature vectors yet our problem needs to classify the whole graph either as stable or slippery. To produce such binary graph-level classification output we need to introduce pooling operations to reduce the amount of nodes in the graph and/or fully connected layers to perform high-level reasoning.

4.4 Experiments

We conducted several experiments in order to validate our approach. In this section we describe the dataset we used to carry out such experiments. In addition, we provide all the details of our methodology to ensure the reproducibility of our procedures. At last, we discuss all the experiments that led us to the architecture described in the previous section.

4.4.1 Dataset

The dataset used in our experiments was first introduced in [73] as the *BioTac SP Images* dataset. It contains grasp samples performed over 41 objects with different geometries (i.e. cylinders, spheres, boxes), materials (i.e. wood, plastic, aluminum), stiffness levels (i.e. solid, soft) as well as sizes and weights. Those objects are shown in Figure 4.4. For this work, added 10 new objects with similar materials but different geometries and stiffness levels (see Figure 4.5). The original 41 were left for the training set whilst the new ones were separated into a test set. Both sets, training and test, were recorded following these steps:

1. **Grasp the test object:** the hand performed a three-fingered grasp that contacted the object with each of the fingers equipped with a tactile sensor.

2. **Read the sensors:** a single reading was recorded then from each of the sensors at the same time.

3. **Lift the object:** the hand was raised in order to lift the object and check the outcome.

4. **Label the trial:** the previously recorded tactile readings were labeled according to the outcome of the lifting with two classes (stable, i.e., it is completely static, or slip, i.e., either fell from the hand or it moves within it).

There are two hand configurations in the original dataset: *palm down* grasps were performed pointing the palm of the hand downwards while *palm side* grasps were recorded pointing it to one side, with the thumb upwards. In this work, we have added

a new configuration: *palm 45* which is in between the other two configurations at an angle of 45 degrees. Figure 4.6 shows the aforementioned hand configurations.

Table 4.2 provides a quantitative summary of the extended dataset for both splits and all configurations.

Table 4.2: Summary of the extended BioTac SP dataset which was used in this work to validate our graph-based architecture.

	Training Set		Test Set	
Configuration	**Stable**	**Slippery**	**Stable**	**Slippery**
Palm Down	667	609	153	163
Palm Side	603	670	157	165
Palm 45	1058	1075	250	261
All	2328	2354	560	589

To the best of our knowledge, there is only one previous work that released a dataset of tactile recordings for the task of grasp stability detection, which is the BiGS dataset [92]. In their work, Chebotar *et al.* recorded 2000 grasps on three standing objects (a cylindrically-shaped box of wipes, a cubically-shaped box of candy and a ball) using a Barret three-fingered hand, which was equipped with three BioTac tactile sensors. Our work extends the BioTac SP dataset firstly introduced in [73], counting with more than 4000 training grasps and 1000 test grasps with three BioTac SP tactile sensors recorded using 51 objects and various orientations, both for the objects and the hand. The dataset is freely available at GitHub [1].

4.4.2 Experimental Setup

All experiments were run on a computer with an i7-8700 CPU 3.20 GHz (6 cores / 12 threads) with an Z370 chipset motherboard, 16 GiB DDR4 RAM 2400 MHz CL15, a Samsung SSD 860 EVO 250 GiB, and an NVIDIA Titan X Maxwell (12 GiB) GPU. Everything was implement in Python 3.6, PyTorch 0.4.1, PyTorch Geometric 0.3.1, CUDA 10.0 (with driver version 410.73).

For most experiments, we report accuracy as our main metric to iterate and draw conclusions over training and validation sets. For the test set, we report four different metrics: accuracy, precision, recall, and F1-score (the harmonic mean of precision and

[1] https://github.com/3dperceptionlab/biotacsp-stability-set-v2

recall). To ensure generalization and give an accurate (and statistically correct) estimate of our prediction model performance we employ k-fold cross validation with $k = 5$. All reported results are the average of 10 rounds of 5-fold cross validation. For each cross-validation split, we train our models for 512 epochs using the ADAM optimizer. The hyperparameters were chosen empirically as follows: 0.01 learning rate and $5e^{-4}$ weight decay.

The whole source code and dataset for this work can be downloaded from the corresponding GitHub repository[2].

4.4.3 Network Depth and Width

In these experiments, we investigate the impact of network depth (convolution layers) and width (amount of features per layer). To that end, we have tested ten different models ranging from one to ten *GCNConv* layers with increasing number of features (8, 16, 32, 48, 64). ReLU activations were used after each convolutional layer. Two fully connected layers were also placed at the end of the network (with 128 and 2 output features respectively) to produce the classification result. We made use of the manually defined graph connections ($k = 0$). Figure 4.7 shows the results of this set of experiments.

As we can observe, there is a dependency on both width and depth. Shallow networks tend to perform better than their deep counterparts. However, we can find a sweet spot on the architecture with 5 layers and 32 features ($8 - 8 - 16 - 16 - 32$). Shallower networks are not able to fully capture our problem while deeper ones tend to overfit our training data. Consequently, we will proceed with that network.

4.4.4 Graph Connectivity

For the connectivity experiments we took the previous best network and investigated the effect of graph connectivity. We experimented with manually specified edges ($k = 0$) and the k-NN strategy with $k = [1, 23]$. As shown in Figure 4.8, the performance of the network degraded as the connectivity of the graph increased in each experiment. Using the k-NN strategy, smaller k values achieved greater performance in terms

[2] https://github.com/3dperceptionlab/tactile-gcn

of validation accuracy. However, none of them improved the performance (92.7%) yielded by the network trained with the graph created using the manual connectivity ($k = 0$).

In the manually created graph there are electrodes connected by an edge to just one other electrode, some others are connected up to four neighbors and the electrode in the center (24th electrode) is connected to six other points. As a result, there are different degrees of connectivity within the graph that could have given some insight to the network about the importance of each node in order to better learn the problem.

4.4.5 Generalization Tests

In order to prove the generalization capabilities of our system, we trained our best network ($8 - 8 - 16 - 16 - 32$ with $k = 0$) with our whole training set and evaluated it on the various test sets (palm down, palm side and palm 45). All results are reported in Table 4.3.

Table 4.3: Results of generalization experiments on the testing splits.

Test Set	Accuracy	Precision	Recall	F1
Down	0.741	0.741	0.751	0.745
45	0.774	0.774	0.783	0.778
Side	0.751	0.785	0.709	0.745

There is a significant drop in accuracy when dealing with completely unknown objects. Recall that the test set consists of new objects with different geometries and stiffness levels so they are substantially different from the training set. Taking all of this into account, and despite the difficulty of the testing set, we can expect gains from applying regularization and augmentation strategies to increase performance on data whose distribution is not that similar to the training set.

4.5 Conclusion

Tactile sensors provide useful information for robotic manipulation tasks like predicting grasp stability. Prior works in the literature tend to compute hand-engineered features that are later used for training a machine learning model. A recent trend process

them as images, so deep learning techniques like CNNs can calculate relevant characteristics that lets the system distinguish a slippery grasp from a stable one. Inspired by this methodology, we propose in this work a novel approach to tactile data interpretation: we build a graph with the sensor's taxels because this structure keeps more accurately the spatial distribution and the local connectivity of these sensing points. The goodness of these properties and the tactile graph for grasp stability prediction were tested in experimentation.

We used three BioTac SP tactile sensors mounted in the tip of the index, middle and thumb of a Shadow Dexterous hand. In order to predict grasp stability using these graph representations of the tactile sensors, we trained a GCN with a custom dataset which was captured with more than 50 objects and 3 hand orientations. The robustness of the proposed system was checked by testing the system with novel orientations and objects. In average, the GCN yielded a 92.7% validation accuracy on the prediction of grasp stability with novel objects or orientations.

4.5.1 Limitations and Future Works

Given the obtained results, graph representations of tactile readings can be successfully used for learning the task of grasp stability prediction. Nevertheless, there are some drawbacks linked to their used. The first limitation of this proposal is the problem of defining the graph connectivity. We had to find a way of defining the location of the taxels as well as their connections in order to define the graph. In the case of using the tactile readings directly, none of this is necessary.

Moreover, GCN showed to be data hungry models for learning. In a previous work [73], the authors obtained higher validation rates (94.2%) with fewer data samples for training a CNN. For this work, it was necessary to capture more data in order to achieve similar accuracy rates in training. Furthermore, generalization to radically new objects has still a lot of room for improvement by leveraging techniques such as L2 regularization, dropout, or data augmentation itself.

As a future work, we also plan to decouple the currently unified GCN for the three fingers so that each graph is processed by a different network path. Furthermore, we plan to model the noise of each individual taxel and augment each sample on the fly

by adding random noise following each taxel's distribution. At last, we plan to extend the architecture to predict grasp stability over temporal sequences by fusing the GCN model with LSTM networks.

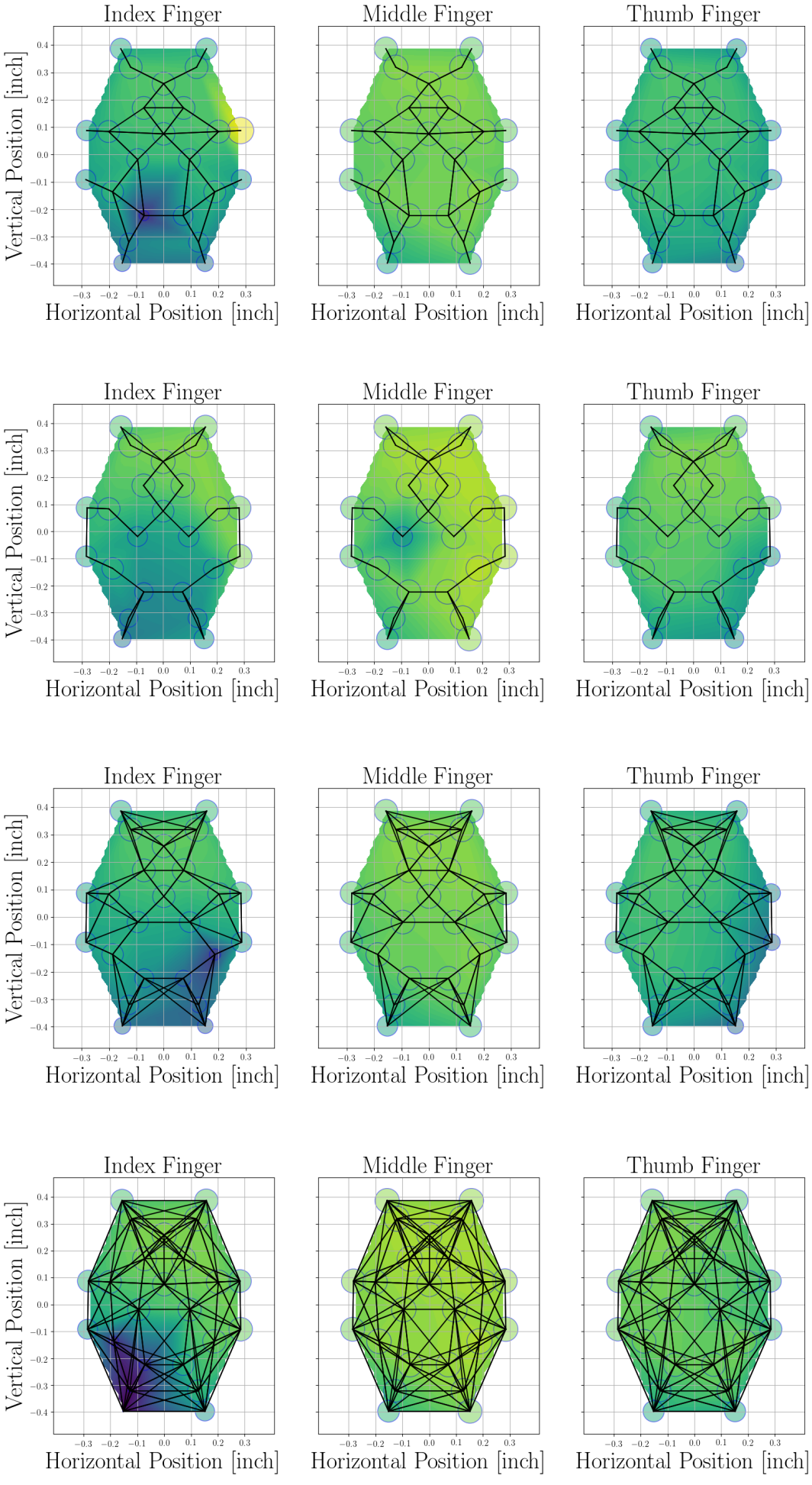

Index Finger
Middle Finger
Thumb Finger
Vertical Position [inch]
Horizontal Position [inch]
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3

Figure 4.4: The original training set of 41 objects.

Figure 4.5: The newly captured test set of 10 objects.

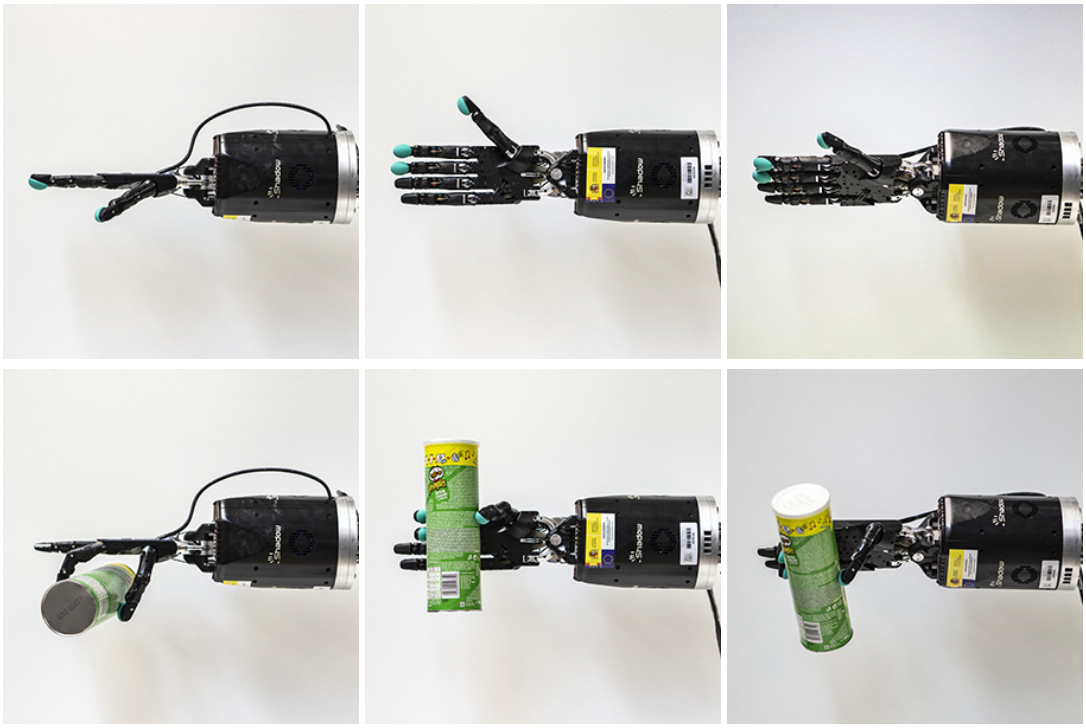


Figure 4.6: (Top row) Samples of the three hand configurations in the dataset: (from left to right) *palm down*, *palm side*, and *palm 45*. (Bottom row) The same configurations but grasping an object.

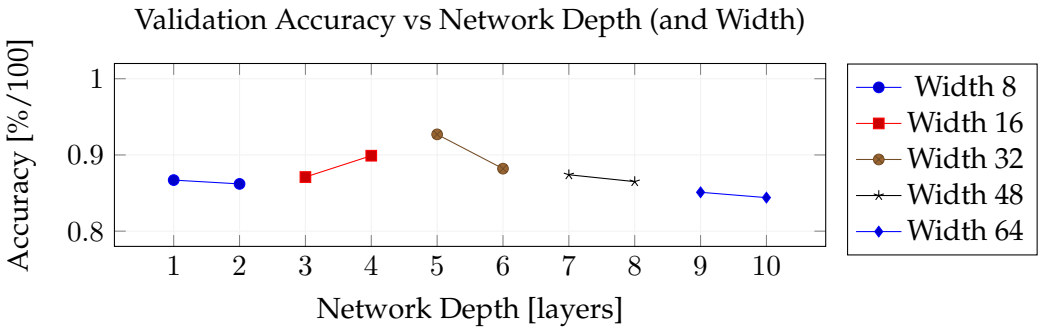


Figure 4.7: Results of network depth and width study.

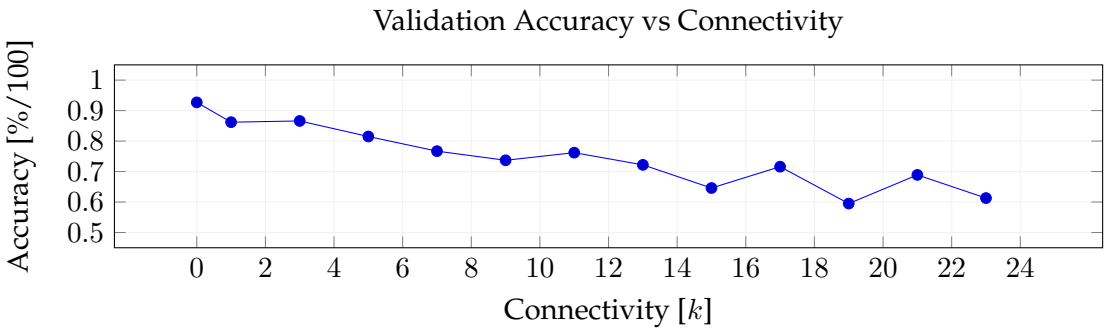


Figure 4.8: Performance of the network according to the connectivity of the graph.

Chapter 5

Conclusion

5.1 Findings and Conclusions

5.2 Limitations

5.3 Future Work

Bibliography

[1] Alberto Garcia-Garcia, Francisco Gomez-Donoso, Jose Garcia-Rodriguez, et al. "PointNet: A 3D Convolutional Neural Network for real-time object class recognition". In: *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24-29, 2016*. 2016, pp. 1578–1584. DOI: 10.1109/IJCNN.2016.7727386. URL: https://doi.org/10.1109/IJCNN.2016.7727386.

[2] Alberto Garcia-Garcia, Jose Garcia-Rodriguez, Sergio Orts-Escolano, et al. "A study of the effect of noise and occlusion on the accuracy of convolutional neural networks applied to 3D object recognition". In: *Computer Vision and Image Understanding* 164 (2017), pp. 124–134. DOI: 10.1016/j.cviu.2017.06.006. URL: https://doi.org/10.1016/j.cviu.2017.06.006.

[3] Francisco Gomez-Donoso, Alberto Garcia-Garcia, Jose Garcia-Rodriguez, et al. "LonchaNet: A Sliced-based CNN Architecture for Real-time 3D Object Recognition". In: *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, Alaska, May 14-19, 2017*. 2017. URL: https://ieeexplore.ieee.org/document/7965883/.

[4] Alberto Garcia-Garcia, Pablo Martinez-Gonzalez, Sergiu Oprea, et al. "The RobotriX: An eXtremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 6790–6797. URL: https://ieeexplore.ieee.org/abstract/document/8594495.

[5] Pablo Martinez-Gonzalez, Sergiu Oprea, Alberto Garcia-Garcia, et al. "UnrealROX: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation". In: *CoRR* abs/1810.06936 (2018). arXiv: 1810.06936. URL: http://arxiv.org/abs/1810.06936.

[6] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, et al. "A Visually Plausible Grasping System for Object Manipulation and Interaction in Virtual Reality Environments". In: *CoRR* abs/1903.05238 (2019). arXiv: 1903.05238. URL: http://arxiv.org/abs/1903.05238.

[7] Alberto Garcia-Garcia, Pablo Martinez-Gonzalez, Sergiu Oprea, et al. "The RobotriX: A Large-scale Dataset of Embodied Robots in Virtual Reality". In: *International Conference on Computer Vision and Pattern Recognition (CVPR). Workshop on 3D Scene Generation*. 2019.

[8] Alberto Garcia-Garcia, Brayan Stiven Zapata-Impata, Sergio Orts-Escolano, et al. "TactileGCN: A Graph Convolutional Network for Predicting Grasp Stability with Tactile Sensors". In: *CoRR* abs/1901.06181 (2019). arXiv: 1901.06181. URL: http://arxiv.org/abs/1901.06181.

[9] Brayan Stiven Zapata-Impata, Alberto Garcia-Garcia, Sergio Orts-Escolano, et al. "Tactile Graphs for Grasp Stability Prediction". In: *International Conference on Learning Representations (ICLR). Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[10] Sergiu Oprea, Alberto Garcia-Garcia, Jose Garcia-Rodriguez, et al. "A Recurrent Neural Network based Schaeffer Gesture Recognition System". In: *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, Alaska, May 14-19, 2017*. 2017. URL: https://ieeexplore.ieee.org/document/7965885/.

[11] Francisco Gomez-Donoso, Sergio Orts-Escolano, Alberto Garcia-Garcia, et al. "A robotic platform for customized and interactive rehabilitation of persons with disabilities". In: *Pattern Recognition Letters* 99 (2017), pp. 105–113. DOI: 10.1016/j.patrec.2017.05.027. URL: https://doi.org/10.1016/j.patrec.2017.05.027.

[12] Sergiu Oprea, Alberto GarciaGarcia, Sergio OrtsEscolano, et al. "A long short-term memory based Schaeffer gesture recognition system". In: *Expert Systems* 0.0 (2017), e12247. DOI: 10.1111/exsy.12247. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.12247.

[13] Alberto Garcia Garcia, Andreas Beckmann, and Ivo Kabadshow. "Accelerating an FMM-Based Coulomb Solver with GPUs". In: *Software for Exascale Computing-SPPEXA 2013-2015*. Springer, 2016, pp. 485–504. URL: https://link.springer.com/chapter/10.1007/978-3-319-40528-5_22.

[14] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, et al. "Multi-sensor 3D object dataset for object recognition with full pose estimation". In: *Neural Computing and Applications* 28 (2016), pp. 941–952. ISSN: 1433-3058. DOI: 10.1007/s00521-016-2224-9. URL: http://dx.doi.org/10.1007/s00521-016-2224-9.

[15] Marcelo Saval-Calvo, Jorge Azorin-Lopez, Andres Fuster-Guillo, et al. "Evaluation of sampling method effects in 3D non-rigid registration". In: *Neural Computing and Applications* 28 (2016), pp. 953–967. ISSN: 1433-3058. DOI: 10.1007/s00521-016-2258-z. URL: http://dx.doi.org/10.1007/s00521-016-2258-z.

[16] Sergio Orts-Escolano, Jose Garcia-Rodriguez, Miguel Cazorla, et al. "Bioinspired point cloud representation: 3D object tracking". In: *Neural Computing and Applications* 29 (2016), pp. 663–672. ISSN: 1433-3058. DOI: 10.1007/s00521-016-2585-0. URL: https://doi.org/10.1007/s00521-016-2585-0.

[17] Alberto Garcia-Garcia, Sergio Orts-Escolano, Jose Garcia-Rodriguez, et al. "Interactive 3D object recognition pipeline on mobile GPGPU computing platforms using low-cost RGB-D sensors". In: *Journal of Real-Time Image Processing* 14 (2016), pp. 585–604. ISSN: 1861-8219. DOI: 10.1007/s11554-016-0607-x. URL: https://doi.org/10.1007/s11554-016-0607-x.

[18] Higinio Mora, Jerónimo M Mora-Pascual, Alberto Garcia-Garcia, et al. "Computational analysis of distance operators for the iterative closest point algorithm".

In: *PloS one* 11.10 (2016), e0164694. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0164694.

[19] Sergio Orts-Escolano, Jose Garcia-Rodriguez, Vicente Morell, et al. "3D Surface Reconstruction of Noisy Point Clouds Using Growing Neural Gas: 3D Object/Scene Reconstruction". In: *Neural Processing Letters* 43 (2015), pp. 401–423. DOI: 10.1007/s11063-015-9421-x. URL: http://dx.doi.org/10.1007/s11063-015-9421-x.

[20] Sergio Orts-Escolano, Jose Garcia-Rodriguez, Jose Antonio Serra-Perez, et al. "3D model reconstruction using neural gas accelerated on GPU". In: *Applied Soft Computing* 32 (2014), pp. 87–100. DOI: 10.1016/j.asoc.2015.03.042. URL: http://dx.doi.org/10.1016/j.asoc.2015.03.042.

[21] Alexander Andreopoulos and John K. Tsotsos. "50 Years of object recognition: Directions forward". In: *Computer Vision and Image Understanding* 117.8 (2013), pp. 827–891.

[22] David G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[23] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[24] Michael Calonder, Vincent Lepetit, Christoph Strecha, et al. "Brief: Binary robust independent elementary features". In: *Computer Vision–ECCV 2010* (2010), pp. 778–792.

[25] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. "BRISK: Binary robust invariant scalable keypoints". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2548–2555.

[26] Ethan Rublee, Vincent Rabaud, Kurt Konolige, et al. "ORB: an efficient alternative to SIFT or SURF". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2564–2571.

[27] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. "FREAK: Fast retina keypoint". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. Ieee. 2012, pp. 510–517.

[28] David G. Lowe. "Object recognition from local scale-invariant features". In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[29] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, et al. "3D object recognition in cluttered scenes with local surface features: A survey". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 36.11 (2014), pp. 2270–2287.

[30] Jean Ponce, Svetlana Lazebnik, Fredrick Rothganger, et al. "Toward true 3D object recognition". In: *Reconnaissance de Formes et Intelligence Artificielle*. 2004.

[31] Paul J. Besl and Ramesh C. Jain. "Three-dimensional object recognition". In: *ACM Computing Surveys (CSUR)* 17.1 (1985), pp. 75–145.

[32] Jim P. Brady, Nagaraj Nandhakumar, and Jake K. Aggarwal. "Recent progress in object recognition from range data". In: *image and vision computing* 7.4 (1989), pp. 295–307.

[33] Farshid Arman and Jake K. Aggarwal. "Model-based object recognition in dense-range imagesa review". In: *ACM Computing Surveys (CSUR)* 25.1 (1993), pp. 5–43.

[34] Richard J. Campbell and Patrick J. Flynn. "A survey of free-form object representation and recognition techniques". In: *Computer Vision and Image Understanding* 81.2 (2001), pp. 166–210.

[35] George Mamic and Mohammed Bennamoun. "Representation and recognition of 3D free-form objects". In: *Digital Signal Processing* 12.1 (2002), pp. 47–76.

[36] Yann Le Cun, Yoshua Bengio, and Geoffrey E. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[37] Paul J. Werbos. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences". PhD thesis. Harvard University, 1974.

[38] Yann Le Cun. "A learning scheme for asymmetric threshold networks". In: *Proceedings of Cognitiva* 85 (1985), pp. 599–604.

[39] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5 (1988), p. 3.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[41] Stjepan Marčelja. "Mathematical description of the responses of simple cortical cells". In: *JOSA* 70.11 (1980), pp. 1297–1300.

[42] Alex Berg, Jia Deng, and Fei-Fei Li. *ImageNet large scale visual recognition challenge 2010*. 2010.

[43] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[44] Yoshua Bengio, Pascal Lamblin, Dan Popovici, et al. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems* 19 (2007), p. 153.

[45] Pierre Sermanet, Koray Kavukcuoglu, Sandhya Chintala, et al. "Pedestrian detection with unsupervised multi-stage feature learning". In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 3626–3633.

[46] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. "Large-scale deep unsupervised learning using graphics processors". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 873–880.

[47] Zhirong Wu, Shuran Song, Aditya Khosla, et al. "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015. URL: http://arxiv.org/abs/1406.5670.

[48] Shuran Song and Jianxiong Xiao. "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images". In: *CoRR* abs/1511.02300 (2015). URL: http://arxiv.org/abs/1511.02300.

[49] Daniel Maturana and Sebastian Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 922–928.

[50] Kevin Lai, Liefeng Bo, Xiaofeng Ren, et al. "A large-scale hierarchical multi-view rgb-d object dataset". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1817–1824.

[51] Ashutosh Singh, Jin Sha, Karthik S. Narayan, et al. "Bigbird: A large-scale 3d database of object instances". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 509–516.

[52] Bo Li, Yijuan Lu, Chunyuan Li, et al. "Shrec14 track: extended large scale sketch-based 3D shape retrieval". In: *Eurographics Workshop on 3D Object Retrieval*. 2014, pp. 121–130.

[53] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, et al. "A Large Dataset of Object Scans". In: *CoRR* abs/1602.02481 (2016).

[54] Radu B. Rusu. "Point Cloud Library". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 1–4.

[55] Aitor Aldoma, Zoltan-Csaba Marton, Federico Tombari, et al. "Tutorial: Point cloud library: Three-dimensional object recognition and 6 dof pose estimation". In: *IEEE Robotics & Automation Magazine* 19.3 (2012), pp. 80–91.

[56] Yangqing Jia, Evan Shelhamer, Jeff Donahue, et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the ACM International Conference on Multimedia*. 2014, pp. 657–678.

[57] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, et al. "cuDNN: Efficient Primitives for Deep Learning". In: (2014), pp. 1–9. ISSN: 08876266. DOI: 10.1002/polb.23894. eprint: 1410.0759.

[58] Alexandre Dalyac, Murray Shanahan, and Jack Kelly. "Tackling Class Imbalance with Deep Convolutional Neural Networks". In: *Imperial College* (2014), pp. 30–35.

[59] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, et al. "BlenSor: blender sensor simulation toolbox". In: *Advances in Visual Computing*. Springer, 2011, pp. 199–208.

[60] Andrew Brock, Theodore Lim, JM Ritchie, et al. "Generative and Discriminative Voxel Modeling with Convolutional Neural Networks". In: *arXiv preprint arXiv:1608.04236* (2016).

[61] Nima Sedaghat, Mohammadreza Zolfaghari, and Thomas Brox. "Orientation-boosted Voxel Nets for 3D Object Recognition". In: *arXiv preprint arXiv:1604.03351* (2016).

[62] Vishakh Hegde and Reza Zadeh. "FusionNet: 3D Object Classification Using Multiple Data Representations". In: *CoRR* abs/1607.05695 (2016). URL: http://arxiv.org/abs/1607.05695.

[63] Edward Johns, Stefan Leutenegger, and Andrew J Davison. "Pairwise Decomposition of Image Sequences for Active Multi-View Recognition". In: *arXiv preprint arXiv:1605.08359* (2016).

[64] Song Bai, Xiang Bai, Zhichao Zhou, et al. "Gift: A real-time and scalable 3d shape search engine". In: *arXiv preprint arXiv:1604.01879* (2016).

[65] Jiajun Wu, Chengkai Zhang, Tianfan Xue, et al. "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling". In: *arXiv preprint arXiv:1610.07584* (2016).

[66] Baoguang Shi, Song Bai, Zhichao Zhou, et al. "Deeppano: Deep panoramic representation for 3-d shape recognition". In: *IEEE Signal Processing Letters* 22.12 (2015), pp. 2339–2343.

[67] Hang Su, Subhransu Maji, Evangelos Kalogerakis, et al. "Multi-view convolutional neural networks for 3d shape recognition". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 945–953.

[68] Zhanat Kappassov, Juan-Antonio Corrales, and Véronique Perdereau. "Tactile sensing in dexterous robot hands Review". In: *Robotics and Autonomous Systems* 74 (2015), pp. 195–220. ISSN: 09218890. DOI: 10.1016/j.robot.2015.07.015. URL: http://dx.doi.org/10.1016/j.robot.2015.07.015http://linkinghub.elsevier.com/retrieve/pii/S0921889015001621.

[69] Miao Li, Yasemin Bekiroglu, Danica Kragic, et al. "Learning of grasp adaptation through experience and tactile sensing". In: *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 3339–3346. ISBN: 978-1-4799-6934-0. DOI: 10.1109/IROS.2014.6943027. URL: http://ieeexplore.ieee.org/document/6943027/.

[70] Hao Dang and Peter K. Allen. "Stable grasping under pose uncertainty using tactile feedback". In: *Autonomous Robots* 36.4 (2014), pp. 309–330. ISSN: 0929-5593. DOI: 10.1007/s10514-013-9355-y. URL: http://link.springer.com/10.1007/s10514-013-9355-y.

[71] Zhe Su, Karol Hausman, Yevgen Chebotar, et al. "Force estimation and slip detection/classification for grip control using a biomimetic tactile sensor". In: *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE, 2015, pp. 297–303. ISBN: 978-1-4799-6885-5. DOI: 10.1109/HUMANOIDS.2015.7363558. URL: http://ieeexplore.ieee.org/document/7363558/.

[72] Filipe Veiga, Herke van Hoof, Jan Peters, et al. "Stabilizing novel objects by learning to predict tactile slip". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. Vol. 2015-Decem. IEEE, 2015, pp. 5065–5072. ISBN: 978-1-4799-9994-1. DOI: 10.1109/IROS.2015.7354090. URL: http://ieeexplore.ieee.org/document/7354090/.

[73] Brayan S. Zapata-Impata, Pablo Gil, and Fernando Torres Medina. "Non-Matrix Tactile Sensors: How Can Be Exploited Their Local Connectivity For Predicting Grasp Stability?" In: *IEEE/RSJ IROS 2018 Workshop RoboTac: New Progress in Tactile Perception and Learning in Robotics* abs/1809.05551 (2018). arXiv: 1809.05551. URL: http://arxiv.org/abs/1809.05551.

[74] Martin Meier, Florian Patzelt, Robert Haschke, et al. "Tactile Convolutional Networks for Online Slip and Rotation Detection". In: *25th International Conference on Artificial Neural Networks*. Ed. by Alessandro E.P. Villa, Paolo Masulli, and Antonio Javier Pons Rivero. Vol. 9887. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 12–19. DOI: 10.1007/978-3-319-

44781-0_2. URL: http://link.springer.com/10.1007/978-3-319-44781-0{_}2.

[75] Deen Cockbum, Jean-philippe Roberge, Thuy-hong-loan Le, et al. "Grasp stability assessment through unsupervised feature learning of tactile images". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2238–2244. ISBN: 978-1-5090-4633-1. DOI: 10.1109/ICRA.2017.7989257. URL: http://ieeexplore.ieee.org/document/7989257/.

[76] Jennifer Kwiatkowski, Deen Cockburn, and Vincent Duchaine. "Grasp stability assessment through the fusion of proprioception and tactile signals using convolutional neural networks". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 286–292. ISBN: 978-1-5386-2682-5. DOI: 10.1109/IROS.2017.8202170. URL: http://ieeexplore.ieee.org/document/8202170/.

[77] Roberto Calandra, Andrew Owens, Manu Upadhyaya, et al. "The Feeling of Success: Does Touch Sensing Help Predict Grasp Outcomes?" In: *Proceedings of the 1st Annual Conference on Robot Learning*. Vol. 78. 2017, pp. 314–323. URL: http://proceedings.mlr.press/v78/calandra17a.html.

[78] Jianhua Li, Siyuan Dong, and Edward Adelson. "Slip Detection with Combined Tactile and Visual Information". In: (2018). arXiv: 1802.10153. URL: http://arxiv.org/abs/1802.10153.

[79] Yazhan Zhang, Zicheng Kan, Yu Alexander Tse, et al. "FingerVision Tactile Sensor Design and Slip Detection Using Convolutional LSTM Network". In: (2018). arXiv: 1810.02653. URL: http://arxiv.org/abs/1810.02653.

[80] David I Shuman, Sunil K Narang, Pascal Frossard, et al. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 83–98.

[81] Joan Bruna, Wojciech Zaremba, Arthur Szlam, et al. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[82] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3844–3852.

[83] Thomas N Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[84] Martin Simonovsky and Nikos Komodakis. "Dynamic edgeconditioned filters in convolutional neural networks on graphs". In: *Proc. CVPR*. 2017.

[85] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* 1.2 (2017).

[86] Matthias Fey, Jan Eric Lenssen, Frank Weichert, et al. "SplineCNN: Fast geometric deep learning with continuous B-spline kernels". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 869–877.

[87] Syntouch. *BioTac SP*. https://www.syntouchinc.com/en/sensor-technology/. 2018.

[88] Shadow Robot Company. *Shadow Dexterous Hand*. http://www.shadowrobot.com/products/dexterous-hand/. 2018.

[89] Morgan Quigley, Ken Conley, Brian P. Gerkey, et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5.

[90] Michael M Bronstein, Joan Bruna, Yann LeCun, et al. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[91] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, et al. "Modeling relational data with graph convolutional networks". In: *European Semantic Web Conference*. Springer. 2018, pp. 593–607.

[92] Yevgen Chebotar, Karol Hausman, Zhe Su, et al. "BiGS: BioTac Grasp Stability Dataset". In: *ICRA 2016 Workshop on Grasping and Manipulation Datasets*. 2016. DOI: 10.1038/nrn2621..